



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

**Sistema de visión artificial y
control para un
microlaboratorio a bordo de
un nanosatélite**

TESIS

Que para obtener el título de
Ingeniera Mecatrónica

P R E S E N T A

Rubí Janet Núñez Dorantes

DIRECTOR DE TESIS

Dr. Saúl De La Rosa Nieves



Ciudad Universitaria, CDMX, 2017



Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradecimientos

Dedico esta tesis y todo el esfuerzo que en ella se concentra a mi madre, es ella el motor de mi vida y la fuente inagotable de apoyo a quien sin más palabras le debo la vida y la dicha de haber concluido mis estudios profesionales, y a la memoria de mi padre, de quien conservo los más bellos recuerdos de mi niñez y quien siempre es y será mi más grande inspiración.

Agradezco a mis hermanas, Eli y Fabi, las adoro con todo mi corazón y admiro profundamente su desempeño profesional. Su comprensión, apoyo y tutoría forman parte de quién soy y por consecuencia se ven reflejadas en cada parte de mi vida personal y profesional.

Durante mi formación académica he tenido la fortuna de conocer a profesores, compañeros y diferentes personas cuya contribución formó las bases de mi educación y por ello merecen todo mi reconocimiento, pues considero que cada uno de mis logros es en realidad el resultado de un trabajo en equipo de todos aquellos que me aprecian o que amablemente me guían y con quienes estaré siempre en deuda. Resulta imposible mencionarlos a todos, por lo cual me disculpo recordando que tienen toda mi gratitud y aprecio.

Quiero agradecer especialmente a mis amigos, Amaury, Onasis y Marco, de quienes siempre recibí todo el apoyo y motivación para ver realizadas mis metas, incluyendo la culminación de esta tesis.

A José María y Ana, quienes me cobijaron en mi primera etapa de ejercicio profesional, y a quienes agradezco su tutoría, caracterizada por la más grande empatía. Reconozco enormemente su contribución para la sociedad al tener una empresa que pondera siempre la calidad humana de quienes en ella trabajamos.

Al Dr. Saúl De La Rosa, su tutoría y amistad son un tesoro invaluable que nunca podré agradecer por completo. Gracias por creer en mí, por trabajar hombro a hombro conmigo para asesorar este proyecto haciendo que yo misma explorara mis capacidades, y sobre todo gracias por su intensa lucha para abrir el panorama de oportunidades de quienes tenemos la fortuna de conocerlo.

Investigación realizada como parte del proyecto “*An autonomous microlaboratory for biological experimentation in microgravity*” gracias al financiamiento del Consejo Nacional de Ciencia y Tecnología (CONACYT).

Índice de contenidos

Agradecimientos	2
I. INTRODUCCIÓN	7
I.1 Problema de investigación.....	7
I.2 Pregunta de investigación.....	7
I.3 Objetivos.....	7
I.4. Hipótesis	8
II. MISION “MICROLAB”	9
II.1 Descripción general de la Misión científica	9
II.2. Misión del sistema de visión artificial.....	11
II.3 Discusión del análisis de la imagen desde el punto de vista del algoritmo de visión	14
II.4 Determinación de parámetros de operación	14
III. ESTADO DEL ARTE DE EXPERIMENTACIÓN BIOLÓGICA EN EL ESPACIO.....	16
III.1 Gene-Sat 1.....	16
II.1.a. Misión:.....	16
III.1.b. Especificaciones generales de la plataforma:.....	16
III.1.c. Sistema óptico:	18
III.2 PharmaSat	19
III.2.a. Misión	19
III.2.b. Especificaciones generales de la plataforma:.....	20
III.2.c. Sistema óptico:	21
III.3 O/OREOS	21
III.3.a. Misión	21
III.3.b. Especificaciones generales de la plataforma.....	22
III.3.c. Sistema óptico	22
III.4 Conclusiones del capítulo	24
IV. DISEÑO DEL ALGORITMO DE VISIÓN ARTIFICIAL	25
IV.1 Metodología.....	25
IV.2. Marco teórico.....	26
IV.2.a. Introducción a la Visión Artificial	26
IV.2.b. Etapas de un sistema de visión artificial.....	29

IV.2.c. Adquisición y representación de imágenes digitales	30
IV.2.d. Relaciones básicas entre píxeles	31
IV.2.e. Procesamiento digital de la imagen.....	33
IV.2.f. Principios de evaluación de desempeño	42
IV.3 Diseño y análisis de las propuestas de solución.....	43
IV.3.a. Elección de resolución espacial y radiométrica	43
IV.3.b. Planteamiento de soluciones	45
IV.3.c. Determinación de si ya hay célula en la trampa (propuestas y comparación entre algoritmos)	49
IV.3.d. Monitoreo de movimiento y crecimiento (propuesta y pruebas al algoritmo).....	73
IV.3.e. Determinación y clasificación de reproducción de la célula (propuestas y comparación entre algoritmos)	85
V. INTEGRACIÓN DEL SISTEMA E IMPLEMENTACIÓN DE LOS MÓDULOS EN FPGA	96
V.1. Propuesta de diseño de la arquitectura para el Algoritmo de Visión Artificial.....	96
Proceso “¿Ya hay célula?”:.....	97
Proceso “Monitoreo de centros”:	99
Proceso “¿Se Reproduce?”:	100
V.2. Integración del Sistema	101
V.2.a. Primera integración de los módulos del Sistema de Visión, orientada a facilitar la implementación en hardware reconfigurable	101
V.2.b. Comprobación del funcionamiento del sistema	132
V.3. Simulación de los módulos de procesamiento en VHDL	133
V.3.a. Método “¿Ya Hay?”:.....	133
V.3.b. Método “Monitoreo de Centros”:.....	138
V.3.c. Método “¿Se Reproduce?”:	140
V.4. Definición de requerimientos de procesamiento y memoria.....	142
VI. CONCLUSIONES Y TRABAJO A FUTURO.....	148
Conclusiones:.....	148
Trabajo a futuro.....	149
VII. REFERENCIAS.....	151

Resumen

La presente tesis describe el desarrollo de un Sistema de Visión Artificial para el análisis autónomo de esperanza de vida replicativa bajo los efectos de microgravedad en células eucariotas, el cual será albergado como carga útil de la misión MicroLab en un nanosatélite bajo el estándar CubeSat 3U, y ofrece una solución de innovación tecnológica en el uso de visión computacional para experimentación biológica en el espacio.

Este Sistema de Visión Artificial satisface las características particulares del análisis por el que se pretende obtener los resultados del experimento en la carga útil. La arquitectura que lo conforma cuenta con tres módulos de procesamiento de imagen basados en técnicas de sustracción de fondo en imágenes binarias. Cada uno de estos módulos es controlado por un algoritmo que interpreta los resultados obtenidos y a su vez retroalimenta al algoritmo de máximo nivel de abstracción que controla cada proceso del análisis y de acuerdo al estado de la experimentación biológica determina la acción siguiente del sistema.

La propuesta planteada ofrece una solución a la medida, en la que se obtuvo una arquitectura orientada a satisfacer los requerimientos de desempeño y consumo energético y que por requerimiento del proyecto fue diseñada específicamente para su implementación en FPGA, debido a que éste es un dispositivo de alto rendimiento al permitir procesamientos en paralelo, y por ser de hardware reconfigurable hace posible modificar total o parcialmente los elementos descritos en él cuando sea necesario, lo que hace a los FPGAs sumamente atractivos para la aplicación de técnicas de tolerancia a fallas, factor de diseño primordial e indispensable en sistemas de calidad espacial.

I. INTRODUCCIÓN

I.1 Problema de investigación

Se necesita desarrollar un sistema autónomo capaz de analizar la longevidad de la levadura *saccharomyces cerevisiae* bajo efectos de microgravedad, de cuya población en estudio se modifica genéticamente a un sector y se compara su mortandad con respecto a la de la población que no fue modificada, para esto se hace un análisis cada cierto tiempo para caracterizar el comportamiento de las células, lo cual puede ser visualizado en imágenes capturadas por una cámara digital con ayuda de filtros ópticos y cuantificado a través de algoritmos de procesamiento digital de imágenes para determinar el comportamiento de cada cepa.

I.2 Pregunta de investigación

¿Es posible desarrollar un sistema de visión artificial a bordo de un nanosatélite, que sea capaz de retroalimentar a un sistema de control, implementados en un FPGA, para analizar de manera autónoma el experimento que caracterice la longevidad de la levadura *saccharomyces cerevisiae* bajo efectos de microgravedad?

I.3 Objetivos

Generales:

- Desarrollar un sistema de visión artificial y control de procesos cuyo diseño esté orientado en la obtención y manejo de datos a través de la toma de imágenes de un dispositivo microfluídico para el análisis del envejecimiento celular, mediante el uso de componentes COTS, empleando algoritmos de reconocimiento, detección y evaluación, en un sistema de visión artificial y control a bordo de un nanosatélite, implementado en hardware reconfigurable (FPGA), lo cual representaría una técnica de innovación tecnológica en experimentación biológica en el espacio.

Particulares:

- Proponer una solución para el análisis y monitoreo del experimento que resulte más adecuada en dimensiones y consumo de energía que las reportadas en el estado de arte.
- Diseñar una arquitectura de procesamiento de alto desempeño en FPGA, la cual cumpla los requerimientos del sistema de visión.

- Diseñar e implementar los algoritmos de reconocimiento, detección, evaluación, seguimiento y estimación del sistema de visión artificial.
- Lograr un análisis cuantitativo de los organismos en estudio.
- Diseñar el algoritmo de control de procesos a partir de la interpretación de las imágenes tomadas a las levaduras.
- Integrar y validar el correcto funcionamiento del sistema de visión y su algoritmo de control.

1.4. Hipótesis

Es posible hacer un sistema autónomo para el estudio de la longevidad de células eucariotas bajo efectos de microgravedad mediante técnicas de visión artificial, tales como procesamiento digital de imágenes y reconocimiento de patrones, cuyo análisis, al retroalimentar a un sistema de control de procesos que interprete los resultados y tome decisiones basadas en un análisis previo, puede conformar un sistema mecatrónico de experimentación biológica innovador en el desarrollo de tecnología espacial.

II. MISION “MICROLAB”

II.1 Descripción general de la Misión científica

La existencia del vector de gravedad es un factor físico básico y permanente en nuestro planeta, que ha estado presente en toda la evolución de las formas de vida. Como es conocido, los procesos biológicos se ven afectados por la existencia de esta fuerza y su ausencia desencadena una respuesta celular, cuyo conocimiento es necesario para una mejor comprensión de los organismos vivos, lo que contribuye también en la investigación para conseguir la supervivencia de organismos terrestres en condiciones distintas de las de la Tierra, lo cual es uno de los grandes objetivos movilizadores de las iniciativas de exploración espacial actualmente en curso, como la Estación Espacial Internacional (ISS) o los proyectos de exploración de Marte o la Luna.[1]

En este contexto, este proyecto se define como la iniciativa de estudiar las alteraciones celulares inducidas por la ausencia de gravedad en células eucariotas, más precisamente en la levadura *saccharomyces cerevisiae*, donde se plantea un experimento que será realizado durante la misión “MICROLAB”, en la que se pretende diseñar y construir un nanosatélite de 3U que contenga un laboratorio autónomo para experimentación biológica con la participación del grupo interdisciplinario que conforman el Laboratorio de Biomicrofluidos del Cinvestav-Monterrey, el Taller de Óptica de la BUAP y el Laboratorio de Ingeniería Espacial de la Facultad de Ingeniería de la UNAM.

El objetivo general de la misión es el estudio de esperanza de vida replicativa bajo los efectos de la microgravedad en células eucariotas, con el fin de obtener una mayor comprensión de los mecanismos biológicos.

Se eligió a la levadura *saccharomyces cerevisiae*, de cuya población en estudio se generan cepas con diferentes modificaciones genéticas y se compara la cantidad de células hijas producidas por una célula madre con respecto a la población que no fue modificada en un determinado lapso. Cada cultivo experimental se encuentra en un dispositivo microfluídico, que consta de 100-500 trampas en donde serán atrapadas las células a analizar como se muestra en la Figura 1 [2] Descripción del experimento. Donde vemos en la sección A el dispositivo microfluídico con la dirección del flujo y cómo se atrapan las células al ser arrastradas hacia las trampas. En la sección B vemos la generación del brote de la célula madre.

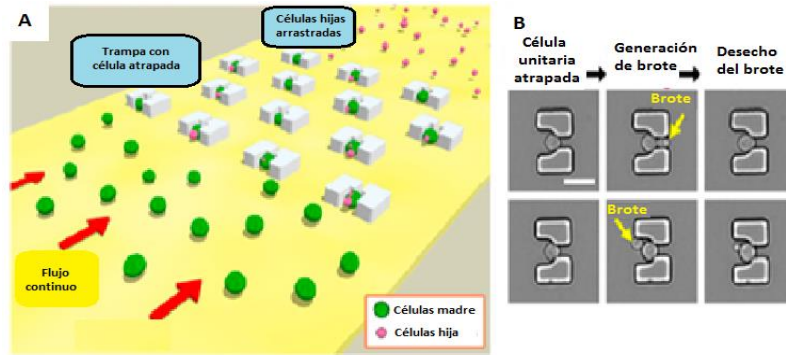


Figura 1 [2] Descripción del experimento. Donde vemos en la sección A el dispositivo microfluídico con la dirección del flujo y cómo se atrapan las células al ser arrastradas hacia las trampas. En la sección B vemos la generación del brote de la célula madre.

Cada sistema microfluídico cuenta con válvulas para la administración de nutrientes y desechos, los canales asociados y el dispositivo a analizar están dispuestos en un arreglo como el mostrado en la Figura 2.

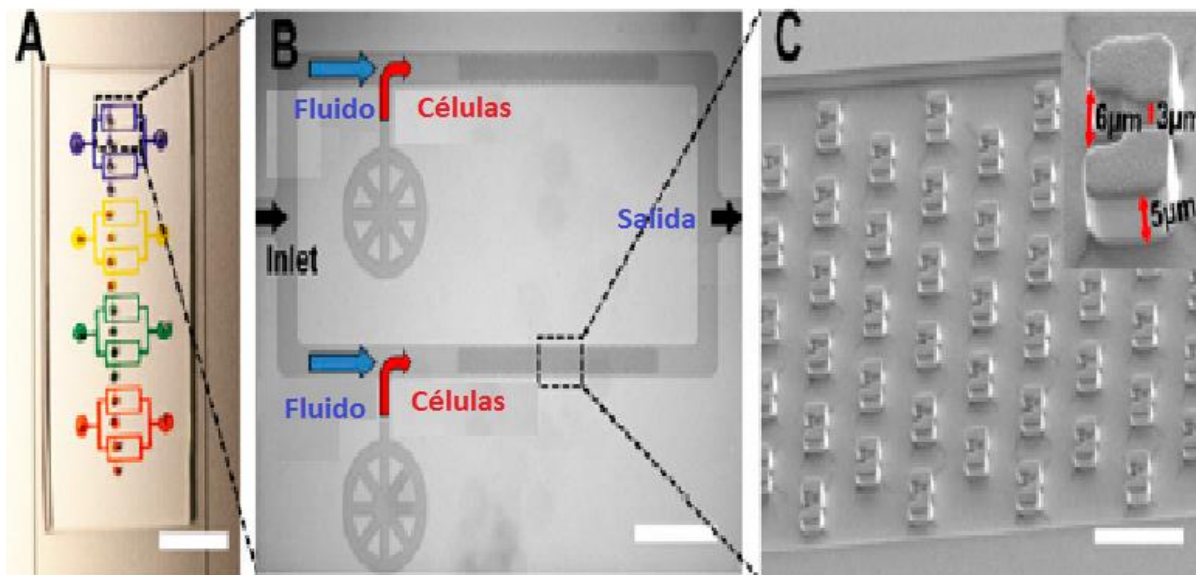


Figura 2 [2] Disposición de elementos microfluídicos. En la sección A Se muestra un diagrama general de cómo estarán dispuestos los diferentes sistemas de control de válvulas para la distribución de nutrientes a los dispositivos microfluídicos, cada uno de los cuales contendrá un cultivo de células diferente. En la sección B se muestran las válvulas que permiten el paso de las células al fluido que posteriormente en la sección C pasa al dispositivo microfluídico para llevar a cabo el experimento, se muestran también en esta última sección las dimensiones de las trampas en micrómetros.

A lo largo del experimento se abrirán las diferentes válvulas para pasar el fluido a través del dispositivo microfluídico en una sola dirección, primero con un flujo de 15ul/min, hasta que se logre atrapar al menos 100 células individuales en las trampas, disminuyendo en este momento el flujo a 5 ul/min.

Una vez que se hayan atrapado la cantidad mínima de células, el análisis de longevidad a través de esperanza de vida replicativa debe comenzar, para lo cual en la presente tesis se propone el desarrollo

de un sistema de visión artificial con el fin de hacer el análisis a bordo, lo cual tiene varias ventajas en la reducción de recursos, pues si el análisis tuviera que hacerse en Tierra, transmitir las imágenes demandaría un sistema de telecomunicaciones complejo, que además de requerir más energía para la transmisión, necesita un sistema de orientación de alta precisión, dispositivos que en su conjunto elevan costos en diseño, componentes, volumen, energía, peso, etc.

II.2. Misión del sistema de visión artificial

En la actualidad, experimentos similares han sido llevados a cabo exitosamente de forma autónoma en el espacio, en misiones como Genesat (2006) [3], Pharmasat (2009) [7] y O/OREOS [11]. En el caso de los nanosatélites Genesat y Pharmasat, se utilizaron proteínas fluorescentes para marcar a los diferentes organismos y estudiar así sus procesos biológicos. Sin embargo, para hacer el análisis de fluorescencia, sus sistemas contaban con transductores de intensidad a frecuencia.

Debido a la necesidad en la misión MICROLAB de hacer un análisis cualitativo de cada célula, en esta tesis se propone un sistema de visión artificial, pues si bien es cierto que la implementación con un transductor ha resultado más sencilla en el caso de los nanosatélites similares ya lanzados, para los requerimientos de esta misión resulta imposible utilizar esa técnica.

En esta tesis se propone que estos sistemas ópticos a base de transductores, sean reemplazados por un sistema de visión artificial que conste de una cámara y un sistema óptico para el estudio de cada dispositivo, lo cual permite un análisis de las características individuales de los organismos en estudio, además de que el uso de técnicas de visión artificial representaría una innovación tecnológica en experimentación biológica en el espacio.

Durante esta misión se requiere monitorear los cultivos cada 10 minutos, por lo que se toma una fotografía de cada dispositivo con resolución requerida de 5Mpx. De esta imagen, mediante un algoritmo de visión artificial, se detectarán las características de envejecimiento celular. Para lograr este monitoreo existen dos métodos de análisis de imagen, que explicaremos a continuación.

Análisis por fluorescencia

Este método consiste en aplicar a las células un marcador biológico basado en proteínas fluorescentes. Para esta misión se propusieron los marcadores CFP (Cyan Fluorescent Protein) y RFP (Red Fluorescent Protein). Con esta técnica la célula absorbe la proteína fluorescente la cual, al ser excitada con el espectro adecuado según el tipo de marcador, resalta las estructuras subcelulares de las levaduras, lo cual puede ser visualizado por una cámara digital con ayuda de filtros ópticos. De esta manera podemos monitorear la célula, así como determinar cuándo se está reproduciendo.

Existe suficiente documentación acerca del uso de esta técnica en distintos organismos, particularmente el comportamiento de la levadura *saccharomyces cerevisiae* está extensamente estudiado,

lo que nos permite tener una gráfica de las características de la célula durante su ciclo de vida como podemos ver en la Figura 3.

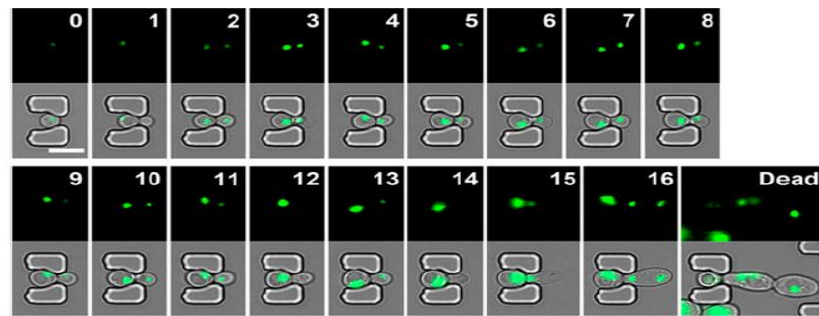


Figura 3 [2] Ciclo de vida de la célula visto en imágenes por fluorescencia. En las fotografías mostradas en la parte superior vemos la proteína fluorescente durante el ciclo de vida de la célula, y en las representaciones en gris mostradas en la parte inferior podemos observar la localización de la proteína en cada levadura. Los índices indican el intervalo al que pertenece la imagen de la célula madre atrapada hasta su muerte.

El método de estudio de células individuales por imágenes de fluorescencia durante todo el proceso de envejecimiento permite un examen de alta resolución espacio-temporal y de alto rendimiento del fenotipo de envejecimiento.

Análisis con imágenes de Campo Claro

En este método se obtienen las imágenes en una escala de grises a través de un microscopio, como se puede ver en la Figura 4 se observan contornos bien definidos que permiten diferenciar los límites de las estructuras. El umbral de los elementos es muy parecido, por lo que para hacer una identificación de las células se hace necesario el uso de alguna técnica de reconocimiento de patrones, cuyo propósito sería fundamentalmente diferenciar la estructura perteneciente a los distintos tipos de célula (madre, hija, célula de no interés o ruido) y la estructura de la trampa.

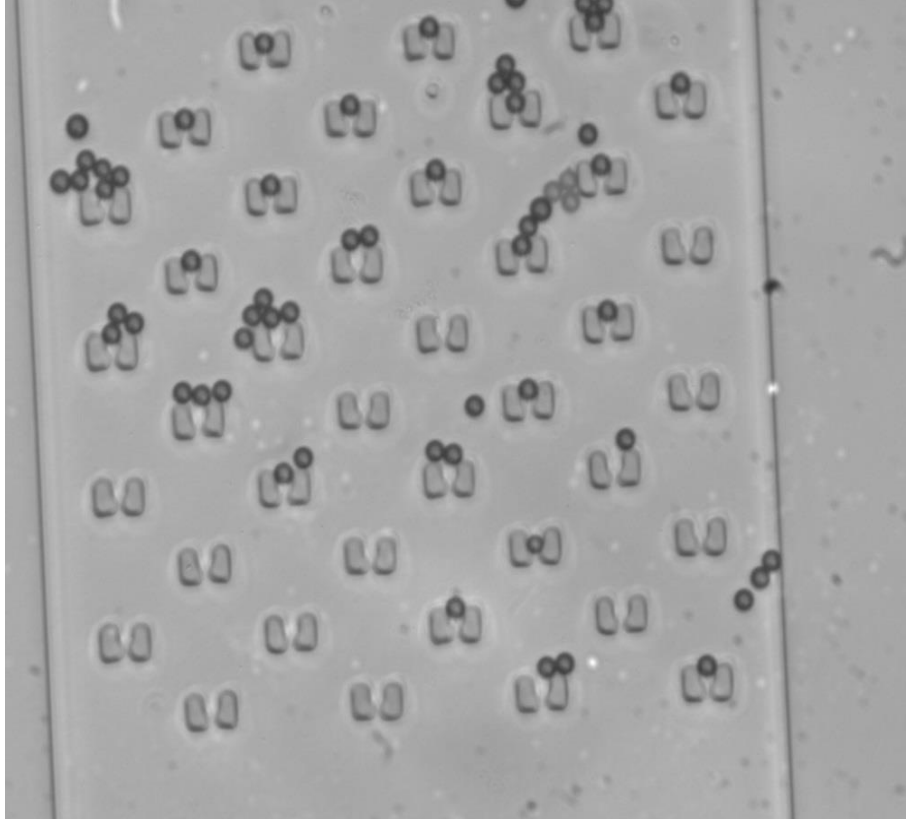


Figura 4 [4] Imagen tomada con la técnica de Campo Claro.

Cuando la célula comienza a reproducirse se observa en la imagen una figura irregular, de contornos también definidos que puede ser caracterizada por su elongación dependiendo del punto de crecimiento en el que se encuentre, como podemos ver en la Figura 5.

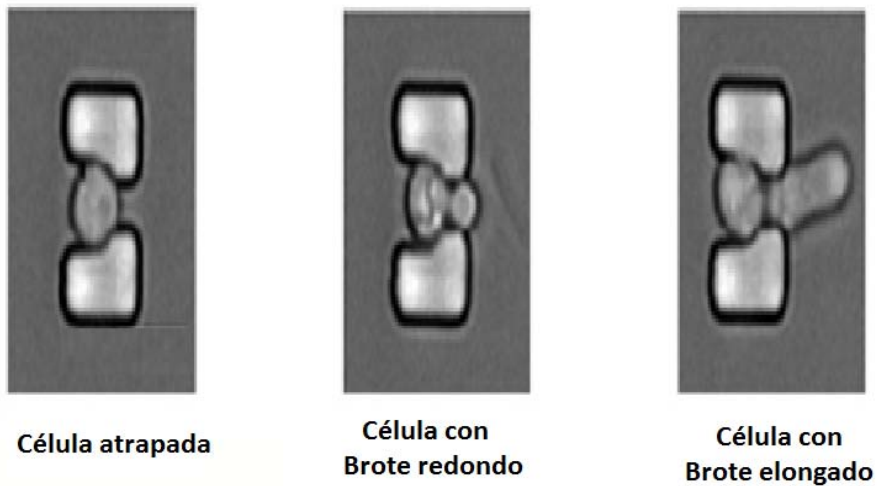


Figura 5 [2] Diferencias morfológicas de la reproducción de la célula vistas en campo claro.

II.3 Discusión del análisis de la imagen desde el punto de vista del algoritmo de visión

Las imágenes por fluorescencia ofrecen un contraste alto entre los pixeles pertenecientes a las células y los que representan el fondo de la imagen, lo que permite que el algoritmo pueda ser implementado a partir de una binarización de datos y después obtener un análisis más preciso desarrollando alguna técnica de aprendizaje estadístico como primera propuesta para conocer el tamaño promedio de cada célula y la distancia a la que se empieza a generar una célula hija, todo lo cual puede ser descrito directamente en el FPGA, es decir, se generaría una arquitectura a la medida.

Las imágenes obtenidas por campo claro tienen un contraste alto, pero dejan ver otras estructuras que no pertenecen a las células de interés, por lo que para diferenciar a los organismos del entorno una binarización por umbral no es suficiente. Notamos que la forma de células está bien definida por lo cual podría ser implementado un algoritmo de detección de patrones, los cuales generalmente utilizan transformadas como la de Hough y algoritmos de corrección que requieren un análisis más complejo que el utilizado en la técnica de fluorescencia, por lo que es conveniente analizar la posibilidad de programarlo en un microcontrolador que también puede ser embebido en el FPGA, o utilizar módulos de propiedad intelectual en conjunto con módulos propios (codiseño) que adapten la arquitectura a las necesidades de la misión. Para el análisis de la célula hija con imagen de campo claro, vemos que comienza a formarse una figura irregular y se presenta una ruptura en el borde de la célula, la cual puede ser analizada estadísticamente si se quisieran obtener datos de monitoreo más específicos.

Una vez discutidas las observaciones de lo que implicaría el análisis por cada una de las diferentes técnicas, se consultó con los equipos de Óptica y Microfluídica y se decidió hacer el análisis con imágenes de Campo Claro.

II.4 Determinación de parámetros de operación

Para el diseño del sistema de visión se necesitó determinar los parámetros de operación, para lo cual se consultó al equipo de Microfluídica, quien es el responsable de la propuesta de la carga útil del nanosatélite, así como al equipo de óptica para la realización del banco de pruebas para la validación del algoritmo.

Las condiciones de experimentación (temperatura, humedad, presión, etc.), estarán controladas durante el tiempo que dura la misión, es decir, que el único parámetro que hará una diferencia ambiental con respecto a las condiciones a las que estarán sujetas las levaduras en la experimentación en la Tierra, es la microgravedad. Para el sistema de visión es necesario que no haya fuentes de iluminación externas además de los LEDs de excitación y las fuentes que lleguen a requerir la toma de imagen, dicho en otras palabras, las condiciones de iluminación en las que se obtienen las imágenes deben ser constantes.

La toma y análisis de fotografía será cada 10 minutos, con lo que se busca obtener cuántas células fueron generadas por una célula madre en el tiempo que dura la misión, lo que indica cómo es el crecimiento y mortandad de cada cepa modificada genéticamente con respecto a la cepa salvaje, de lo cual con los datos obtenidos se puede generar una gráfica como se muestra en la Figura 6.

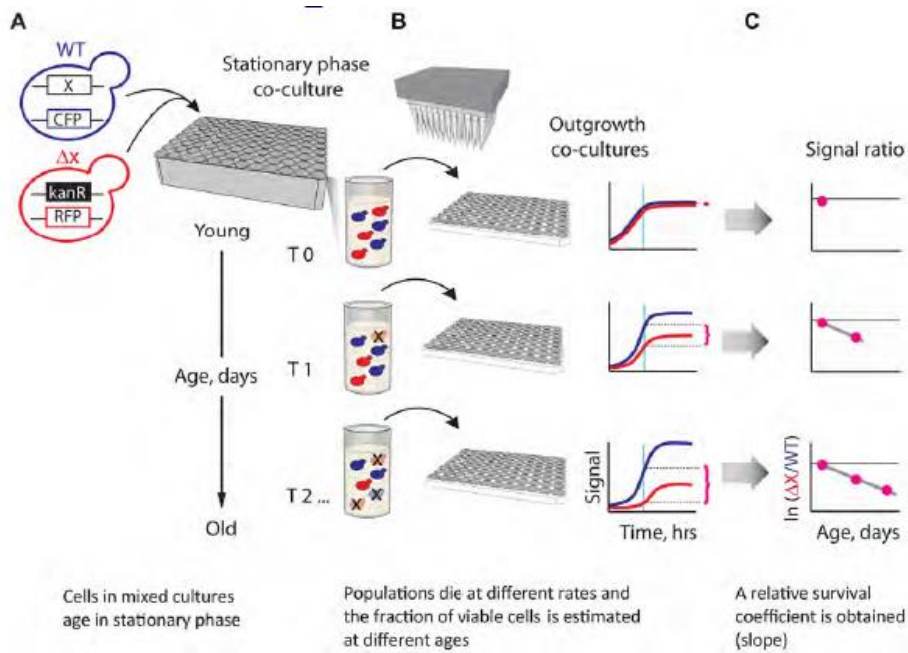


Figura 6 [18] Comparación de mortandad entre diferentes células de levadura.

Es importante mencionar que debido al carácter multidisciplinario de este proyecto, el diseño del dispositivo microfluídico aún no es definitivo y puede cambiar ligeramente en cuanto a dimensiones o disposición de las trampas a lo largo del diseño de la carga útil por parte del Laboratorio de Microfluídica, por lo que será conveniente que este factor de adaptabilidad se contemple en el diseño del sistema de visión.

III. ESTADO DEL ARTE DE EXPERIMENTACIÓN BIOLÓGICA EN EL ESPACIO

III.1 Gene-Sat 1

III.1.a. Misión:

El Gene-Sat 1 es un nanosatélite totalmente autónomo que mantuvo el crecimiento de microorganismos en múltiples fluidos, monitoreando distintos genes a través de fluorescencia. Fue lanzado el 16 de Diciembre del 2006 como carga secundaria a bordo del cohete espacial Minotaur I.

En el GeneSat-1 se incluyeron celdas solares, un módulo de control y distribución de la energía, baterías y comunicaciones bidireccionales, además de un contenedor de presión que alojó a los subsistemas de biofluidos, óptica, control térmico y sensores. La estabilidad de la temperatura, presión, y humedad relativa fueron monitoreadas por múltiples sensores, así como los eventos de radiación y aceleración en tres ejes. [5]

En esta misión se eligió a la bacteria *E. coli*, debido a su bien caracterizada capacidad de permanecer inactiva en períodos de días a meses en un rango bastante amplio de temperatura, lo cual representa una característica importante en la selección de microorganismos para experimentación biológica en el espacio debido al tiempo que deben sobrevivir entre el lanzamiento del nanosatélite que los aloja y el inicio del experimento, sin que su inactividad comprometa o altere los resultados del estudio. Los datos fueron transmitidos por radio a la Tierra en un lapso de varias semanas, durante y después del proceso de análisis del crecimiento biológico.

III.1.b. Especificaciones generales de la plataforma:

El Gene-Sat 1, mostrado en la Figura 7, tiene un peso de 4.4 kg en una estructura de $10 \times 10 \times 30 \text{ cm}^3$, que incluye cuatro celdas solares de triple unión, una estructura metálica para soportar todos los sistemas, un sistema de energía/procesamiento/comunicación/control conocido como “bus”, un contenedor presurizado con un volumen interno de 0.9 L para la carga útil [6] y una antena autodesplegable de 2.4 GHz con la que se transmitieron ~200 kB/día, con una demanda energética de 1W. [3]

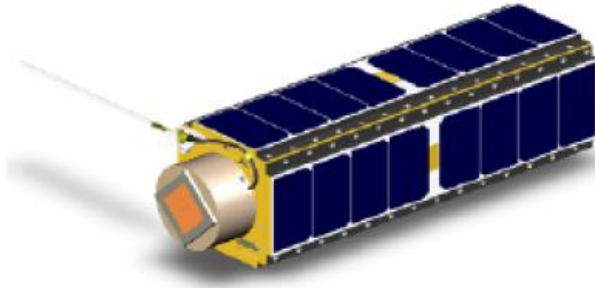


Figura 7 [6] Estructura del Gene-Sat 1.

El sistema de comando y manejo de información a bordo se realizó con un PIC18F672 [5] que tiene una demanda aproximada de 11 mW, y el sistema de control de orientación fue pasivo con una varilla magnética.

La carga útil del Gene-Sat 1 se encontraba en un contenedor cilíndrico presurizado que albergaba al experimento. Dentro de este cilindro estaba el dispositivo microfluídico, mostrado en la Figura 8, el cual constaba de doce muestras de *E. coli* y el equipo de soporte necesario para incubar y caracterizar el experimento durante 96 horas.



Figura 8 [6] Carga útil del Gene-Sat 1: dispositivo microfluídico.

El equipo de soporte incluyó toda una red de microfluidos, componentes de control de temperatura y sensores ópticos [2]. La carga energética total del GeneSat-1 fue de 4-5 W.

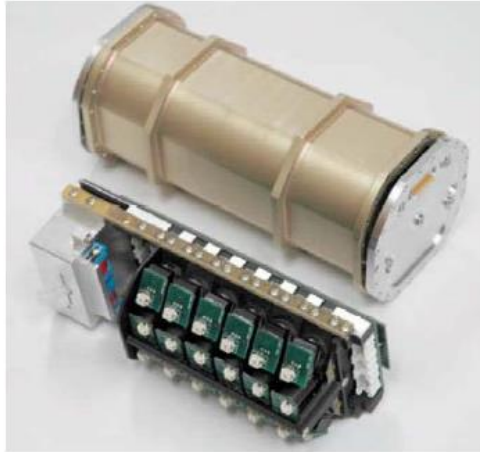


Figura 9 [6] En la imagen se observa la estructura metálica y el contenedor presurizado, así como la electrónica del “bus”.

III.1.c. Sistema óptico:

En los sistemas espaciales es recomendable prescindir de elementos móviles, por lo que en el Gene-Sat 1 se utilizó un subensamble óptico para cada uno de los 12 pozos. Cada subensamble consta de un LED azul de 3-W de excitación fluorescente (470 nm), cuyo espectro fue enfocado y filtrado por un par de lentes y un filtro de excitación (Chroma). Para captar la emisión del espectro, se utilizó un filtro de 525 nm y un par de lentes para coleccionar y enfocar la fluorescencia hacia un dispositivo conversor de intensidad a frecuencia, con lo que se logró la medición indirecta de la fluorescencia en cada pozo.

El procedimiento anterior fue realizado nuevamente utilizando esta vez un LED verde de 2.3 mW, con lo que se midió la dispersión de luz (135° *forward-scatter*), que es directamente proporcional al tamaño de la población [3], dato que fue usado para normalizar las mediciones de fluorescencia.

Los *LEDs* azul y verde se energizaron secuencialmente y las mediciones fueron hechas en un pozo a la vez para eliminar el efecto de *crossstalk*, el cual es un fenómeno indeseable en el que una señal genera una perturbación a otra, en este caso la señal es la emisión de luz de cada espectro. El diagrama de un subensamble óptico es mostrado en la Figura 10.

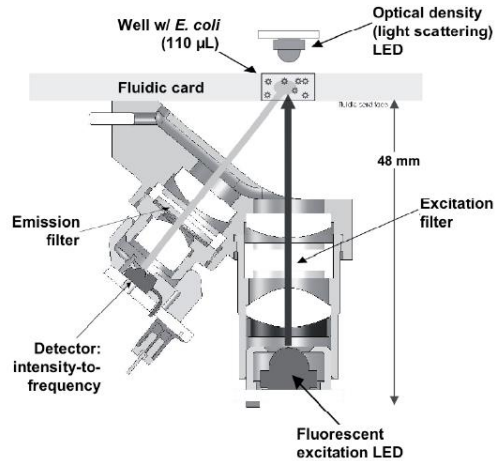


Figura 10 [3] Subensamble óptico. Vemos de arriba hacia abajo el LED de iluminación, el dispositivo microfluídico, el filtro óptico de emisión y el detector de intensidad a frecuencia del lado derecho y dl lado izquierdo el filtro de excitación y el LED de fluorescencia.

Uno de los grandes retos de la misión Gene-Sat 1 fue llevar a cabo los objetivos en un CubeSat de 10 cm x 10 cm x 30 cm de volumen y menor a 5 kg de masa. Esto implicó esfuerzos de desarrollo significativos para miniaturizar los sistemas de sensores ópticos. Finalmente cada uno de los doce subensambles ópticos ocuparon un volumen $<30 \text{ cm}^3$. [3]

III.2 PharmaSat

III.2.a. Misión

El PharmaSat es un nanosatélite de 3U y peso aproximado de 10 libras. Contiene un microlaboratorio de ambiente controlado lleno de sensores y sistemas ópticos capaces de detectar el crecimiento, la densidad y la salud de células de levadura y transmitir esos datos para su análisis en la Tierra. [8]

El nanosatélite PharmaSat se basó en la exitosa misión de la NASA GeneSat-1, y fue lanzado el 19 de mayo del 2009 a bordo del cohete Minotaur I, teniendo como objetivo investigar la eficacia de los agentes antifúngicos en el medio ambiente espacial. El nanosatélite se construyó a partir de componentes COTS (Commercial of the Shelf), así como de partes diseñadas por la NASA, para albergar un laboratorio de ciencia espacial totalmente autónomo con técnicas innovadoras de control de potencia, sensores para monitorear los niveles de presión, temperatura y aceleración, y un sistema de comunicaciones capaz de transmitir datos a la Tierra para su análisis científico. [8]

Para lograr el objetivo, el PharmaSat caracterizó el crecimiento de múltiples cultivos de levadura, que fueron expuestos a diferentes niveles de agentes antimicóticos durante su ciclo de crecimiento.

III.2.b. Especificaciones generales de la plataforma:

El PharmaSat, mostrado en la Figura 4, consiste en un módulo de bus (1U de volumen) y la carga útil (2U de volumen). El satélite completo tiene un volumen aproximado de 100mm x 100mm x 340mm y pesa menos de 5 kg [9]. El bus incluye celdas solares, una batería, un sistema de procesamiento basado en PIC, control de orientación pasivo con una varilla magnética y un transmisor-receptor de 2.4 GHz.

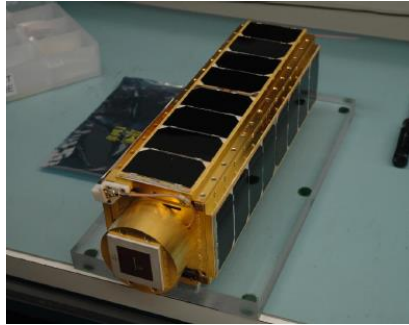


Figura 11 [9] Estructura del nanosatélite PharmaSat.

La carga útil del PharmaSat está contenida en un cilindro presurizado, cuyo módulo incluye:

- Un dispositivo microfluídico de 4 x 8 in, mostrado en la Figura 12, el cual se compone de 48 pozos en los que se depositan los cultivos de levaduras.
- Un sistema microfluídico de válvulas, bombas y canales que alimentan a las levaduras y dosifican los agentes antimicóticos.
- Un sensor óptico para detectar la salud y tamaño de la población.
- Un sistema de control del medio ambiente y control de energía.

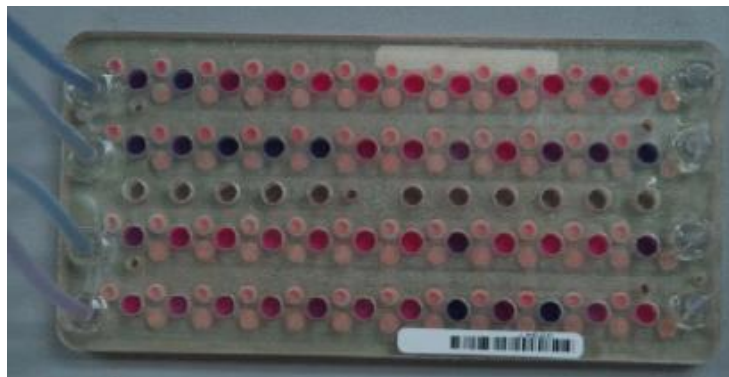


Figura 12 [7] Dispositivo microfluídico del PharmaSat.

III.2.c. Sistema óptico:

Para este estudio se utilizó el agente *Alamar Blue*, que daba una coloración entre azul y rosa a las levaduras dependiendo de su consumo de nutrientes [8], lo cual fue monitoreado a través de un sensor óptico mostrado en la Figura 13, cuya metodología de funcionamiento [9] consiste principalmente en lo siguiente:

1. Un LED tricolor ilumina a través de cada pozo.
2. Las levaduras absorben la luz.
3. Un sensor mide la absorción de cada espectro RGB.
4. El espectro verde indica el OD (conteo de población).
5. El espectro azul-rojo indica el metabolismo.

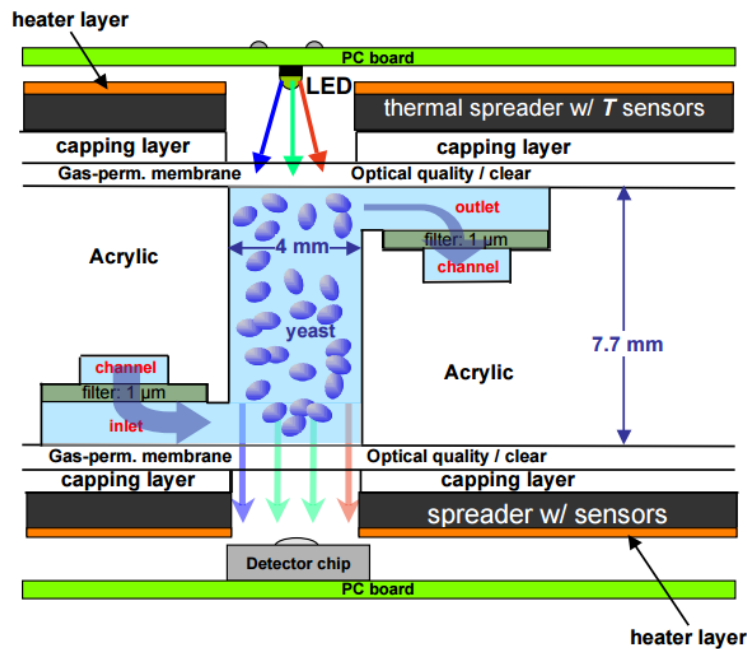


Figura 13 [9] Arquitectura fluidica/térmica/óptica.

III.3 O/OREOS

III.3.a. Misión

El O/OREOS (Organismos/Orgánicos Expuestos a Fuerzas Orbitales) es un nanosatélite lanzado el 19 de Noviembre del 2010. Los principales objetivos de esta plataforma incluyeron llevar a cabo experimentos de astrobiología en dos cargas útiles independientes dentro de un estándar 3U. Estas cargas útiles

contienen experimentos que evalúan la viabilidad de diferentes organismos en el ambiente espacial (SESLO por sus siglas en inglés) y la estabilidad de moléculas orgánicas en el espacio (SEVO). [12]

El nanosatélite fue puesto exitosamente en una órbita de alta inclinación (72°), a 650 km, lo cual significa un aumento en la exposición a partículas atrapadas en los cinturones *Van Allen* comparado con las que afectan a la Estación Espacial Internacional, asimismo la nave fue expuesta a radiación ultravioleta y rayos cósmicos. Estudiar microorganismos vivos y moléculas orgánicas complejas bajo estas condiciones es de un interés muy grande para la ciencia.

Los antecedentes a esta misión fueron los nanosatélites GeneSat-1 y PharmaSat, sin embargo esta misión fue la primera diseñada para operar durante seis meses, por lo que el equipo de desarrollo tuvo que aumentar las técnicas de tolerancia a fallas y las pruebas de radiación y blindaje.

III.3.b. Especificaciones generales de la plataforma

Cada carga útil de la plataforma incluye su propia electrónica, microcontrolador, y almacenamiento de datos para ejecutar autónomamente sus respectivos experimentos.

El bus empleó un microcontrolador PIC que se comunica con los microcontroladores de las cargas secundarias a través del protocolo I2C. La estructura del nanosatélite está hecha de aluminio y tiene montadas las celdas solares que lo abastecen de energía. Las baterías para almacenamiento y energía que se usaron son de ión de Litio.

III.3.c. Sistema óptico

SESLO

Ésta carga útil estudia a dos tipos de organismos: *Halorubrum chaoviatoris* y esporas de *Bacillus subtilis*. Estos especímenes están bajo condiciones de presión y humedad que permiten su crecimiento, el cual es medido a través de absorbancia y densidad con un sensor óptico. Los datos de crecimiento son almacenados en una memoria y transmitidos a Tierra.

Al igual que en los satélites PharmaSat y GeneSat un LED tricolor ilumina cada pozo y el detector en el lado opuesto lee la intensidad con la que es emitida el espectro. Lo anterior se muestra en la Figura 14.

El crecimiento de los microorganismos es cuantificado en tiempo real para ambos especímenes en tres longitudes de onda: 470, 525 y 615 nm. [12]

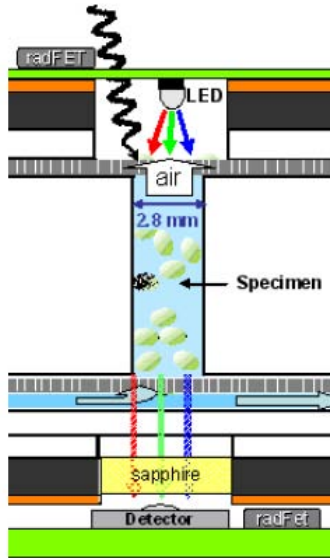


Figura 14 [12] Sistema óptico de la misión O/OREOS.

SEVO

Esta carga útil pretende investigar la estabilidad, modificación y degradación de moléculas orgánicas en distintos modelos ambientales incluyendo la exposición a las condiciones del espacio interplanetario e interestelar, para lo que se estudian los efectos de su exposición a los rayos UV, luz visible y radiación espacial a través de espectroscopia ultravioleta-visible.

Esta carga útil incluye además del espectrómetro un arreglo óptico que permite utilizar la luz del Sol como fuente de luz para el espectroscopio. [12] De esta carga útil cabe destacar que fue utilizado un carrusel que permite analizar 24 muestras con el mismo instrumento de medición el cual se muestra en la Figura 15.



Figura 15 [12] Carrusel para el análisis de muestras de la carga útil SEVO.

III.4 Conclusiones del capítulo

La presente documentación de los diferentes sistemas que se han puesto en órbita es de gran ayuda como antecedente para el diseño del sistema propuesto en esta tesis, pues aunque no hay precedentes de Sistemas de Visión Artificial en nanosatélites, los datos técnicos nos muestran que por las características de cada misión las demandas tanto energéticas como de componentes se deben adaptar a los recursos disponibles, donde podemos destacar que a pesar de que en las soluciones que se han presentado se utilizan microprocesadores de bajo consumo energético (lo cual es ideal para las condiciones en el nanosatélite), éstos no podrían realizar el procesamiento de imágenes que requiere la misión MicroLab, por lo que se hace necesario aportar soluciones innovadoras en el ámbito aunque ello demande un costo energético y computacional mayor, todo lo cual debe ser incluido en el diseño para ajustarse a las características de la plataforma.

IV. DISEÑO DEL ALGORITMO DE VISIÓN ARTIFICIAL

IV.1 Metodología

Para el diseño del algoritmo de visión se utilizara la siguiente metodología:

1. Determinación de los requerimientos del algoritmo de visión.

En esta etapa se hizo una discusión entre los equipos de Microfluídica, Óptica e Ingeniería Espacial, con el fin de determinar los requerimientos, alcances y limitaciones de la plataforma y del algoritmo de visión.

2. Revisión documental de los conceptos útiles para implementar el algoritmo.

Se hizo una investigación de los conceptos teóricos que pueden ser implementados en el desarrollo del algoritmo de procesamiento digital de la imagen y de interpretación de características extraídas.

3. Hacer un primer planteamiento de solución.

Se hizo una primera propuesta de solución para cada uno de los problemas a resolver, las cuales se compararon con otras soluciones para seleccionar la que mejor se adecúe a las características de implementación y necesidades del sistema.

4. Desarrollar la solución en MATLAB.

En una primera etapa el algoritmo se desarrollará en MATLAB para analizar resultados y hacer comparaciones entre los distintos métodos.

5. Hacer una primera validación del algoritmo.

Se plantean diferentes escenarios para poner a prueba la robustez del algoritmo y sacar un porcentaje de error con el fin de ver si se valida como solución o se requiere una mejora.

6. Correcciones.

Se hace una retroalimentación con los resultados obtenidos al correr el algoritmo de visión en distintos casos y compararlos con los resultados esperados, después de lo cual se proponen mejoras a la solución.

7. Implementación.

Se hace la descripción e integración del algoritmo de visión y control en hardware reconfigurable.

IV.2. Marco teórico

IV.2.a. Introducción a la Visión Artificial

La visión artificial tiene como finalidad la extracción de información del mundo que nos rodea a través de la obtención, procesamiento, análisis y comprensión de imágenes, utilizando para ello una computadora.

Un sistema de visión artificial actúa sobre una representación de la realidad que le proporciona información sobre brillo, colores, formas, etc. Estas representaciones suelen estar en forma de imágenes estáticas, escenas tridimensionales o imágenes en movimiento.

Una imagen bidimensional es una función que a cada par de coordenadas (x,y) asocia un valor relativo a alguna propiedad del punto que representa (por ejemplo su brillo o su matiz). Una imagen acromática, sin información de color, en la que a cada punto se le asocia la información relativa al brillo, se puede representar como una superficie en la cual la altura de cada punto indica su nivel de brillo. Una imagen a color en formato RGB se puede representar asociando a cada punto una terna de valores que indica la intensidad de tres linternas (una roja, una verde y una azul), y por último una imagen de espectro completo se puede representar asociando a cada punto un diagrama espectral de emisión de color.

Definiciones

En este apartado se discuten algunos conceptos físicos y fisiológicos imprescindibles para entender el por qué se toman algunas decisiones de diseño al construir los sistemas de visión computacional.

En general desde el punto de vista de procesamiento digital de imágenes, basta considerar la luz como una onda. Según el modelo ondulatorio las características de un rayo de luz vienen dadas por dos propiedades: su amplitud y su longitud. A pesar de que las ondas luminosas constituyen una fracción muy pequeña del conjunto de ondas electromagnéticas, son de especial interés debido a que son captadas por los ojos y procesadas por el cerebro. El ojo humano es capaz de distinguir entre 400 y 700 nanómetros (nm). Nuestro sistema sensorial visual interpreta las diferentes amplitudes y longitudes de onda de la luz, produciendo las sensaciones que conocemos como brillo y color respectivamente.

Como podemos observar en la Figura 16, la parte de la radiación electromagnética que constituyen las ondas luminosas abarca desde el fin del ultravioleta (400 nm) hasta el comienzo del infrarrojo (700 nm).

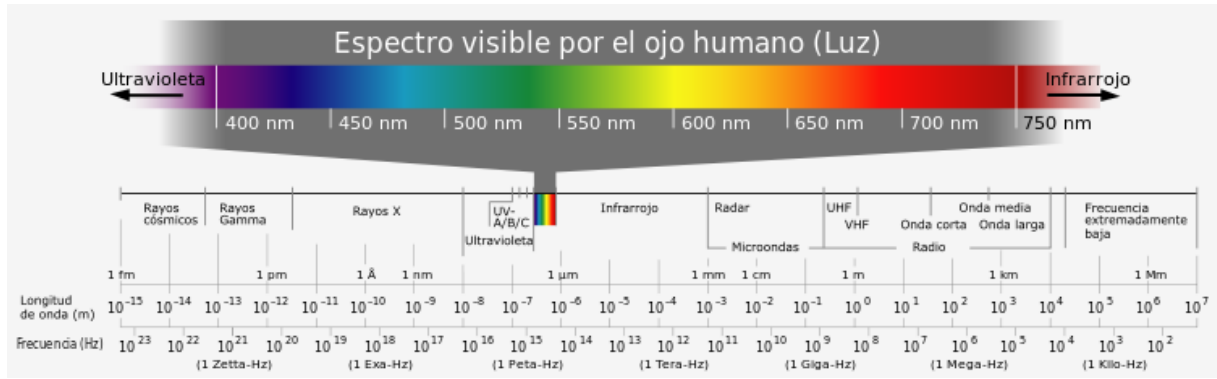


Figura 16 [13] Espectro visible.

Distribución espectral de energía

Una curva de distribución espectral de energía representa la cantidad de energía (en watts) asociada a cada longitud de onda en una radiación electromagnética. Si se representa el diagrama espectral de una radiación electromagnética que posee una longitud de onda determinada, y se obtiene un gráfico con un pico en la longitud correspondiente y 0 en el resto se dice que es una luz *monocromática*.

En general las radiaciones no son tan puras y resultan de la mezcla de diferentes haces con diferentes longitudes de onda. Además entre más monocromático sea un haz de luz, tendrá menos energía asociada, por lo que será más difícil percibirlo, es por ello que los diagramas espectrales que encontramos en la naturaleza sean más parecidos a los que se presentan por ejemplo en las curvas de emisión y excitación de las proteínas fluorescentes que se presentan en la Figura 17.

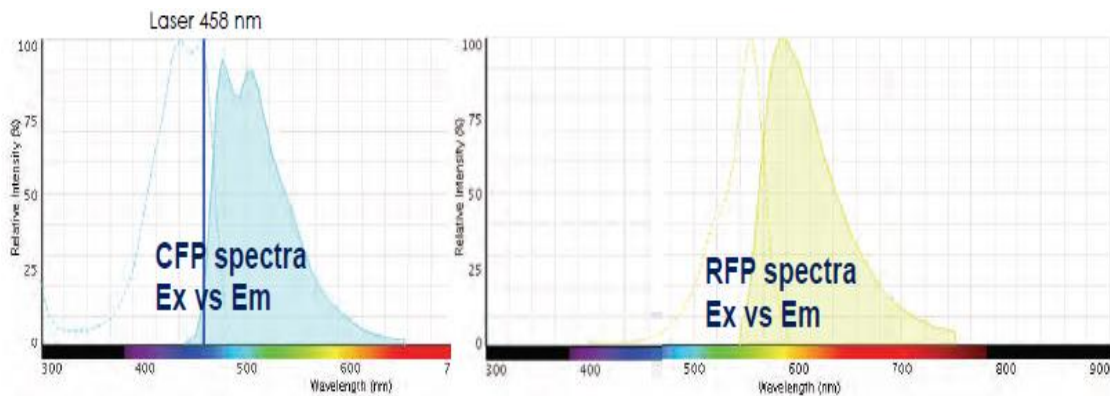


Figura 17 [14] Espectros visibles de proteínas fluorescentes.

Flujo luminoso

El *flujo luminoso* es la parte del flujo radiante detectada por el ojo. La unidad de flujo luminoso es el lumen (L). Un lumen corresponde al flujo luminoso procedente de una abertura de $1/60 \text{ cm}^2$ en un

cilindro de material refractario que contiene un material patrón que radia a través de un cono de radiación de un estereorradián. El flujo luminoso se puede medir con un fotómetro y se representa con el símbolo Φ

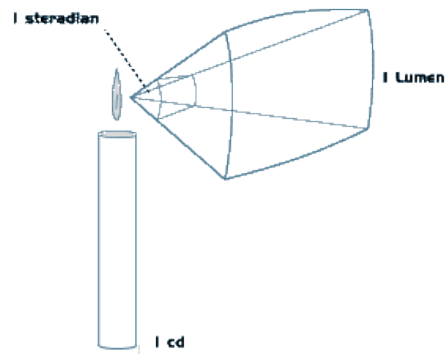


Figura 18 [15] Flujo luminoso.

Intensidad luminosa

La *intensidad luminosa* es el flujo luminoso emitido por unidad de ángulo sólido. Se representa como I , y su unidad es la *bujía* (b), que se corresponde a un lumen/estereorradián. Puede comprobarse que la intensidad luminosa es independiente de la distancia a la que se encuentra la fuente, y sólo varía según la orientación de la medición.

$$I = \frac{d\Phi}{d\omega} \text{ (b)}$$

Luminancia o brillo

La luminancia o brillo de una fuente de luz es la intensidad luminosa por unidad de superficie. Así, por ejemplo si una noche dividimos secciones de área iguales y medimos el brillo de una estrella y el de un foco, encontraremos que el foco tiene un brillo mayor. Sin embargo, la intensidad luminosa de cualquier objeto de nuestro entorno es menor que la de la estrella, pues la intensidad luminosa no depende de la distancia.

Saturación

La saturación mide la proporción entre la longitud de onda dominante y el resto de longitudes de onda.

Definidos los conceptos de matiz, saturación y brillo se dice que se ve un color determinado cuando se percibe una cierta combinación de estos tres elementos.

Histograma

El histograma de una imagen F es una gráfica que representa los niveles de intensidad del color de F con respecto al número de píxeles presentes en f con cada intensidad de color.

IV.2.b. Etapas de un sistema de visión artificial

El ser humano captura la luz a través de los ojos, y esta información circula a través del nervio óptico hasta el cerebro donde es procesada, y existen razones para creer que el primer paso de ese procesamiento es descomponer la imagen en elementos más simples como segmentos y arcos. La visión artificial en un intento por reproducir este comportamiento, tiene normalmente cuatro fases principales:

- *Captura o adquisición:* Se adquieren las imágenes mediante algún tipo de sensor.
- *Tratamiento digital de las imágenes:* Esta fase es de procesamiento previo y tiene por objeto facilitar las etapas posteriores, consiste en eliminar partes indeseables o realzar partes interesantes de la imagen mediante filtros y transformaciones geométricas.
- *Segmentación:* Consiste en aislar los elementos que interesan de una escena para comprenderla.
- *Reconocimiento o clasificación:* En esta etapa se pretende distinguir los objetos segmentados, gracias al análisis de ciertas características que se establecen previamente para diferenciarlos.

Cabe mencionar que estas cuatro etapas no se siguen de manera secuencial, pues deben retroalimentarse para asegurar que el sistema funcione adecuadamente. Así, es normal volver a la etapa de segmentación si falla la etapa de reconocimiento o la de preproceso, o incluso a veces es necesario volver a la etapa de captura cuando falla alguna de las anteriores.

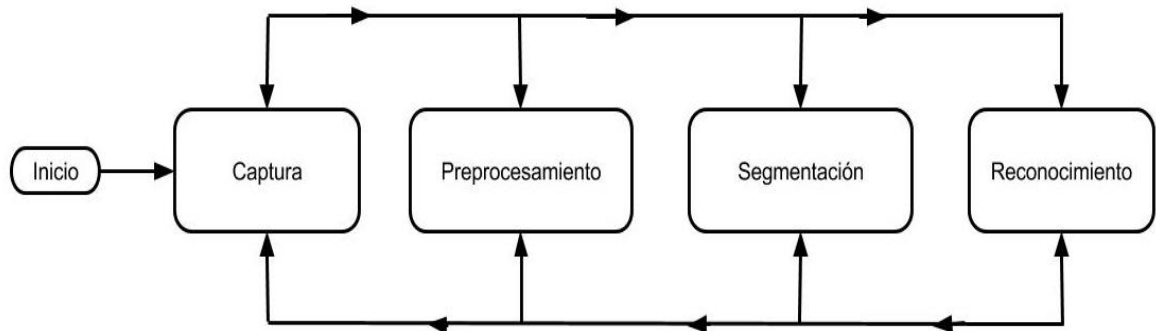


Diagrama 1 Etapas típicas en un sistema de visión computacional.

IV.2.c. Adquisición y representación de imágenes digitales

En esta sección se tratan algunos aspectos de la adquisición de la imagen del mundo físico y su paso al dominio discreto y virtual informático.

Una vez digitalizada una imagen bidimensional digital está constituida por un conjunto de elementos llamados píxeles. Cada píxel ofrece cierta información sobre una región fundamental de la imagen. En imágenes en niveles de gris esta información es el brillo.

Captura y digitalización

Las imágenes digitales son “señales” discretas que suelen tener origen en una “señal” continua. Por ejemplo una cámara digital toma imágenes del mundo real que es continuo tanto en tiempo como en espacio. En el proceso de obtención de imágenes digitales se distinguen dos etapas, la primera conocida como *captura*, utiliza un dispositivo, generalmente óptico con el que se obtiene información relativa a una escena.

En la segunda etapa, que se conoce como *digitalización*, se transforma esa información en una imagen digital, que es una señal con todas sus componentes discretas, dicho de otra manera es el proceso de paso del mundo continuo (o analógico) al mundo discreto (o digital). En la digitalización normalmente se distinguen dos procesos: el muestreo y la cuantificación.

Muestreo

El muestreo de una señal continua consiste en su medición a intervalos respecto de alguna variable (generalmente el tiempo o el espacio), siendo su parámetro fundamental la frecuencia de muestreo, que representa el número de veces que se mide un valor analógico por unidad de cambio.

Mediante el muestreo se convierte una imagen I_C , que es algo continuo, en una matriz discreta I_D de $N \times M$ píxeles. El número de muestras por unidad de espacio sobre el objeto original conduce al concepto de *resolución espacial* de la imagen. Ésta se define como la distancia, sobre el objeto original, ente dos píxeles adyacentes, o dicho de otra forma, la distancia real que representa un píxel.

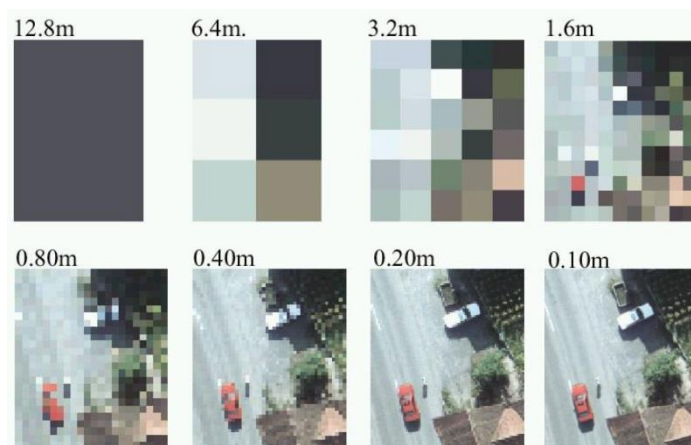


Figura 19 [19] En la imagen superior observamos el cambio de la resolución espacial sobre una misma escena.

De esta forma. El proceso de muestreo, para una imagen, que asocia a cada punto un valor real, cambia una imagen del formato:

$$I_C(x, y) \in \mathfrak{R} \text{ en donde } x, y \in \mathfrak{R}$$

Al formato:

$$I_D(x, y) \in \mathfrak{R} \text{ en donde } x, y \in N \text{ y } 0 \leq x \leq N - 1, 0 \leq y \leq M - 1$$

Cuantificación

La segunda operación es la cuantificación de la señal, que consiste en la asignación de valores a cada pixel. Los niveles de cuantificación suelen ser potencias de 2 para facilitar el almacenamiento de las imágenes en la computadora. De esta forma, $I_D(x, y) \in \mathfrak{R}$ se convierte en $I_{DC}(x, y) \in N$ (imagen discreta cuantificada). El número de niveles posibles define la *resolución radiométrica*.

Cuando las imágenes sólo tienen información sobre el brillo se habla de *imágenes en niveles de gris* y se suelen utilizar hasta 256 niveles para representar los tonos intermedios desde el negro (0) hasta el blanco (255). Si sólo se permiten dos niveles de cuantificación (normalmente blanco y negro) se habla de *imágenes binarias* o bitonales. Para el caso del color suelen usarse 256 niveles para representar la intensidad de cada uno de los tres colores primarios (RGB). De esta forma se obtienen 16 millones de colores aproximadamente y se habla de *imágenes en color real*.

IV.2.d. Relaciones básicas entre pixeles

En este apartado se definen ciertas relaciones que se establecen entre los pixeles de una imagen.

Relaciones de proximidad

Vecindad

Para todo punto p de coordenadas (x, y) se dice que un pixel q pertenece a sus 4-vecinos y se escribe $q \in N_4(p)$ si y sólo si q tiene coordenadas:

$$(x-1, y) \text{ ó } (x, y-1) \text{ ó } (x+1, y) \text{ ó } (x, y+1)$$

Para todo punto p de coordenadas (x, y) se dice que un pixel q pertenece a sus 8-vecinos y se escribe $q \in N_8(p)$ si y sólo si q tiene coordenadas:

$$(x-1, y) \text{ ó } (x, y-1) \text{ ó } (x+1, y) \text{ ó } (x, y+1) \text{ ó } (x-1, y-1) \text{ ó } (x-1, y+1) \text{ ó } (x+1, y-1) \text{ ó } (x+1, y+1)$$

Este concepto lo podemos visualizar a continuación en la Figura 20.



	B	A	B	
	A	p	A	
	B	A	B	

Figura 20 Los 4-vecinos de p son los puntos A. Los 8-vecinos son los puntos A y B.

Conectividad

Podemos ver una imagen como una matriz cuyos elementos representan pixeles. Entre los pixeles de esta matriz se puede definir una relación que define dos pixeles como conectados cuando son vecinos y sus valores son similares desde algún punto de vista. Formalmente se define un conjunto V que representa los valores compatibles para que dos pixeles que sean vecinos se diga que están conectados:

$$V = \{\text{Valores de los pixeles que definen conectividad}\}$$

Se dice que dos pixeles p y q con valores en V están 4-conectados si q pertenece a $N_4(p)$.

Se dice que dos pixeles p y q con valores en V están 8-conectados si q pertenece a $N_8(p)$.

Camino

Un camino desde el pixel p , de coordenadas (x,y) , al pixel q , de coordenadas (s,t) , es una secuencia de pixeles distintos de coordenadas

$$(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$$

Donde $(x_0, y_0) = (x, y)$ y $(x_n, y_n) = (s, t)$ y (x_i, y_i) está conectado a (x_{i-1}, y_{i-1}) , siendo n la longitud del camino. Se puede hablar de 4,8 y m -caminos depende del tipo de conexión involucrada.

Componente conexas

Para todo pixel p de una imagen, el conjunto de los pixeles hasta los que hay un camino desde p se dice que forman su *componente conexas*. Además se cumple que dos componentes conexas distintas tienen conjuntos de pixeles disjuntos.

Relaciones de distancia

La relación de distancia más utilizada es la *distancia geométrica o distancia euclídea*. Se define la distancia euclídea entre el pixel p de coordenadas (x,y) y el pixel q de coordenadas (s,t) como la raíz cuadrada de la diferencia de coordenadas al cuadrado. Es decir:

$$d(p, q) = \sqrt{(x - s)^2 + (y - t)^2}$$

Siempre que sólo sea importante desde un punto de vista comparativo, se puede prescindir del cálculo de la raíz cuadrada lo que resultará en una mayor velocidad de cálculo.

Otra relación de distancia usual es la *distancia Manhattan* o *distancia del taxista*, que se define entre los mismos puntos p y q como:

$$d(p, q) = |x - s| + |y - t|$$

También puede citarse la *distancia del tablero de ajedrez* o *distancia chessboard* que se define como:

$$d(p, q) = \max(|x - s|, |y - t|)$$

Nótese que con la distancia Manhattan sólo los vecinos 4 conexos de un pixel están a distancia unidad, mientras que con la distancia tablero todos los vecinos 8 conexos están a distancia unidad.

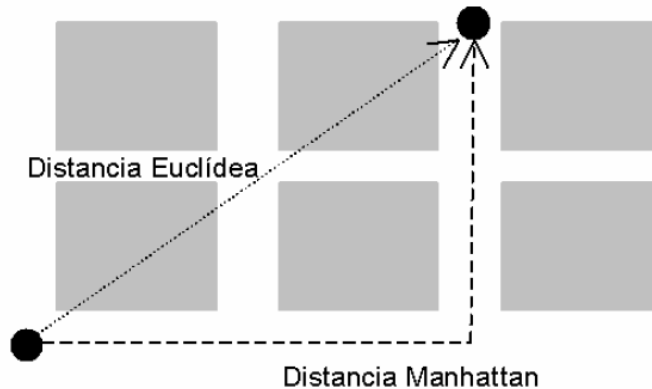


Figura 21 Representación gráfica de la distancia entre dos puntos.

IV.2.e. Procesamiento digital de la imagen

El procesamiento digital de la imagen consiste en eliminar la mayor cantidad de ruido que se le agrega durante la adquisición así como también mejorar las características de dicha imagen como: definición de contornos, color, brillo, etc., valiéndose de procedimientos y herramientas matemáticas. [15]

Filtrado y realzado de la imagen

A lo largo de este apartado se tratan las operaciones y transformaciones que se aplican sobre las imágenes digitales en una etapa de procesamiento previa a la segmentación y reconocimiento, esto con el fin de mejorar algún elemento en la imagen que ayude al procesamiento de segmentación y clasificación.

Las operaciones que se describirán pueden explicarse desde la perspectiva ofrecida por la *teoría de filtros*. Un *filtro* puede verse como un mecanismo de cambio o transformación de una señal de entrada a la que se le aplica una función de transferencia, para obtener una señal de salida. En este contexto se entiende por señal una función de una o varias variables independientes.

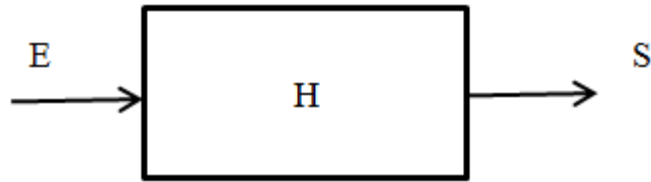


Diagrama 2 Esquema de funcionamiento de un filtro, siendo E la función de entrada, S la salida y H la función de transferencia del filtro.

Las señales que maneja un filtro pueden ser discretas o continuas. Aunque en el tratamiento de imágenes digitales se procesan señales y funciones discretas, suele recurrirse al caso continuo para explicar sus comportamientos.

Operaciones básicas entre píxeles

Las operaciones directas sobre los píxeles se pueden clasificar en operaciones aritmético-lógicas y operaciones geométricas.

Operaciones aritmético-lógicas

Estas operaciones son las más usadas a cualquier nivel en sistemas de tratamiento de imágenes, ya que son las que se utilizan para leer y dar valores a los píxeles de las imágenes. Las operaciones básicas son:

- *Conjunción*: Operación lógica AND entre los bits de dos imágenes. Se utiliza para borrar píxeles en una imagen.
- *Disyunción*: Operación lógica OR entre los bits de dos imágenes. Se usa para añadir píxeles a una imagen.
- *Negación*: Inversión de los bits que forman una imagen. Se usa para obtener el negativo de una imagen.
- *Suma*: Suma de los valores de los píxeles de dos imágenes.
- *Resta*: Resta de los valores de los píxeles de dos imágenes.
- *Multiplicación*: Multiplicación de los valores de los píxeles de una imagen por los de otra. Se usa para añadir textura a una imagen.
- *División*: División de los valores de los píxeles de una imagen entre los de otra.

Normalmente en las imágenes en niveles de gris se usa el valor 255 para representar el blanco y el 0 para el negro. Así, la operación de conjunción entre negro y blanco da como resultado negro. Si para el negro se utilizase el 255 y para el blanco el 0 los valores estarían intercambiados.

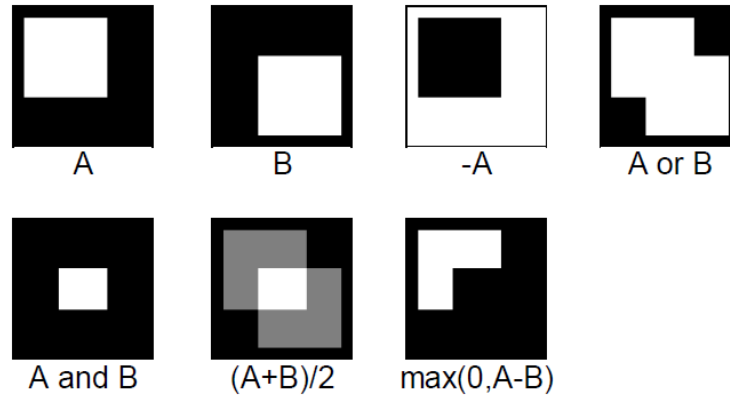


Figura 22 [20] Operaciones entre las imágenes de nivel de gris A y B, donde el negro corresponde a 0 y el blanco a 255.

Cuando se realizan operaciones aritméticas se debe tener cuidado de que el resultado caiga dentro del dominio de valores permitidos, para lo cual también es buena idea utilizar múltiplos del tamaño de palabra del procesador a fin de minimizar el número de operaciones de acceso a la memoria. También se debe implementar los algoritmos de forma tal que en accesos consecutivos se referencien posiciones de memoria cercanas, de manera que el criterio de proximidad referencial, usado por las memorias caché, permita un rendimiento óptimo.

Operaciones geométricas

Si se expresan los puntos en coordenadas homogéneas, todas las transformaciones se pueden tratar mediante multiplicación de matrices. Las operaciones geométricas más usuales son:

- *Traslación.*- Movimiento de los píxeles de una imagen según un vector de movimiento. La siguiente transformación muestra el resultado de trasladar el punto (x,y) según el vector (d_x, d_y) , obteniendo el punto (x',y') .

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- *Escalado.*- Cambio del tamaño de una imagen. La siguiente transformación muestra el resultado de escalar el punto (x,y) en un factor (s_x, s_y) , obteniendo el punto (x',y') .

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- *Rotación.*- Giro de los pixeles de una imagen en torno al origen de coordenadas. La siguiente transformación muestra el resultado de rotar el punto (x,y) un ángulo θ , obteniendo el punto (x',y') .

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \text{Cos}(\theta) & -\text{Sen}(\theta) & 0 \\ \text{Sen}(\theta) & \text{Cos}(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Las operaciones geométricas matriciales se pueden agrupar multiplicando las matrices. De esta forma, por ejemplo, es posible tener una única matriz que realice un desplazamiento, y un reescalado en un solo paso. Al realizar esta composición de operaciones se debe recordar que el producto de matrices no cumple la propiedad conmutativa.

Morfología matemática

Consiste en un conjunto de técnicas que permiten tratar problemas que involucran formas en una imagen. La morfología matemática tiene su origen en la teoría de conjuntos. Para ella las imágenes binarias son conjuntos de puntos 2D, que representan los puntos activos de una imagen, y las imágenes en niveles de gris son conjuntos de puntos 3D, donde la tercera componente corresponde al nivel de intensidad.

Definiciones:

Traslación de A por $X=(x_1, x_2)$, como:

$$(A)_x = \{c \mid c = a + x, \forall a \in A\}$$

Reflexión de A como:

$$\hat{A} = \{x \mid x = -a, \forall a \in A\}$$

Complementario de A como:

$$A_c = \{x \mid x \notin A\}$$

Diferencia entre dos conjuntos A y B como:

$$A - B = \{x \mid x \in A \notin B\}$$

De las operaciones anteriores se deriva:

$$A - B = A \cap B_c$$

Dilatación:

Siendo A y B dos conjuntos en Z^2 , la *dilatación* de A con B , denotada como $A \oplus B$, se define como:

$$A \oplus B = \{x \mid x = a + b \ \forall a \in A \ y \ \forall b \in B\}$$

El elemento B es el elemento que dilata al elemento A , y se conoce como *elemento estructurante* de la dilatación.

La dilatación cumple la propiedad conmutativa.

$$A \oplus B = B \oplus A$$

La implementación directa de la dilatación según la definición dada es demasiado costosa. La siguiente formulación, que puede demostrarse que es equivalente, da pistas para una interpretación mucho más eficiente.

$$A \oplus B = \{x \mid \hat{B}_x \cap A \neq \emptyset\}$$

O escrito de otra forma:

$$A \oplus B = \{x \mid [\hat{B}_x \cap A] \subseteq A\}$$

Lo cual produce el efecto de dilatar el aspecto del elemento A usando para ello a B .

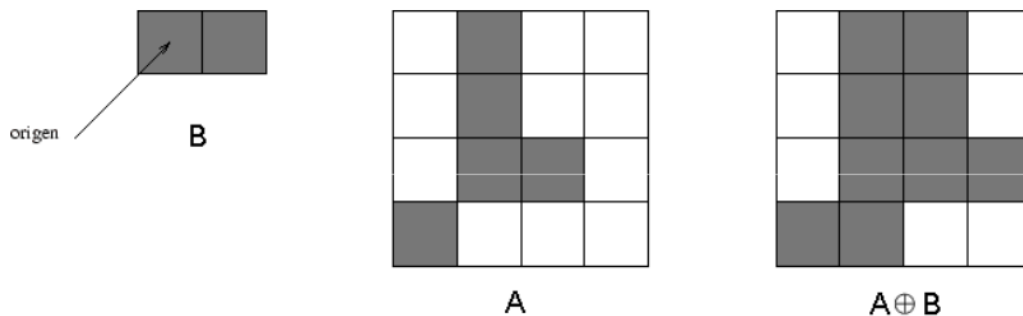


Figura 23 [20] Efecto de dilatar A con el elemento estructurante B.

Erosión:

Siendo A y B dos conjuntos en Z^2 , la *erosión* de A con B , denotada como $A \ominus B$, se define:

$$A \ominus B = \{x \mid x + b \in A \ \forall b \in B\}$$

Lo cual puede reducirse de forma que el coste computacional se reduzca:

$$A \ominus B = \{x \mid B_x \subseteq A\}$$

La erosión adelgaza la imagen sobre la que se aplica siendo, en un sentido no estricto, opuesta a la dilatación.

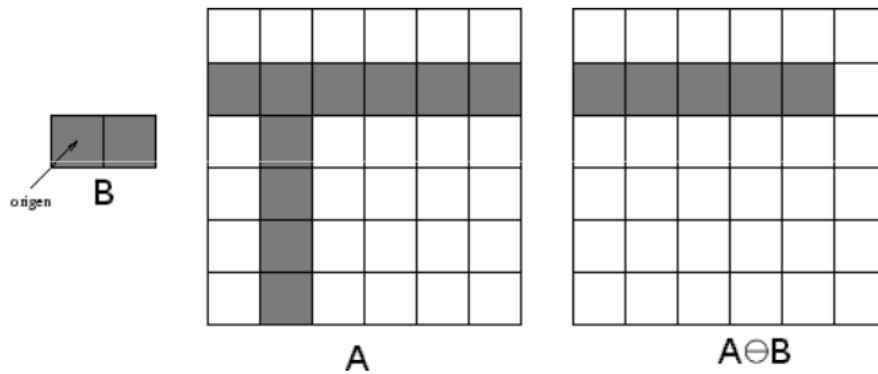


Figura 24 [20] Efecto de erosionar A con el elemento estructurante B .

Apertura:

La *apertura* de A con B se define como;

$$A \circ B = (A \ominus B) \oplus B$$

Sus propiedades son:

$A \circ B$ es un subconjunto de A

$$(A \circ B) \circ B = A \circ B$$

Si C es subconjunto de $D \Rightarrow C \circ B$ es subconjunto de $D \circ B$.

Intuitivamente la apertura de A con un elemento estructurante B equivale a determinar los puntos en los que puede situarse alguna parte de B cuando se desplaza por el interior de A . La apertura abre, o agranda, las zonas de pixeles inactivos presentes en una zona de pixeles activos.

Cierre:

El cierre de A con B se define como:

$$A \bullet B = (A \oplus B) \ominus B$$

Sus propiedades son:

A es un subconjunto de $A \bullet B$

$$(A \bullet B) \bullet B = A \bullet B$$

Si C es subconjunto de $D \Rightarrow C \bullet B$ es subconjunto de $D \bullet B$

Intuitivamente el cierre de A con un elemento estructurante B equivale a los puntos a los que no puede acceder ninguna parte de B cuando se desplaza por el exterior de A . El cierre elimina zonas de pixeles inactivos en el interior de una zona de pixeles activos.

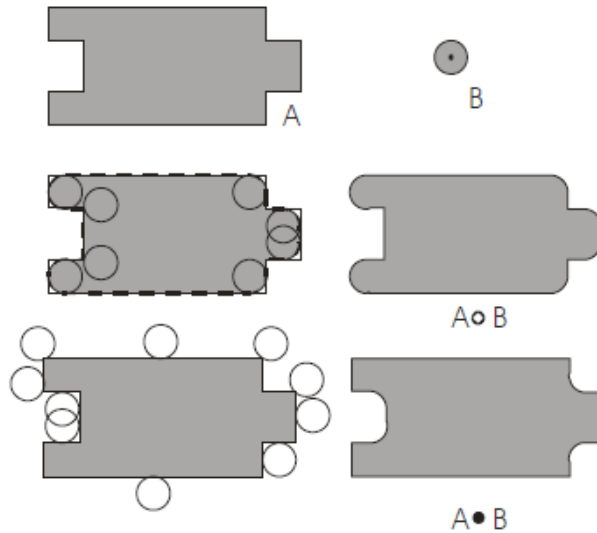


Figura 25 [20] Observamos en la imagen superior la estructura A y el elemento estructurante B , en medio se presenta la operación de apertura y abajo la operación de cierre.

Coincidencia estructural:

Si el elemento B se define teniendo en cuenta los puntos a blanco que lo rodean, se tiene la descripción de un objeto B_1 y su entorno B_2 . La operación de *coincidencia estructural*, o *Hit or Miss*, busca la parte de la imagen A y los puntos del entorno de B estén en A_c .

$$A \circledast B = (A \ominus B_1) \cap (A_c \ominus B_2)$$

El elemento estructurante B suele definirse sobre una cuadrícula donde: un cuadro sombreado indica que el pixel pertenece a B_1 , un cuadro blanco indica que el pixel pertenece a B_2 , un cuadro con una cruz indica que ese cuadro no debe tomarse en cuenta.

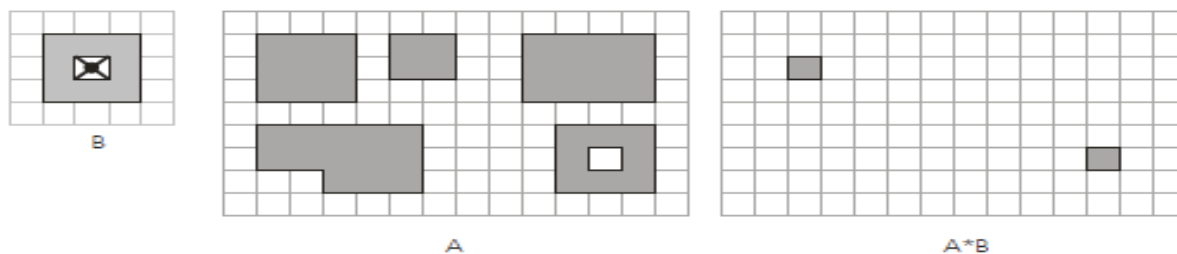


Figura 26 [20] Ejemplo de operación de coincidencia estructural.

Segmentación

La segmentación es una técnica de clasificación de píxeles que permite la formación de las regiones similares en la imagen. Generalmente estas regiones corresponden a objetos de la imagen. La segmentación de una imagen consiste en la división o partición de la imagen en varias zonas o regiones homogéneas y disjuntas a partir de su contorno, su conectividad, o en términos de un conjunto de características de los píxeles de la imagen que permitan discriminar unas regiones de otras. Los tonos de gris, la textura, los momentos, la magnitud del gradiente, la dirección de los bordes, las modas de los tonos de gris en ventanas 3x3, 7x7 y 15x15, etc., son características a utilizar para la segmentación. [15]

Los algoritmos de segmentación de imágenes monocromáticas se basan en alguna de las tres propiedades siguientes:

- a. *Discontinuidad* en los tonos de gris de los píxeles de un entorno, que permite detectar puntos aislados, líneas y aristas (bordes).
- b. *Similaridad* en los tonos de gris de los píxeles de un entorno, que permite construir regiones por división y fusión, por crecimiento o por umbralización.
- c. *Conectividad* de los píxeles.

Conectividad, Adyacencia y Fronteras

La *conectividad* entre píxeles es un concepto fundamental que simplifica la definición de numerosos conceptos en PDI, como por ejemplo regiones y bordes o fronteras entre objetos. Para establecer si dos píxeles están conectados, se debe determinar si son vecinos y si sus niveles de gris satisfacen alguna condición especificada como un criterio de similaridad (por ejemplo si son iguales). [21]

La adyacencia se define de la siguiente forma: Dos píxeles son adyacentes si, y sólo si, tienen en común una de sus fronteras o al menos una de sus esquinas. [22]

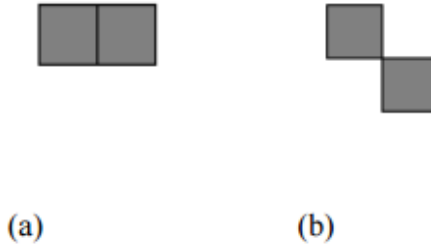


Figura 27 [20] Píxeles adyacentes (a) por frontera (b) por esquina.

Dos píxeles son vecinos si cumplen con la definición de adyacencia. Si comparten una de sus fronteras, se dice que son vecinos directos, y si sólo se tocan en alguna de sus esquinas se llaman vecinos indirectos.

Segmentación basada en umbralización

El método de segmentación basada en umbralización hace uso de histogramas estadísticos que definen uno o múltiples umbrales para clasificar una imagen pixel por pixel.

Un enfoque simple consiste en examinar el histograma en busca de distribución bimodal (blanco y negro). Si el histograma resulta ser bimodal, el umbral puede ser establecido a un valor de gris correspondiente al punto más profundo del valle del histograma. De lo contrario, la imagen puede ser dividida en dos o más regiones haciendo uso de las reglas, con base en las propiedades de la imagen. El histograma de cada partición o división puede ser utilizado para determinar los umbrales de la imagen.

Supongamos que una imagen, o parte de una imagen tiene un histograma de valores grises bimodal, la imagen en términos de x y y dando $f(x,y)$, puede ser segmentada en dos clases empleando un umbral T de valores en gris como se muestra a continuación.

$$g(x, y) = \begin{cases} 1 & \text{Si } f(x,y) > T \\ 0 & \text{Si } f(x,y) < T \end{cases}$$

Una propuesta para determinar los valores en gris del umbral T es mediante el análisis del histograma en el valor pico y después encontrar el punto de valle más profundo entre los dos picos principales que se encuentren de manera consecutiva.

Segmentación basada en movimiento

El movimiento puede constituir una poderosa herramienta para la segmentación de objetos en movimiento sobre fondos estáticos. Las técnicas básicas consisten en el estudio de la imagen resultante de la resta de dos imágenes consecutivas de una secuencia animada. Esta técnica se conoce como *substracción de fondo*. Los objetos que se desplazan entre estas dos imágenes producen en la imagen de resta un conjunto de píxeles con valores distintos a cero. Mientras los elementos estáticos de la imagen, producen cero tras la resta.

Así, partiendo de dos imágenes I_t e I_{t+1} de dos instantes consecutivos los objetos tras esta segmentación serían los píxeles a uno en la imagen I_d .

$$d_{t,t+1}(x, y) = \begin{cases} 1 & \text{si } |I_{t+1}(x, y) - I_t(x, y)| > U \\ 0 & \text{en otro caso} \end{cases}$$

Siendo U un valor umbral que depende de la variación de la iluminación entre los instantes t y $t+1$.

IV.2.f. Principios de evaluación de desempeño

El desempeño de una técnica de segmentación puede ser evaluado visualmente y cuantificado en base a los requerimientos de su tarea, y en este caso debemos hacer una evaluación de las diferentes propuestas para determinar cuál es la más conveniente de acuerdo a las necesidades y limitaciones del sistema.

Una evaluación cuantitativa representa todo un reto en la medida de que es difícil establecer una base verdadera válida, lo que significa el proceso de definir la “respuesta correcta” para lo que “exactamente” se espera que el algoritmo produzca. A continuación explicaremos en qué consisten los métodos elegidos para la evaluación de los criterios propuestos en cada proceso de un sistema de visión:

Clasificación de porcentaje correcto (PCC)

Es usada como una medida estadística para determinar qué tan bien un algoritmo identifica o excluye los objetos de primer plano en la segmentación. [23]

Se expresa matemáticamente como:

$$PCC = \frac{TP + TN}{TP + FP + TN + FN}$$

Donde:

$TP \rightarrow$ Verdadero positivo (*True positive*)

$TN \rightarrow$ Verdadero negativo (*True negative*)

$FP \rightarrow$ Falso positivo (*False positive*)

$FN \rightarrow$ Falso negativo (*False negative*)

Entonces una precisión del 100% significa que los valores medidos son exactamente iguales a los de la base verdadera.

Cálculo del error en distancia

En este método se propone obtener el centro real de la célula haciendo un adelgazamiento de contornos de los elementos del sistema a través de las operaciones morfológicas siguientes:

$$A \otimes B = A - (A \circledast B) = A \cap (A \circledast B)^c$$

Una vez hecho el adelgazamiento se procede a determinar el centro geométrico al cual llamaremos centro real y se hace una relación de distancia euclídea entre éste y el centro calculado por el algoritmo a evaluar.

Coefficiente de Dice

Se trata de un coeficiente de similitud entre dos muestras y se define matemáticamente de la siguiente forma:

$$s = \frac{2|A \cap B|}{|A| + |B|}$$

Donde A y B son el número de especies en las muestras y $|A \cap B|$ es el número de especies compartidas por ambas, en el caso de nuestra aplicación cuando mencionamos *especies* hablamos del número de pixeles. El coeficiente de similitud se representará con la letra s y su valor varía de 0 a 1.

IV.3 Diseño y análisis de las propuestas de solución

IV.3.a. Elección de resolución espacial y radiométrica

1. *Resolución espacial* de la imagen: es importante justificar este parámetro debido a que de él depende el diseño del sistema óptico que irá a bordo de la misión.

Existe la necesidad de detectar y monitorear a las células hijas, por lo que se requiere que la mínima resolución espacial contemple el tamaño de las mismas. El algoritmo trabajará con la resolución que nos dan las imágenes de un microscopio invertido con un objetivo de 20X . Partiendo de que con éste obtenemos aproximadamente en cada pixel la representación de 0.17 um, se establece:

Requerimiento planteado: 1 px equivale a 0.17 um x 0.17 um

Justificación

Observamos en las imágenes tomadas por el microscopio electrónico, que cuando una célula comienza a reproducirse el diámetro de la célula hija es de aproximadamente 10 px, lo que representa un mínimo para ser detectada como un cambio en la estructura de la célula madre, por lo que se determina que lo ideal para que se detecten las células hijas es que se respete dicha resolución.

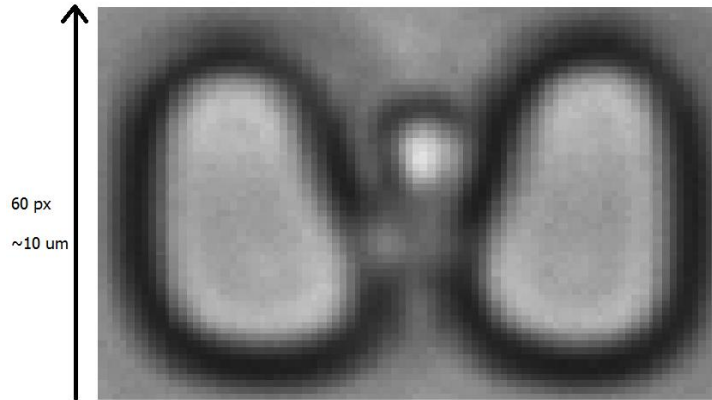


Imagen 1 Imagen en claro oscuro de una célula atrapada en una trampa.

En la Imagen 1 observamos la relación que existe entre el número de pixeles y el tamaño que representan, por lo que de esta imagen por regla de tres hacemos el cálculo de la longitud que tiene un pixel por lado suponiendo que éstos son cuadrados:

$$60 \text{ px} : 10 \text{ um} :: 1 \text{ px} : x$$

$$x = \frac{1 \text{ px} * 10 \text{ um}}{60 \text{ px}}$$

$$x = 0.167 \text{ um}$$

$$\therefore 1 \text{ px} = 0.17 \text{ um}$$

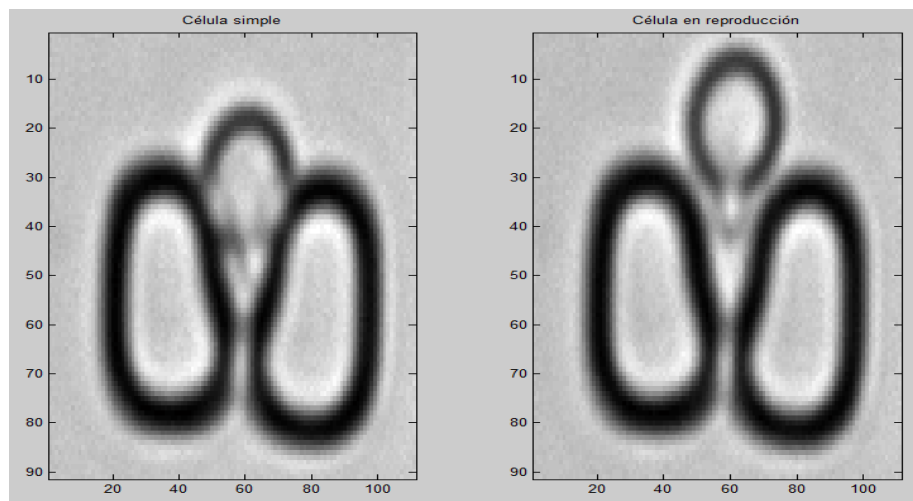


Imagen 2 Muestra de una trampa en un delta de tiempo donde comienza a reproducirse y vemos la forma y tamaño típico de una célula hija.

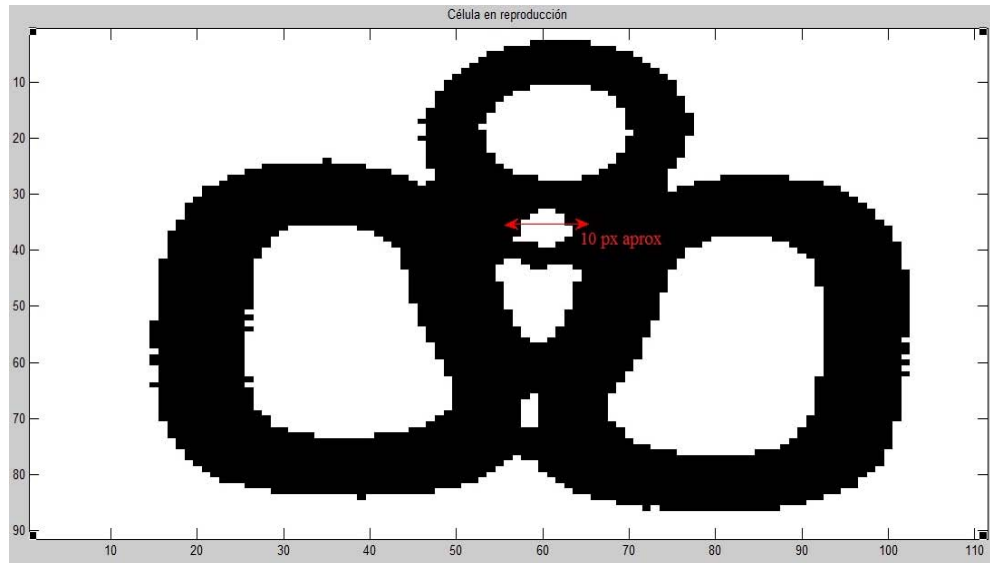


Imagen 3 Medición de una célula hija en una imagen binaria.

1. *Resolución radiométrica de la imagen:* Con respecto a la resolución radiométrica, se contempla que con 256 niveles de cuantificación es suficiente para hacer el procesamiento de la imagen pues con esta resolución aseguramos que la umbralización se hace en un rango de valores tales que puede cumplir el objetivo de segmentación.

IV.3.b. Planteamiento de soluciones

Como se ha expuesto anteriormente todos los sistemas de la misión MICROLAB están en desarrollo, por lo que en el diseño del algoritmo se consideró que la disposición final de las trampas en el dispositivo microfluídico aún puede variar y por ello se propone en esta tesis hacer que el algoritmo trabaje por módulos que procesen las trampas individualmente como se muestra en el Diagrama 3, lo que significa que el algoritmo trabajaría sobre una sección o recorte de las imágenes originales y después esto puede ser controlado con un programa cuyo fin sea ejecutar estos procesos en las diferentes trampas de acuerdo a cómo el análisis lo requiera.

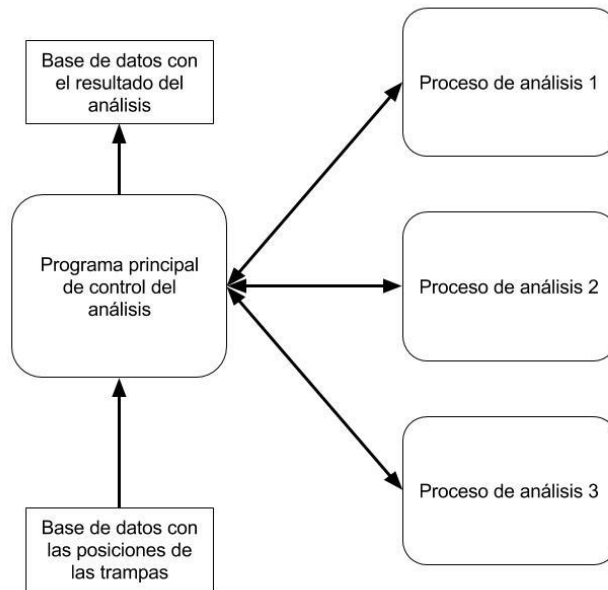


Diagrama 3 Se muestra la interacción entre los algoritmos, donde los procesos que se muestran corresponden a las etapas en las que dividimos el procesamiento de la imagen.

Para el diseño de estas propuestas se asume lo siguiente:

- Imágenes de 5 Mpx en escala de grises tomadas con la técnica de campo claro.
- Resolución radiométrica de la imagen: 1 byte por pixel.
- Resolución espacial de la imagen: 1 px equivale a 0.17 μm x 0.17 μm
- Cámara fija: la estabilidad de ésta es la que permite detectar movimiento en la escena.
- Luz estable: sin parpadeos ni intermitencias en el momento de la toma, además de una misma intensidad luminosa en todas las imágenes.
- Fondo contrastante: se requiere que el fondo contraste con los objetos de interés (contornos de las diferentes estructuras), donde el caso ideal es que se ocupe todo el rango radiométrico, representando con 255 el blanco y con 0 el color negro.
- Tasa de muestreo: de 1 a 3 imágenes consecutivas capturadas cada 10 minutos.

Observamos también que uno de los objetivos del sistema de visión es la detección de movimiento, para lo cual tenemos que tomar en cuenta los problemas presentes en nuestro reconocimiento, entre los que encontramos:

- *Objetos movidos*: Cuando un objeto de fondo se mueve, no debe considerarse como el movimiento del objeto de primer plano. Esto se observa en las trampas, que aunque se suponen fijas, al trabajar con dimensiones tan pequeñas (orden de micrómetros) llega a haber vibraciones que de presentarse pueden generar un error grande en el análisis.

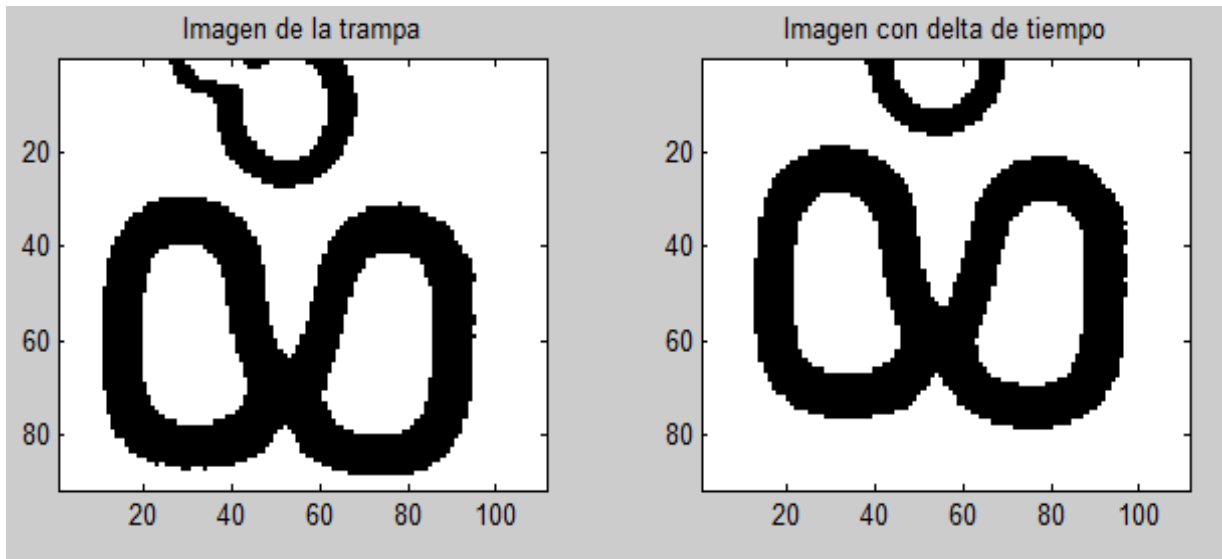


Imagen 4 Se presentan objetos movidos sobre el eje vertical.

En la Imagen 4 observamos el problema de objetos movidos, pues se pierden características de la célula al recortar la misma sección de la imagen, y aunque la trampa (fondo) se supone fija podría confundirse como un movimiento del objeto de interés.

- *Camuflaje:* Algunas características del objeto de primer plano se confunden con el fondo. Esto lo observamos una vez que la célula fue atrapada, pues los límites de la trampa se confunden con las paredes de la célula.

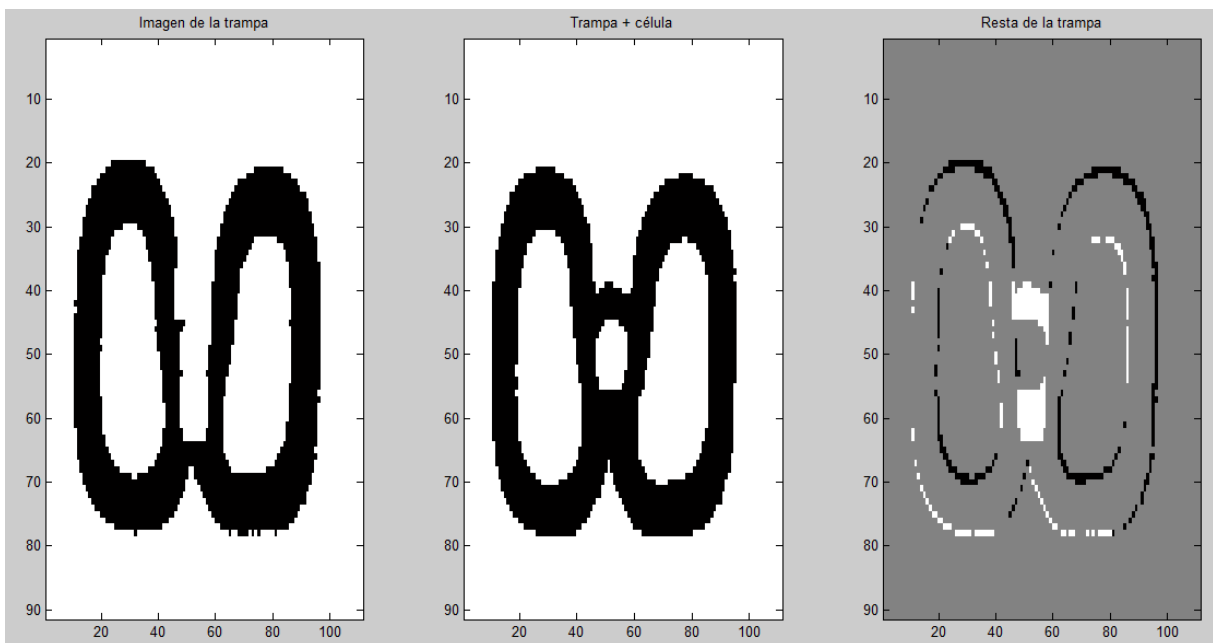


Imagen 5 En la tercera imagen de izquierda a derecha observamos que las paredes laterales de la célula se han confundido con los límites de la trampa.

- *Sleeping person*: Se debe hacer una distinción entre el objeto de interés cuando no se mueve con uno del fondo que presente movimiento.

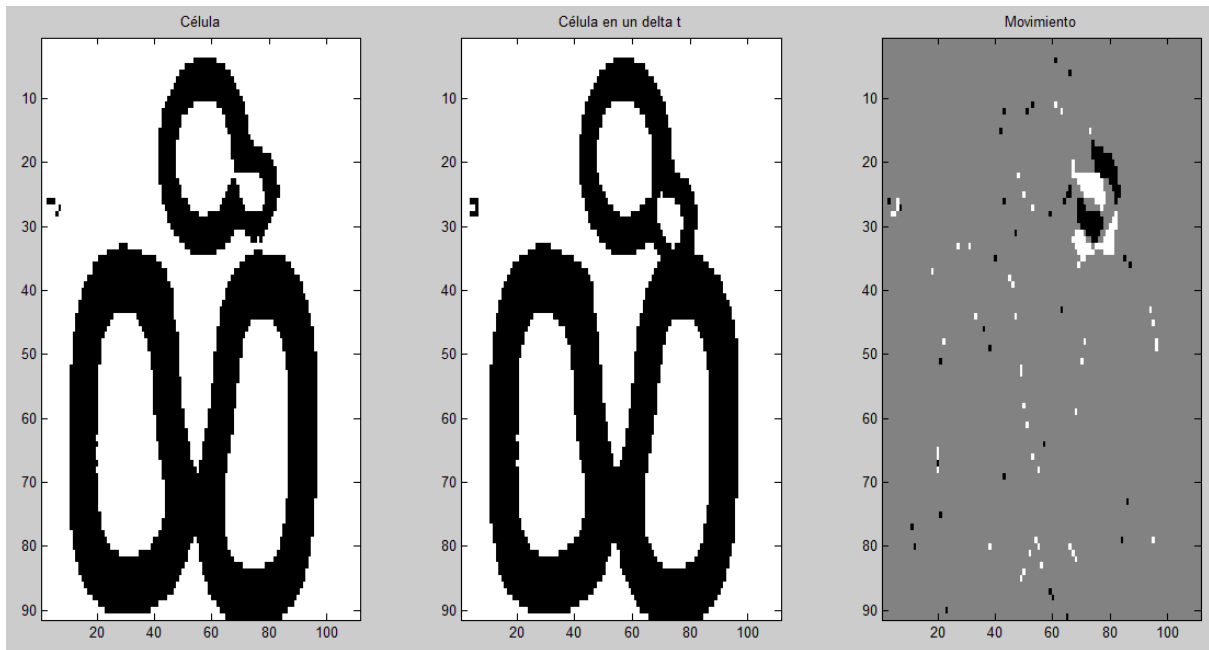


Imagen 6 Tomando la célula madre como objeto de interés, vemos que si no se mueve, en el análisis de movimiento por sustracción de fondo se ve sólo la célula hija.

Las imágenes de campo claro fueron tomadas con un microscopio invertido usando un objetivo de 20X. En estas imágenes se observa que durante el experimento hay distintas clasificaciones que caracterizan a las células y a las trampas y para determinar su estatus se propone un análisis que contemple los siguientes procesos:

1. *Determinar si hay célula*. Esto debe indicar en primer lugar si en la trampa analizada ya ha sido atrapada una célula individual discriminando las que pudieran ser atrapadas en grupos, esto con el fin de saber en qué momento la trampa es de interés para el análisis.
2. *Determinar si la célula se mueve o crece*. Este proceso ya forma parte del análisis una vez que ha sido validado que existe una célula en la trampa, y debe determinar si en un instante de tiempo $t + \Delta t$ la misma célula presenta un cambio en su posición o en su tamaño.
3. *Determinar si la célula se reproduce*. Se realizará el reconocimiento, y monitoreo de las células en estudio, lo cual tendrá como resultado un conteo de las generaciones producidas por cada célula madre clasificada anteriormente como “célula de interés” en el tiempo que dure el experimento.

Cada proceso del sistema de visión debe tomar en consideración las limitaciones energéticas y de procesamiento a las que estará sujeta la plataforma. A continuación describiremos el diseño y la selección del método de análisis para cada módulo de procesamiento.

IV.3.c. Determinación de si ya hay célula en la trampa (propuestas y comparación entre algoritmos)

Se realizará un análisis por trampa del dispositivo microfluídico para determinar si una célula ha sido atrapada de manera individual, misma que a partir de este momento será clasificada como “*célula de interés*”, con esto se llevará un conteo en el que a partir de un umbral (por ejemplo cien células de interés en el dispositivo), ayudará a determinar la acción de control sobre el comienzo del algoritmo de monitoreo y reproducción.

Para la realización de este proceso se hizo una revisión de distintos métodos que pudieran detectar la presencia de la célula en la imagen, siendo los más atractivos la *Transformada de Hough* y los *métodos de sustracción de fondo*, cuyo funcionamiento, análisis y desempeño trataremos más adelante.

En el caso de los algoritmos propuestos para la identificación de si ya hay célula en la trampa, se tomaron en cuenta los siguientes parámetros de evaluación:

Criterios para la evaluación del algoritmo por trampa	
Criterio	Método
¿Detecta que hay célula?	Clasificación de porcentaje correcto
¿El centro calculado es aproximado al centro de la célula?	Cálculo del error en distancia
¿Los bordes corresponden al límite de la célula?	Coficiente de Dice

Tabla 1 Criterios para la evaluación del algoritmo por trampa.

Además de esto es importante señalar que siempre se respetará la viabilidad de la implementación del método en un FPGA, por lo que si se observa que dos algoritmos tienen desempeños similares,

siempre se elegirá el que tenga una mejor relación $\text{desempeño} / \text{costo energético}$ o bien, el que su implementación no represente una arquitectura complicada o de recursos excedentes.

En las imágenes del experimento se presentan cuatro casos diferentes en los que se puede presentar la trampa en esta primera etapa de análisis:

- Caso 1. Trampa vacía
- Caso 2. Trampa con célula pequeña (15-18 px de diámetro)
- Caso 3. Trampa con célula de tamaño regular (19-30 px de diámetro)
- Caso 4. Trampa con célula en reproducción (no deseable, pero se debe considerar)

Entonces, para una correcta evaluación de las propuestas, los criterios de evaluación deben ser aplicados al algoritmo al resolver todos los casos posibles.

Tratamiento de la imagen previo al análisis:

Como primer paso se hizo una binarización de datos tomando en cuenta un nivel de umbral del 50%, es decir, de los límites de luminancia del histograma de la escena se toma el valor medio entre el nivel superior y el inferior, así, si nuestra resolución radiométrica es de 256 valores y nuestro nivel de umbralización 0.5, significa que el nivel de umbralización será:

$$\text{umbral} = (\text{Límite superior} - \text{Límite inferior}) * \text{Nivel de umbralización}$$

$$\text{umbral} = (255 - 0) * 0.5$$

$$\text{umbral} = 128$$

Entonces se binariza la imagen de forma que a partir de este valor hacia el cero se toma como negro (0) y de igual manera de este umbral hacia el límite superior los valores se toman como blanco (1). Con esto obtenemos una primera segmentación de la escena, donde se observa que los bordes de las estructuras en la imagen están bien definidos.

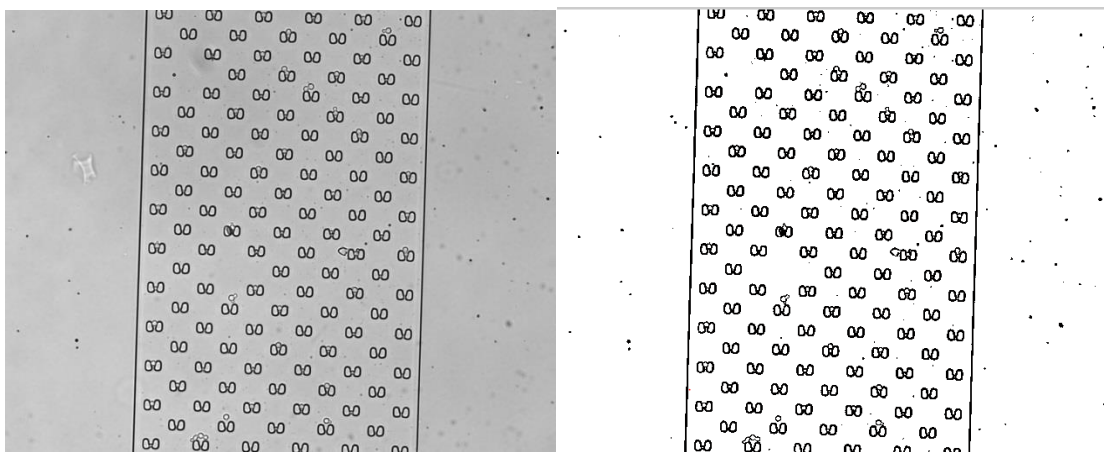


Imagen 7 En esta figura observamos a la izquierda la imagen original del dispositivo microfluídico y a la derecha la misma imagen después de haber sido umbralizada.

Obtenido esto, se procedió a poner un ID a cada trampa del dispositivo microfluídico para tener un mejor control del análisis de su comportamiento durante la experimentación, por lo que en el futuro al hacer referencia a un número de trampa, se puede consultar la siguiente imagen como guía para observar su posición en el dispositivo:

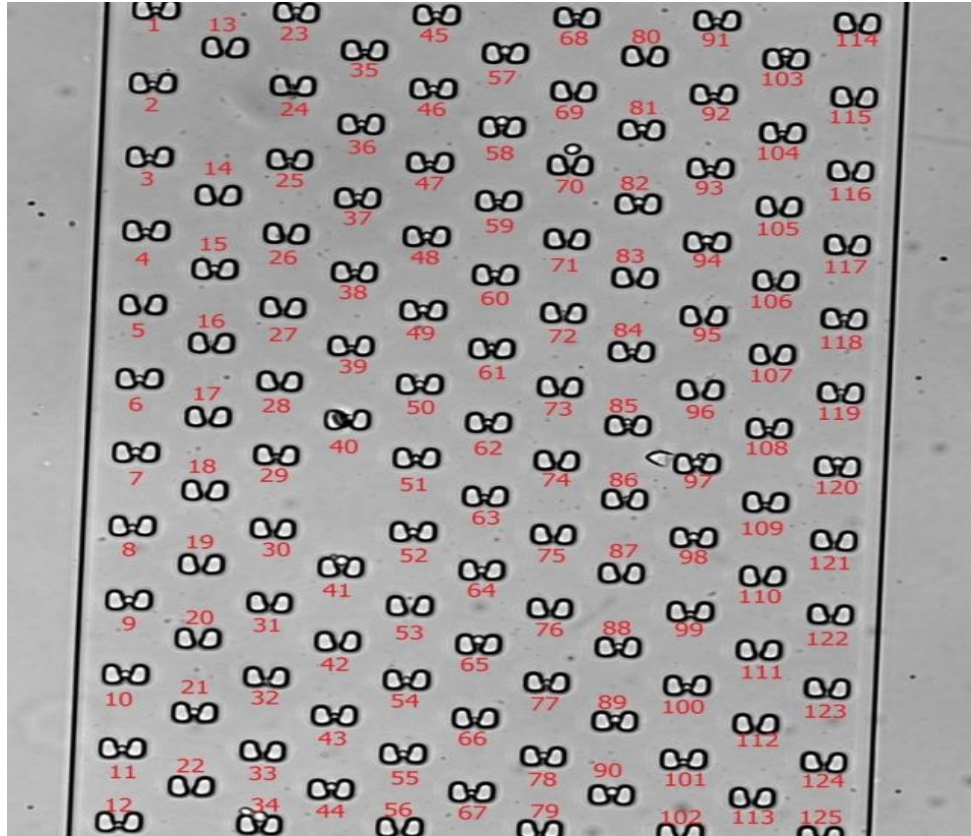


Imagen 8 Dispositivo microfluídico donde vemos el identificador de cada trampa.

De aquí hacemos un recorte de la imagen para hacer el análisis por trampa. Estos recortes se hicieron de forma que se pudiera aislar una sola trampa, pero que tuviera un margen suficiente para observar lo que sucede en su entorno cercano, por lo que se optó por dividir el dispositivo microfluídico en imágenes de 90x110 px, quedando estos recortes como se muestra en la figura x.

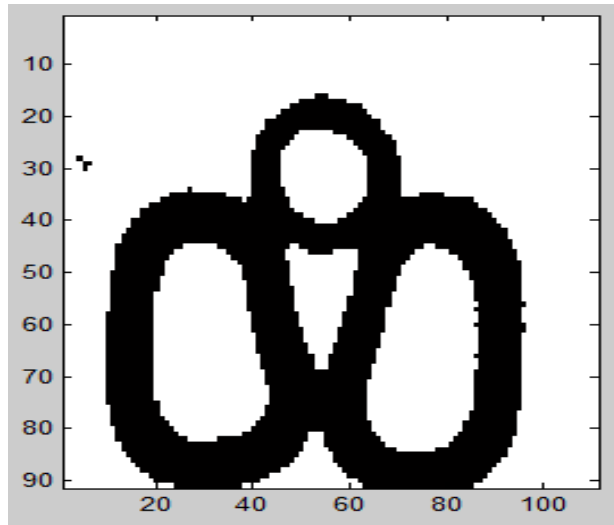


Imagen 9 Recorte de una imagen binarizada de 90x110 px , donde se observa una trampa con célula.

Una vez teniendo la imagen binarizada de cada una de las trampas, se procede a desarrollar las propuestas de solución que se describen a continuación.

Propuesta 1: Transformada de Hough:

La *Transformada de Hough* es un método de análisis global que se diseñó para detectar líneas rectas y curvas a partir de las posiciones de n puntos. La ventaja de esta técnica es la robustez de los resultados de segmentación conseguidos al aplicarla; sin embargo su coste computacional es elevado.

Generalmente se realiza primero una detección de bordes, y luego se aplica la transformada, aunque en el caso de nuestra imagen, se aplicó la transformada sobre la imagen binarizada.

Detección de círculos: La ecuación del círculo tiene tres parámetros, dos para el centro de la circunferencia: (a,b) y uno para el radio: (r).

$$(x - a)^2 + (y - b)^2 = r^2$$

Por lo tanto, el espacio de parámetros para la *Transformada de Hough* es de dimensión tres. El procesamiento se hace para cada punto de la imagen donde se vota por los posibles círculos a los que podría pertenecer ese pixel. Terminando este proceso se buscan los picos de los acumuladores y se encuentran aquéllos que tienen mayor probabilidad de ser circunferencia.

Debido a que la forma de las células por lo general es circular y con esta transformada podemos identificarlas, se hizo la propuesta de ver el desempeño de este algoritmo en los distintos casos de estudio de las células en este proyecto, por lo que se desarrolló en MATLAB obteniéndose resultados como los que se muestran en la Imagen 10.

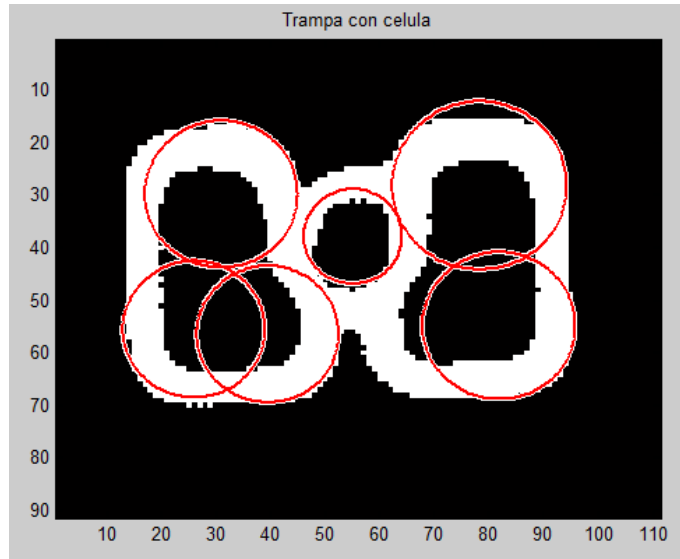


Imagen 10 Trampa donde se observa que la transformada de Hough presenta falsos positivos.

La Imagen 10 fue obtenida haciendo el análisis sobre la imagen binaria con la *Transformada de Hough* para círculos con un radio de 9 a 20 píxeles.

Debido a la presencia de falsos positivos pertenecientes a la trampa y asumiendo que conocemos las coordenadas de la posición de la misma (la cual está fija a lo largo del análisis), se decidió poner una condición a modo de filtro donde sólo se tomaran en cuenta aquéllas circunferencias cuyo centro coincida con el centro de la trampa (zona de interés). Con lo que los resultados quedan como se muestra a continuación:

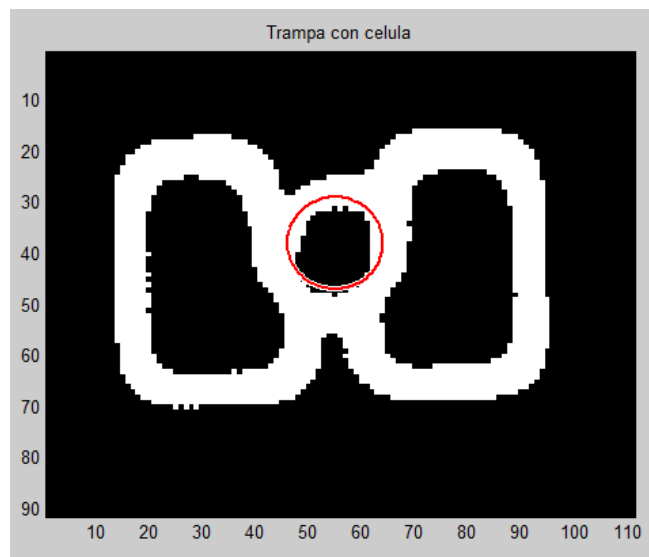


Imagen 11 Trampa donde se identifica la célula de interés a través de la Transformada de Hough.

Cabe mencionar que de esta manera el algoritmo hace el procesamiento de todos los píxeles de la imagen para determinar las posibles circunferencias a las que pudieran pertenecer, lo cual genera una carga computacional importante.

Para reducir la carga computacional, se propuso hacer una segmentación de la célula a través de la resta de la imagen con la trampa, con lo que la transformada se aplicaría a un número menor de píxeles. Esto nos genera los resultados que se muestran:

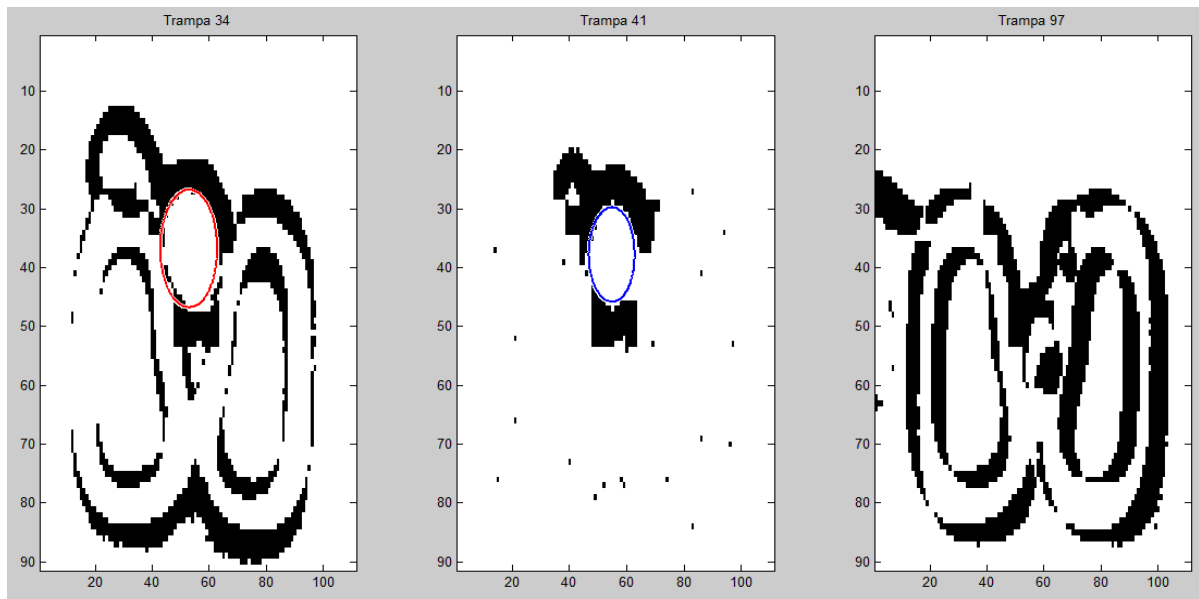


Imagen 12 Imágenes de resta sobre las que se corrió el algoritmo, como vemos, en la tercera hay un cambio significativo en la luminosidad de la escena, lo que genera que los bordes de la trampa produzcan efectos no deseados.

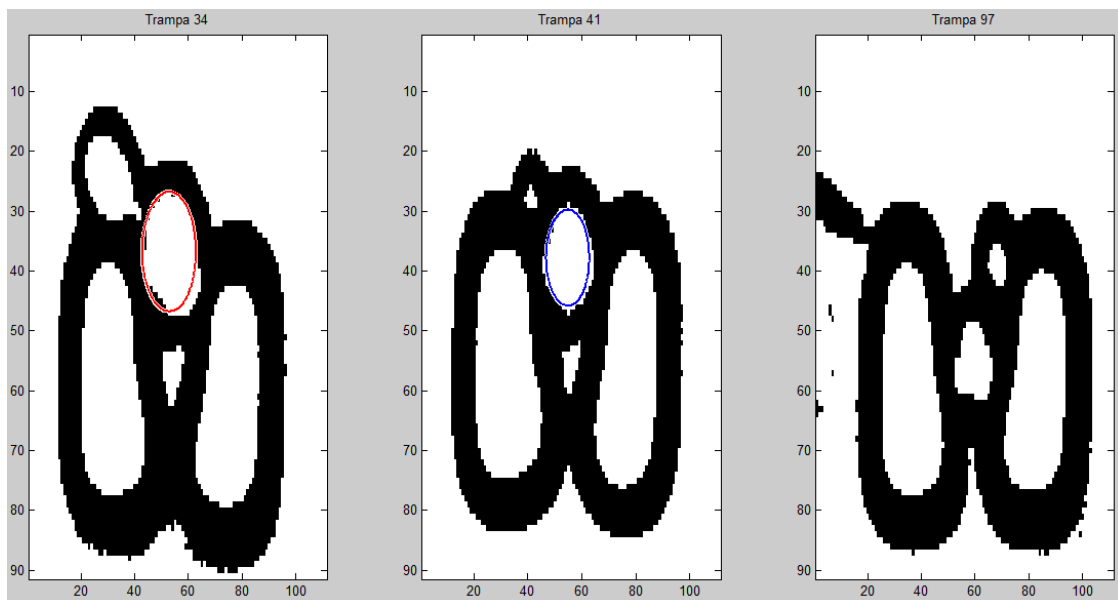


Imagen 13 En esta imagen observamos los mismos resultados vistos en la trampa original.

De aquí se observa que a pesar de que hacer la resta de los píxeles de la trampa para reducir las operaciones, esto genera que el algoritmo tenga errores en el reconocimiento de las células que se confunden con los bordes de las trampas (camuflaje), por lo que se decidió ver el comportamiento del

algoritmo sin usar diferencias de imágenes, y los resultados obtenidos en los diferentes casos se reportan a continuación.

Solución con Transformada de Hough

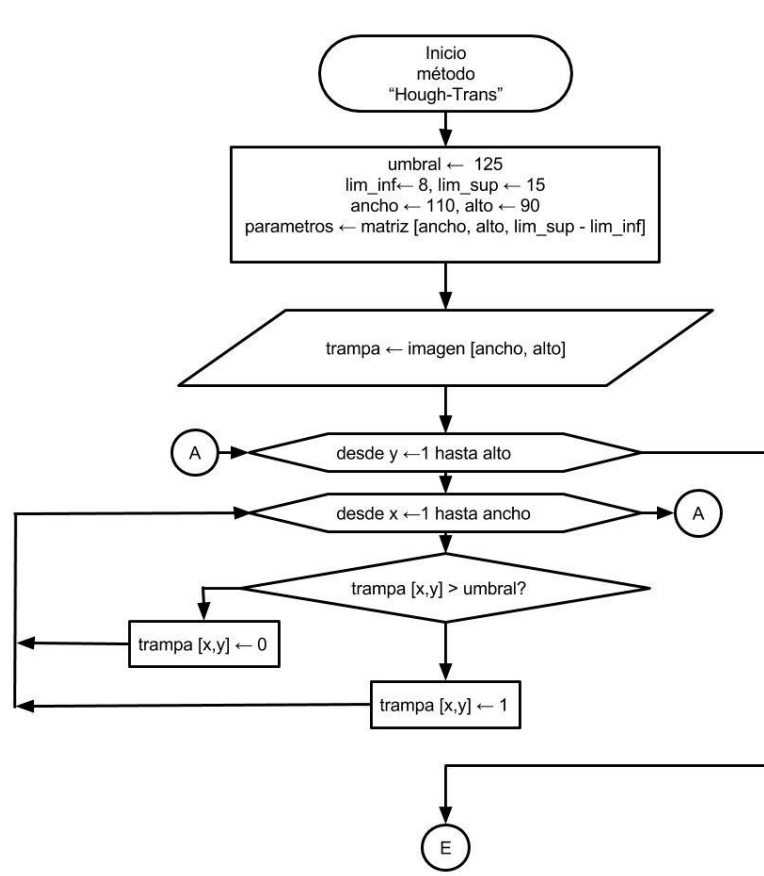
Descripción de la solución:

Este programa detecta los posibles círculos dentro de la imagen (las células tienen esta forma), discriminando aquéllos cuyo centro no coincide con el área de interés que indica que la célula está dentro de la trampa.

Requerimientos:

Tomando en cuenta una imagen de 110x90 px, este algoritmo necesita garantizar que el centro de la trampa coincide con el centro de la imagen, con una tolerancia de ± 15 píxeles en dirección x, y ± 15 píxeles en dirección y, el rango de radios utilizados, que funcionan para la mayoría de los casos es de entre 8 y 15 píxeles.

Diagrama de flujo:



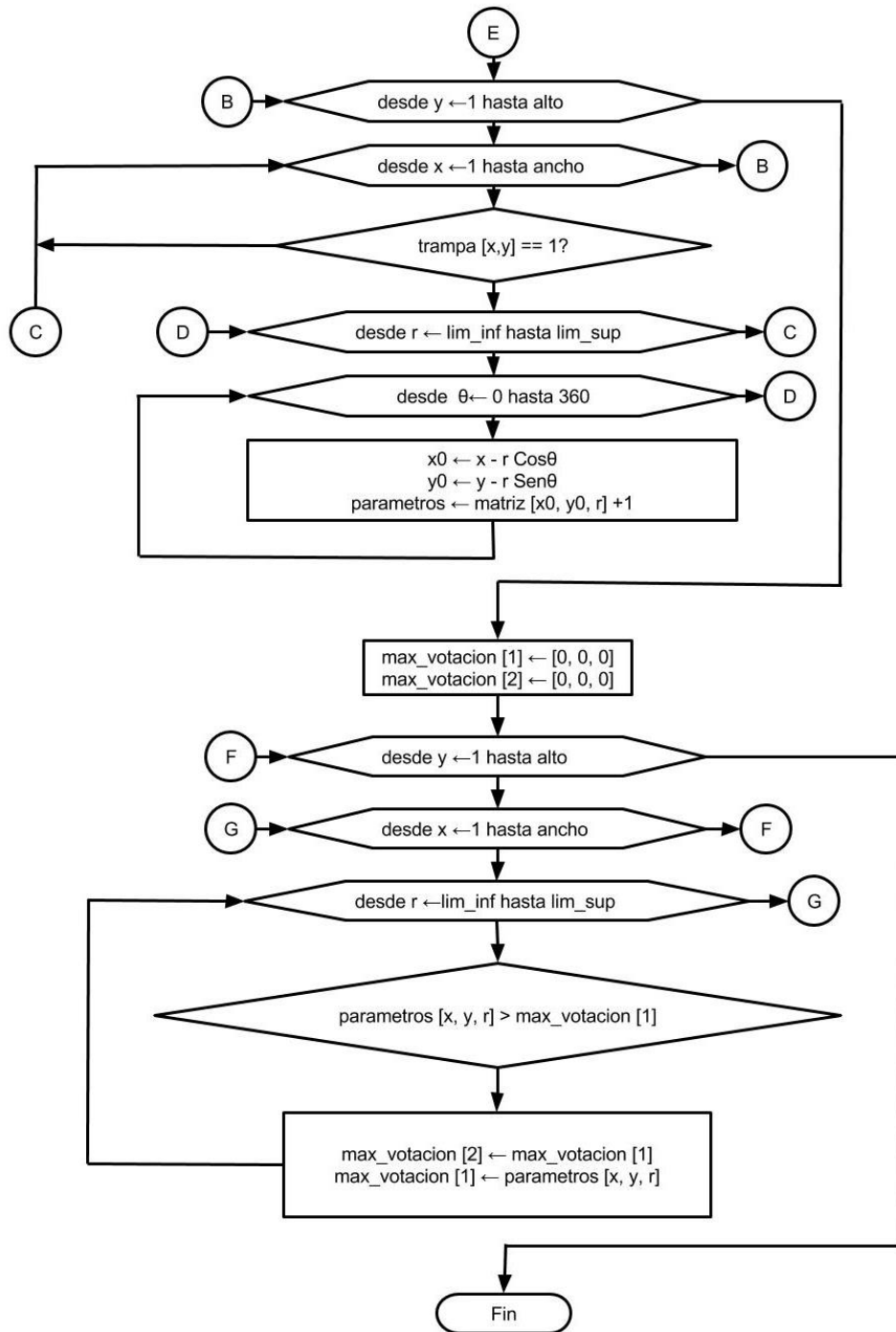


Diagrama 4. Algoritmo que utiliza la Trasmformada de Hough para detectar formas circulares de entre 8 y 15 pixeles de radio (lim_inf ; lim_sup) en una imagen binarizada a partir de un umbral. Los parámetros de los dos círculos con mayores votaciones se almacenan en las variables $max_votacion [1]$ y $max_votacion [2]$. Como vemos, entre mayores sean los rangos del espacio de parámetros, la complejidad aumenta al incrementar las iteraciones necesarias para hacer las votaciones.

Resultados:

Caso 1. Trampa vacía. (Trampas 14, 31 y 121)

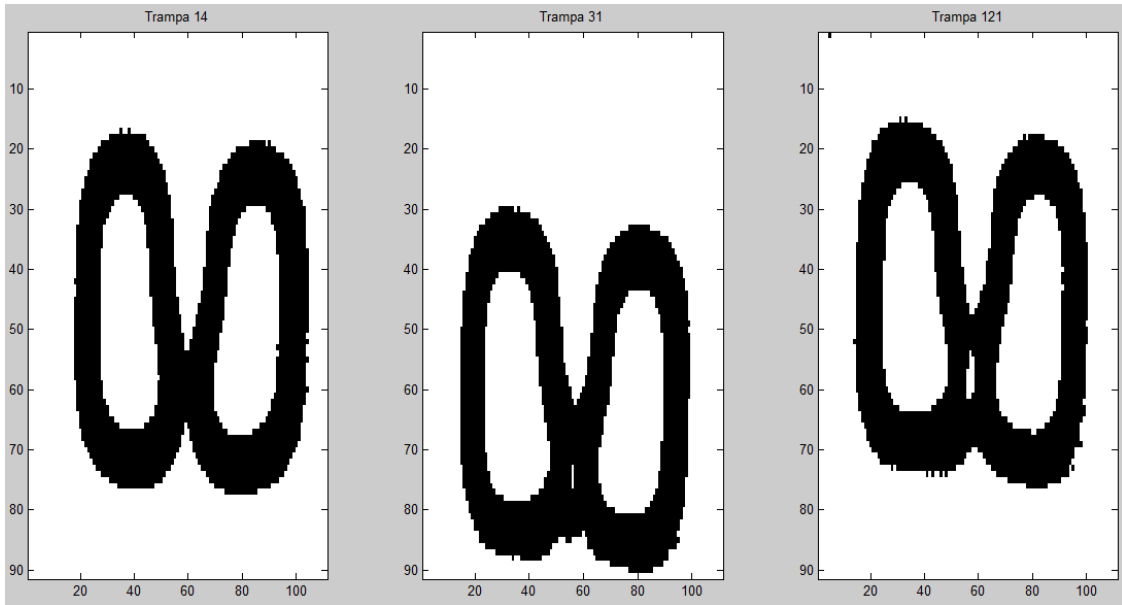


Imagen 14 Vemos que en trampas vacías no hay falsos positivos.

Caso 2. Células pequeñas (15-18 px de diámetro). (Trampas 39, 61, 108)

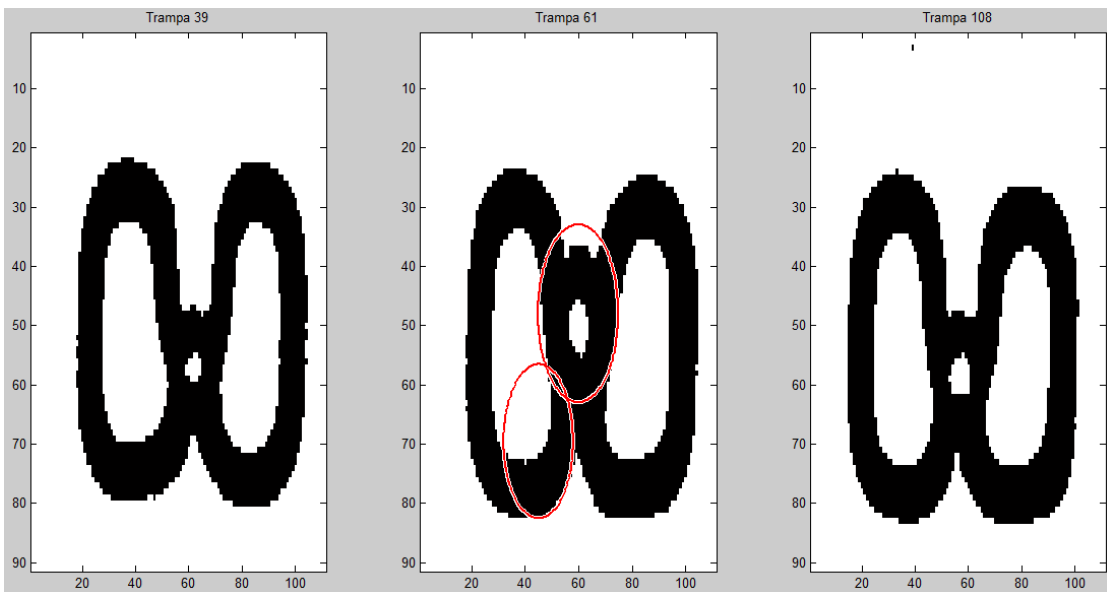


Imagen 15 En esta imagen observamos que el algoritmo tiene problemas con radios pequeños.

Si cambiamos el rango de los valores de trabajo para el radio de la Transformada de Hough, poniendo la búsqueda de 6 a 15 px tiene una mejor detección de estos casos, sin embargo este cambio de rango de radios podría generar problemas en el resto de los casos de estudio.

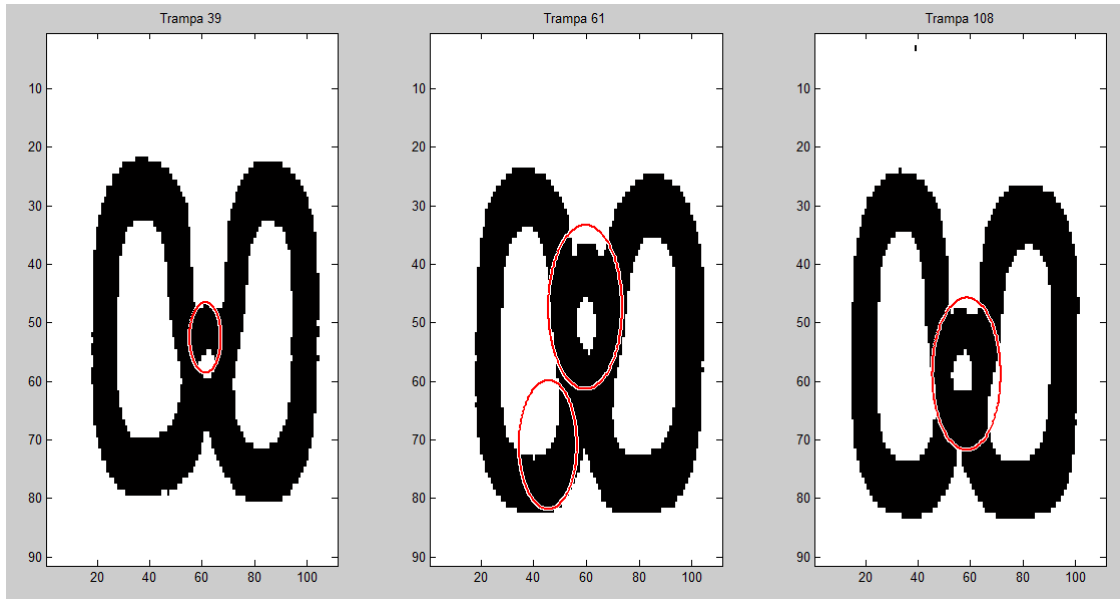


Imagen 16 En esta imagen se observa que si hacemos un cambio en los valores de los radios parece haber una mejor detección en las células pequeñas.

Caso 3. Células regulares (19-30 px de diámetro). (Trampas 7, 50, 120)

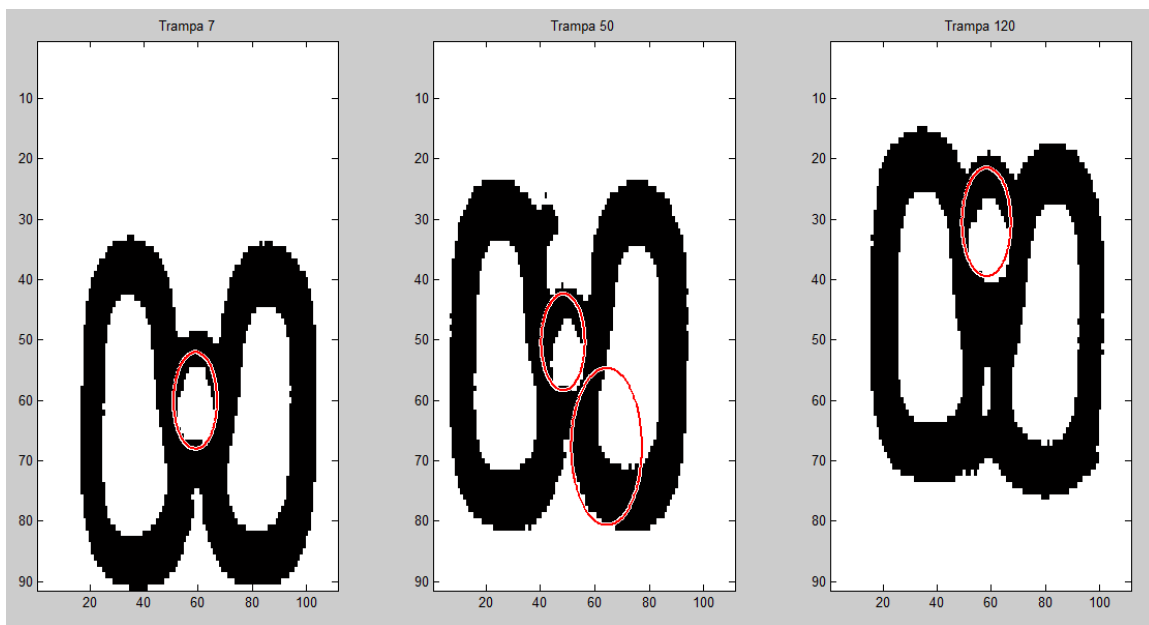


Imagen 17 Detección de células de tamaño regular con transformada de Hough.

Caso 4. Células en reproducción (Trampas 34, 41, 97)

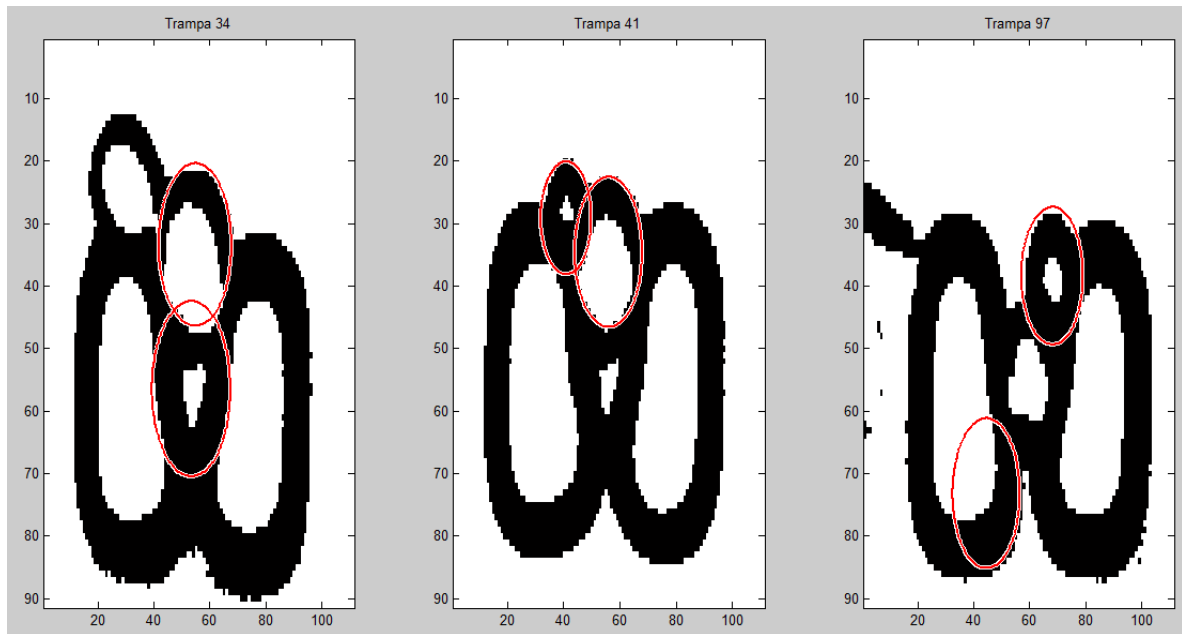


Imagen 18 En esta imagen vemos la detección de células en reproducción.

Además de los casos típicos, se hizo la elección al azar de otras células de análisis para tener una mayor población de estudio:

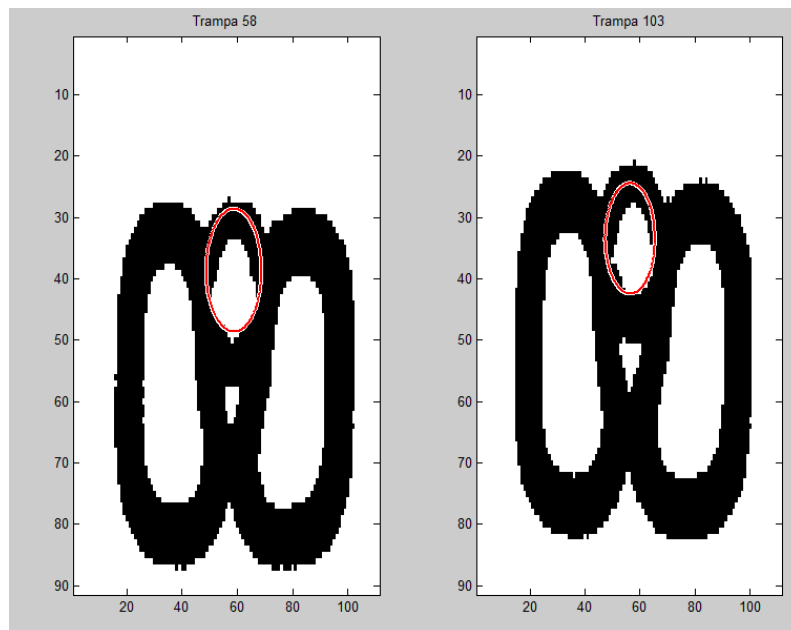


Imagen 19 Análisis de figuras circulares por la transformada de Hough.

Una vez vistos los casos de estudio se procede a hacer una evaluación del algoritmo.

Análisis de Resultados:

Criterio1. Detección de célula

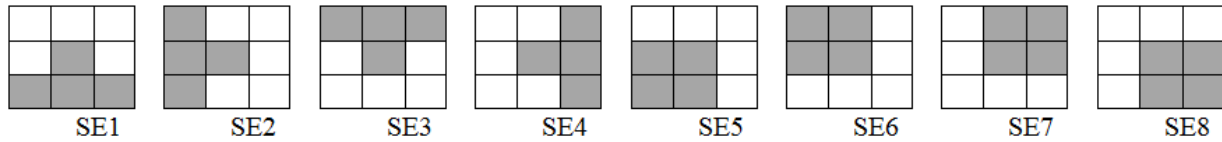
TP=10, TN=3, FP=4, FN=4

$$PCC = \frac{TP + TN}{TP + FP + TN + FN} = \frac{10 + 3}{10 + 4 + 3 + 4}$$

$$PCC = 0.6190 = 61.9\%$$

Criterio2. Cálculo del error de distancia entre centros

Se hace el adelgazamiento de los bordes a través de la Transformada de Hit or Miss con los siguientes elementos estructurales:



Con los que nos quedan imágenes como las mostradas en la Imagen 20.

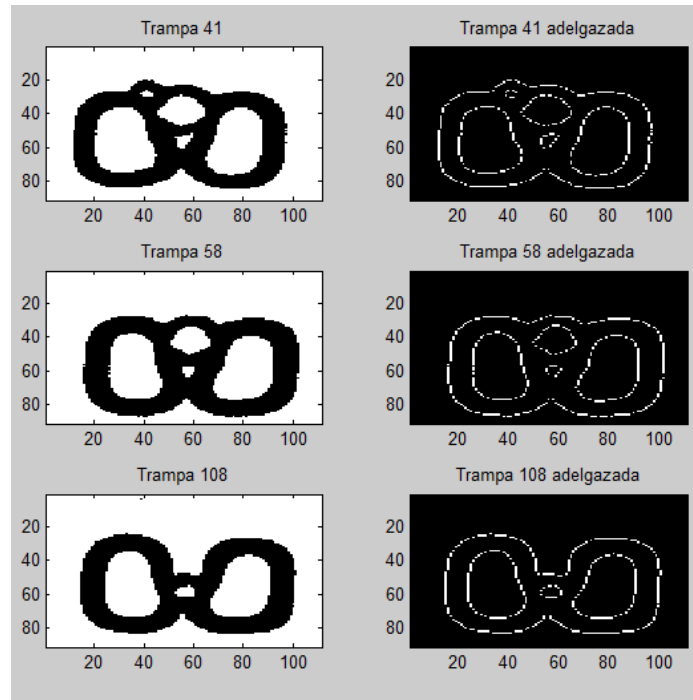


Imagen 20 Adelgazamiento de bordes a través de coincidencia estructural o transformada de *Hit or Miss*, presentada en los distintos casos de análisis, de arriba hacia abajo: célula en reproducción, célula de tamaño regular y célula pequeña.

A partir de estas imágenes procedemos a calcular el centro del área encerrada en el interior de los bordes que coinciden con los TP calculados por la Transformada de Hough. Esto se hace a través de la sumatoria de las coordenadas en x y en y de los pixeles del contorno interno obtenido:

$$x = \frac{\sum_{i=1}^n x_i}{n}$$

$$y = \frac{\sum_{j=1}^n y_j}{n}$$

n = número de pixeles que pertenecen al contorno

x, y | pixel(x, y) ∈ contorno o borde

El cálculo del error en ambas direcciones se hace con la fórmula del error relativo:

$$\%e_{relativo} = \left| \frac{\text{valor medido} - \text{valor real}}{\text{valor real}} \right| \times 100\%$$

$$\%e_{distancia} = \sqrt{\%e_{relativo}^2 x^2 + \%e_{relativo}^2 y^2}$$

Trampa	Centro real (x,y)	Centro calculado por Hough (x,y)	% error x	% error y	% de error en distancia
61	60, 51	60,48	0	5.8	5.8
7	59, 60	59,60	0	0	0
50	50, 53	48,50	4	5.7	6.96
120	60, 34	58,30	3.4	11.8	12.28
34 (madre)	55, 38	55,33	0	13.2	13.2
41 (madre)	55, 38	56,35	1.8	7.9	8.1
41 (hija)	41,28	41,29	0	3.6	3.6

97 (hija)	68, 39	68,38	0	2.6	2.6
58	59, 41	59,38	0	7.9	7.9
103	58, 35	57,33	1.7	5.7	5.95
Promedio	-	-	-	-	6.64%
Máximo	-	-	-	-	13.2%

Tabla 2 Análisis de error de cálculo de centros del Algoritmo por Transformada de Hough.

Criterio3. Cálculo de similitud de bordes.

Para esta propuesta de solución este cálculo no aplica, pues de determinar que la Transformada de Hough es el algoritmo con mejor desempeño para las necesidades del sistema, no se hace necesario conocer los bordes de las células a diferencia de los algoritmos de sustracción de fondo que trabajan directamente con la relación entre pixeles.

Propuesta 2: Técnicas de sustracción de fondo

Estos métodos son de especial interés para esta aplicación debido a que su demanda de recursos computacionales es baja debido a que pueden aplicarse operadores lógicos o de baja complejidad como sumas y restas para extraer características en imágenes binarias.

Se debe tomar en cuenta que la resta es muy sensible al ruido, por lo que su capacidad de anti-interferencia es pobre, así que en el planteamiento de la solución se debe considerar el ruido al que normalmente está sujeta la captura de imagen, así como variaciones en la iluminación, movimientos en objetos estáticos, o sombras proyectadas.

Diferencia entre dos imágenes consecutivas: Sea I_k el valor de la k -ésima imagen de una secuencia. I_{k+1} es el valor de la $k+1$ -ésima imagen de la misma secuencia. La imagen de diferencia absoluta se define como:

$$I_d(k, k + 1) = |I_{k+1} - I_k|$$

Aplicando lo anterior a la imagen binarizada para hacer una primera sustracción de fondo, en este caso de la trampa para determinar si hubo un cambio, lo cual podría interpretarse como la presencia de una célula, el resultado se observamos en la Imagen 21.

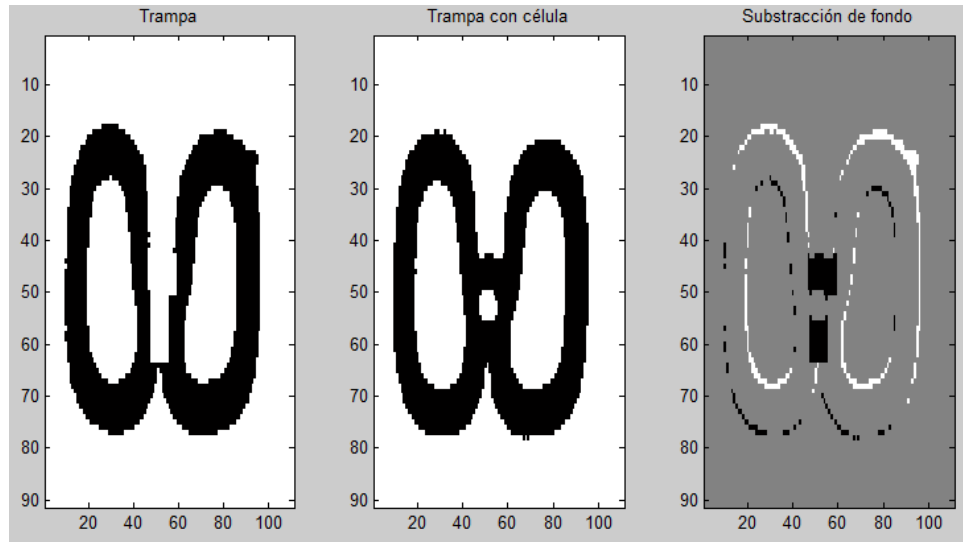


Imagen 21 Se muestra la resta de la trampa 59 con la imagen sucesiva en el time lapse.

Bajo estas condiciones podemos hacer varias propuestas, siendo la primera y más sencilla proponer un umbral de pixeles para detectar un cambio, esto es, si en la imagen en una zona de interés (por ejemplo las coordenadas del centro de la trampa) se detecta que hay un número de pixeles de cambio (diferentes de cero en la resta) por encima de un umbral establecido, entonces se determina que ya hay célula en la trampa.

Este análisis sin embargo, resulta muy susceptible al ruido en la imagen y la pregunta obligada que se hace es, ¿cómo sabemos que lo que hay en el centro de la trampa es efectivamente una célula? Para determinarlo se pensó en implementar un algoritmo de coincidencia estructural debido a que la forma de las células es conocida, sin embargo el tamaño puede variar y con ello la disposición y cantidad de pixeles en la imagen. Como solución de lo anterior se propone el uso de una *huella* estructural, cuya idea consiste en hacer una “plantilla de pesos” en la que el mayor valor lo tendrán los pixeles que sea más probable que pertenecen a una célula. A partir de esto podemos generar un umbral de coincidencia para determinar si lo que se presenta en el análisis es o no es parecido a lo que estamos buscando.

La huella se formó de la suma de probabilidad de aparición de pixeles en una célula, para lo cual se hizo un algoritmo cuyo funcionamiento mostramos a continuación:

1. Leer la imagen de la trampa
2. Leer la imagen de la trampa con la célula
3. $Célula = trampa - trampa \text{ con célula}$
4. Calculamos el centro de los pixeles de cambio
5. Recortamos la imagen alrededor de este centro
6. Repetimos n veces desde el paso 1 con distintas muestras
7. Hacemos la suma de imágenes haciéndolas coincidir en el centro

Corriendo el algoritmo anterior en 10 diferentes células escogidas al azar obtenemos los resultados que se muestran en la Imagen 22.

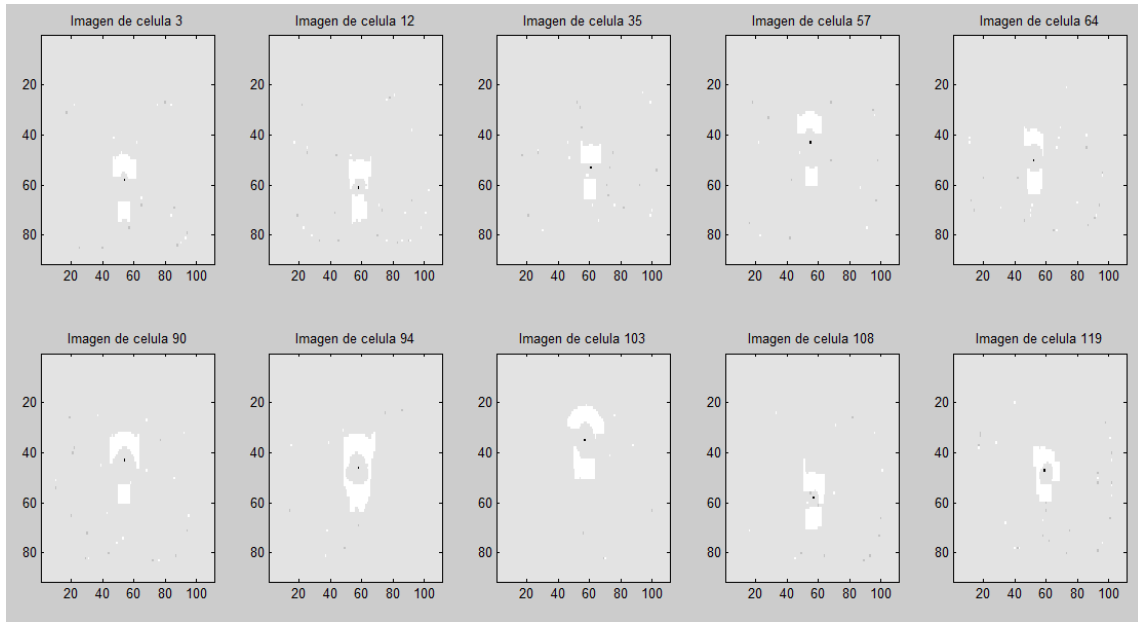


Imagen 22 Imagen donde se observa la resta de la trampa y el centro de pixeles de cambio calculado.

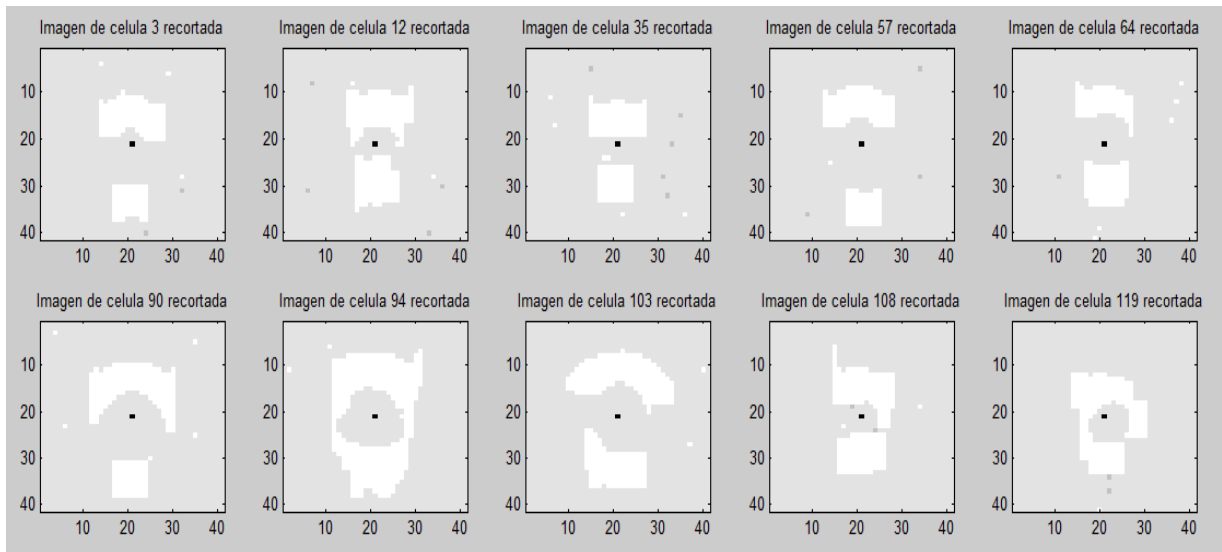


Imagen 23 Recorte de los pixeles de cambio alrededor de su centro (el centro es representado como el pixel en negro).

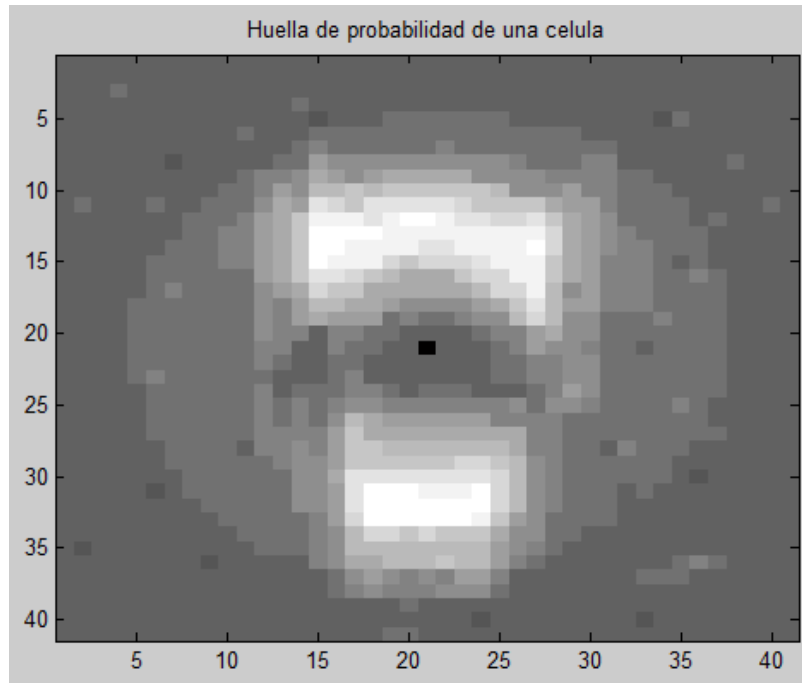


Imagen 24 Plantilla de pesos, donde entre más cercano al blanco es el color del pixel, es más probable que aparezca en la estructura de una célula.

A partir de esto podemos hacer un análisis probabilístico de los resultados obtenidos.

Solución con algoritmos de sustracción de fondo

Descripción de la solución:

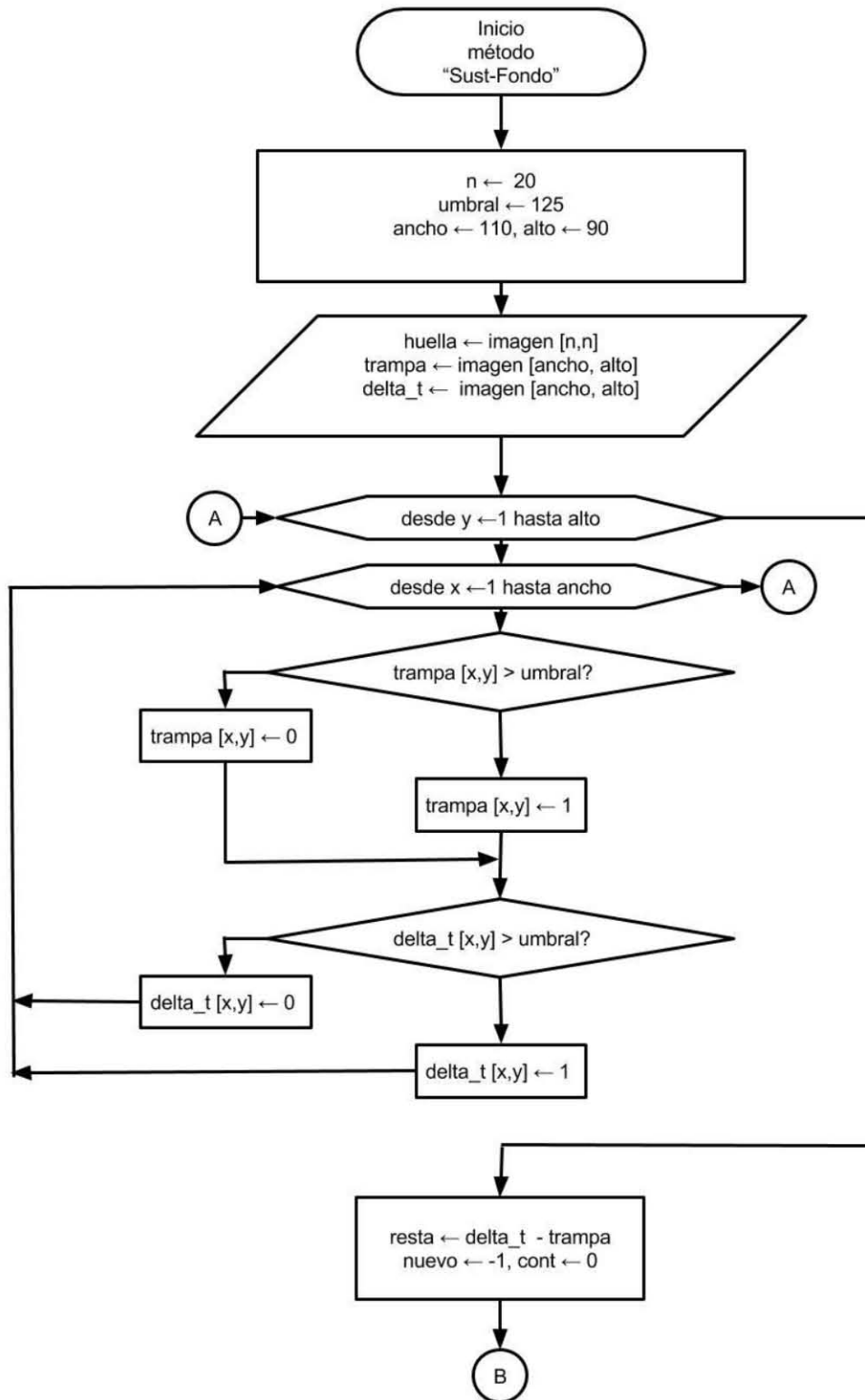
En esta solución se elimina la trampa de la imagen a través de una resta, con lo que se obtienen los pixeles candidatos a ser célula. Para determinar si pertenecen o no, se modifica la idea de coincidencia estructural haciendo una comparación de los pixeles candidatos a ser célula con una “plantilla de pesos” (o elemento estructurante), en los que se determina qué tanto se parece a la forma de una célula.

Dicha plantilla es una matriz generada a través de la suma de diversas células haciendo coincidir su centro, lo que nos proporciona una probabilidad de aparición de cada pixel perteneciente a una célula, misma que usamos como valor de peso para determinar un umbral y hacer la toma de decisión derivada de la pregunta ¿lo que tenemos representado en la imagen tiene similitud con la forma característica de la célula en estudio?

Requerimientos:

Se requiere que la trampa esté centrada en la imagen con una posición conocida y que se garantice que el dispositivo de captura y la trampa estarán fijos a lo largo de todo el experimento. Este algoritmo, al procesar directamente los pixeles, necesita que la imagen tenga un ruido de moderado a nulo para trabajar de la forma ideal.

Diagrama de flujo:



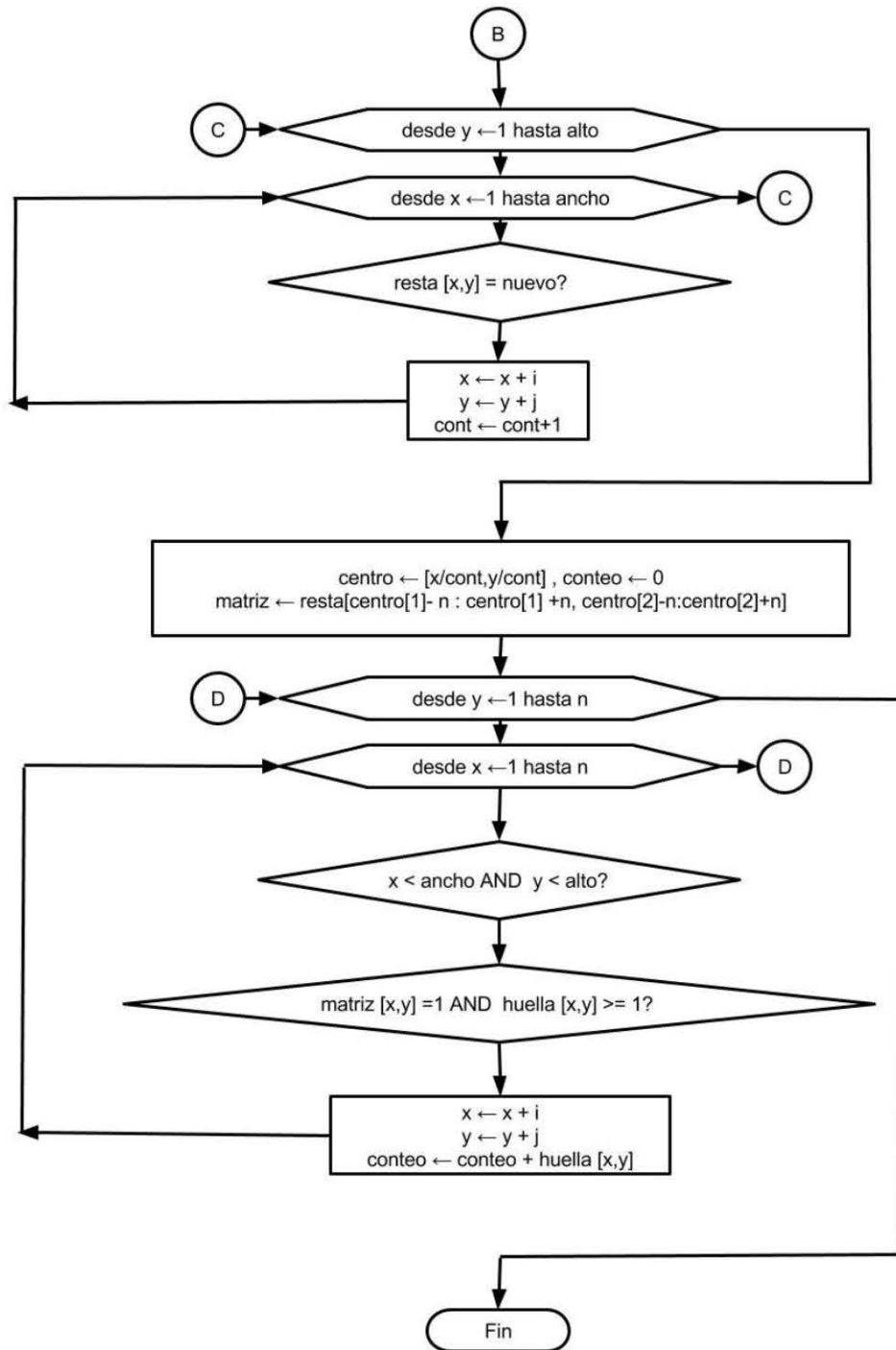


Diagrama 5. Diagrama del algoritmo por sustracción de fondo, donde al final nos entrega un conteo de coincidencia, el cual es usado junto con un análisis estadístico para determinar la presencia de una célula en la trampa de análisis de acuerdo a los niveles de coincidencia con la estructura de una célula (imagen de huella).

Resultados:

El umbral propuesto del contador para determinar que la estructura o cambio de pixeles analizados pertenece a una célula es un conteo mínimo de 1000 puntos.

Caso 1. Trampa vacía. (Trampas 14, 31 y 121)

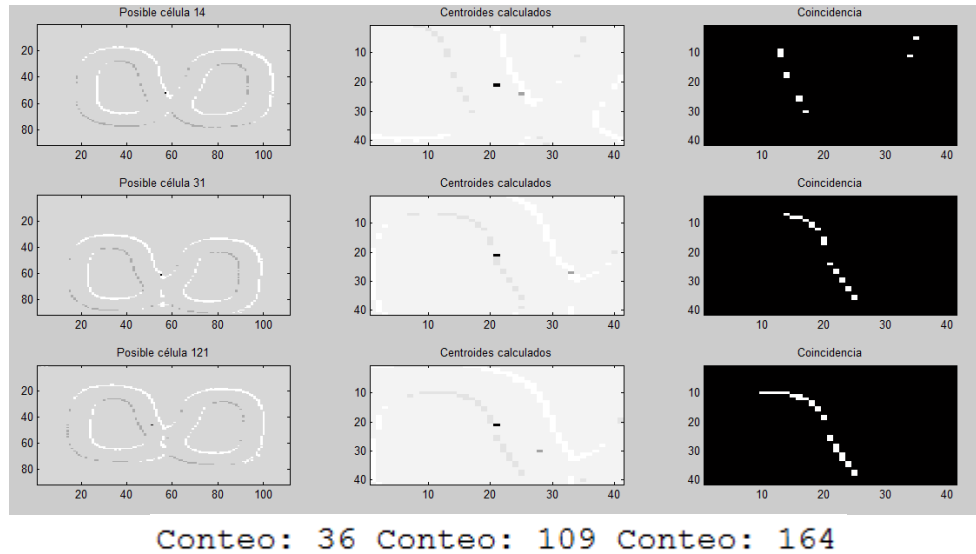


Imagen 25 Resultados para el caso 1.

Caso 2. Células pequeñas (15-18 px de diámetro). (Trampas 39, 61, 108)

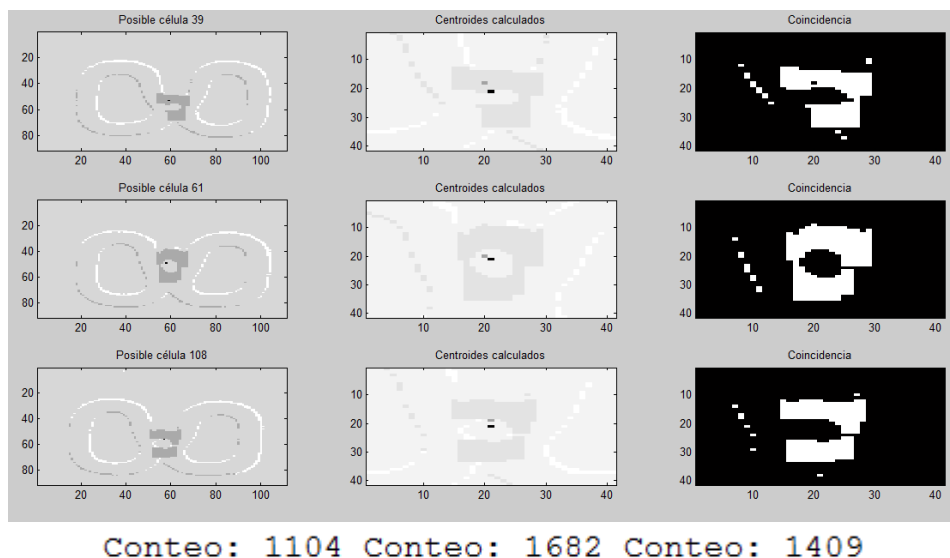
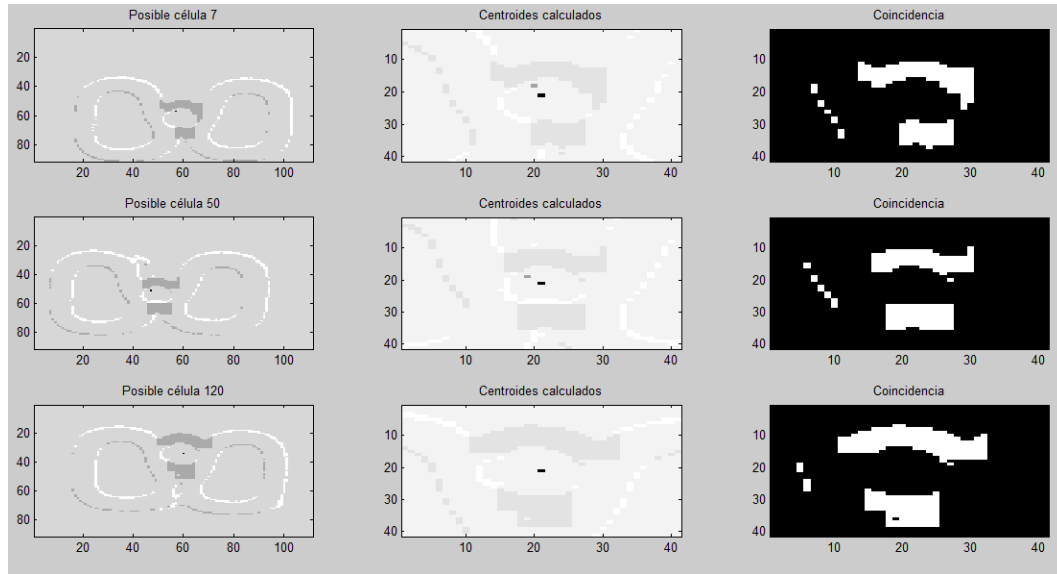


Imagen 26 Resultados para el caso 2.

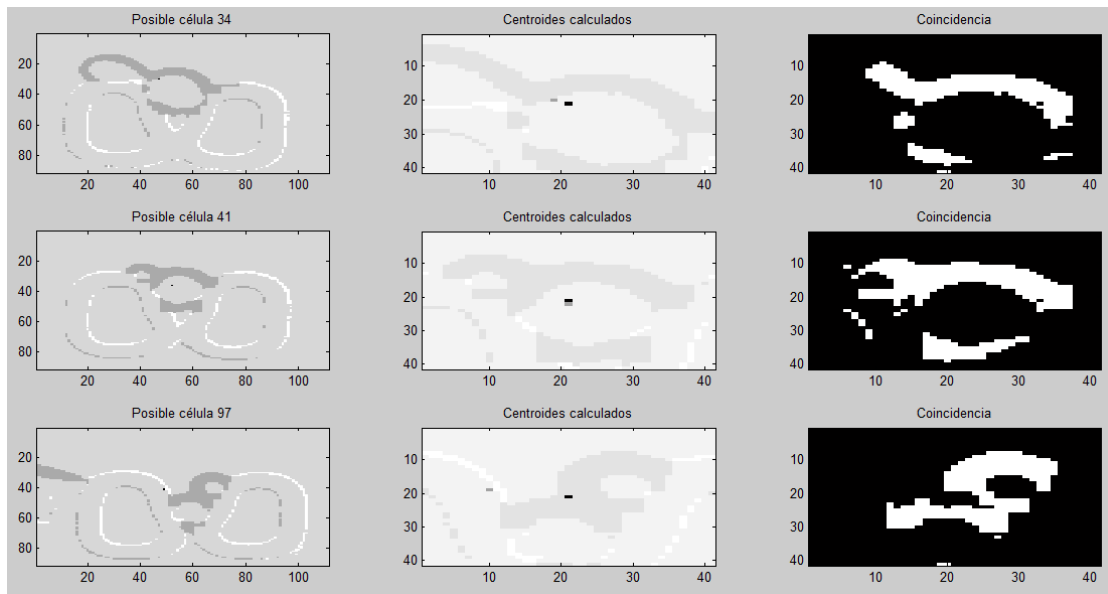
Caso 3. Células regulares (19-30 px de diámetro). (Trampas 7, 50, 120)



Conteo: 1283 Conteo: 1317 Conteo: 1629

Imagen 27 Resultados para el caso 3.

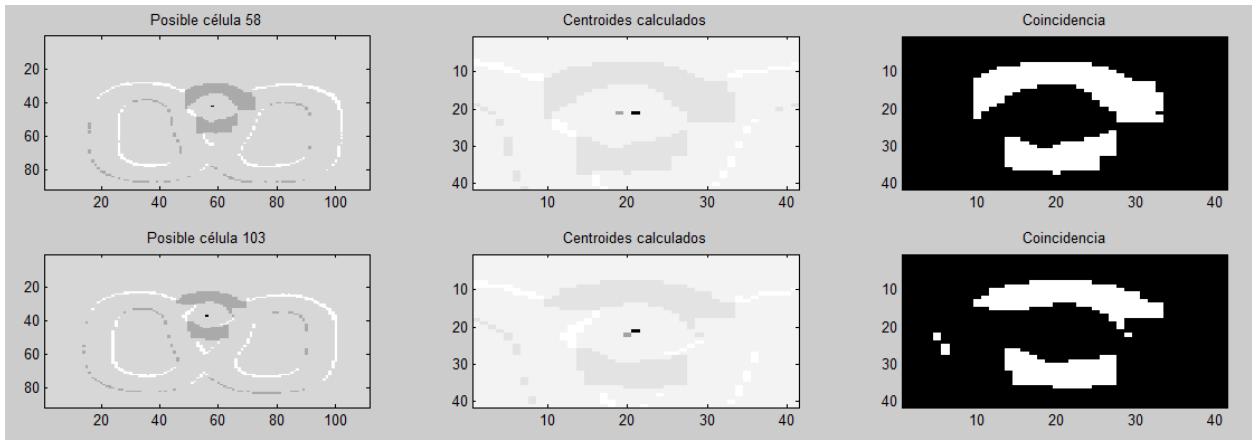
Caso 4. Células en reproducción (Trampas 34, 41, 97)



Conteo: 955 Conteo: 1447 Conteo: 960

Imagen 28 Resultados para el caso 4.

Además de los casos típicos, se analizaron los siguientes casos de estudio que fueron reportados también en el análisis de la propuesta 1:



Conteo: 1769 Conteo: 1662

Una vez vistos los resultados en los distintos casos de estudio se procede a hacer una evaluación del algoritmo.

Análisis de Resultados:

Criterio1. Detección de célula

En el algoritmo de la propuesta 1 se tomaron en cuenta las células hijas ya que la *Transformada de Hough* encuentra todas las circunferencias de la imagen, pero debido a las características en el diseño de este algoritmo, cuyo requerimiento es que localice la existencia de una célula madre atrapada en la trampa, no se tomará en cuenta la detección de las células hijas como falso negativo o verdadero positivo. El umbral propuesto para determinar si la estructura pertenece a una célula es con coincidencia mínima de 1000 puntos.

TP=9, TN=3, FP=0, FN=2

$$PCC = \frac{TP + TN}{TP + FP + TN + FN} = \frac{9 + 3}{9 + 0 + 3 + 2}$$

$$PCC = 0.8571 = 85.71\%$$

Criterio2. Cálculo del error de distancia entre centros

Para esto utilizaremos el mismo método de cálculo de centros que explicamos anteriormente en la propuesta 1, para todos los TP.

Trampa	Centro real (x,y)	Centro calculado por sustracción de fondo (x,y)	% error x	% error y	% de error en distancia
39	62,57	60,56	3.2	1.8	3.67
61	60, 51	59,50	1.7	1.7	2.4
108	57,59	57,58	0	1.7	1.7
7	59, 60	58,60	1.7	0	1.7
50	50, 53	49,53	2	0	2
120	60, 34	60,34	0	0	0
41	55, 38	52,35	5.5	7.9	9.63
58	59, 41	60,42	1.7	2.4	2.94
103	58, 35	57,36	1.7	2.9	3.36
Promedio	-	-	-	-	3.04%
Máximo	-	-	-	-	9.63%

Tabla 3 Error en cálculo de centros del algoritmo por sustracción de fondo.

Criterio3. Cálculo de similitud de bordes.

En el caso de este algoritmo es muy importante validar que la similitud de bordes sea próxima al caso ideal, pues ya que este método trabaja directamente con los pixeles de la imagen se debe determinar si es o no viable para etapas posteriores de procesamiento. Para ello analizaremos los *TP* (verdaderos positivos) utilizando la fórmula del *coeficiente de Dice* que ya hemos visto con anterioridad, sin embargo

como el conjunto B representado por la huella no es particular para cada caso de estudio, cambiamos el coeficiente del divisor (originalmente $|A| + |B|$) por $2 * |A|$ para que compare la intersección de los conjuntos con respecto de sí misma:

$$\%s = \frac{|A \cap B|}{|A|} * 100\%$$

Trampa	A	$A \cap B$	% s
39	212	191	90.09
61	264	253	95.83
108	212	203	95.75
7	196	184	93.88
50	190	177	93.16
120	275	260	94.55
41	409	313	76.53
58	338	318	94.08
103	275	267	97.09
Promedio	-	-	92.33%

Tabla 4 Similitud de bordes.

Conclusiones para la solución del algoritmo que determina la presencia de célula en la trampa:

Observando los resultados de evaluación de ambas propuestas mostrado en la Tabla 5 determinamos que para el caso de las imágenes que tenemos funcionaron mejor los métodos de substracción de fondo, además de que su implementación en FPGA es viable.

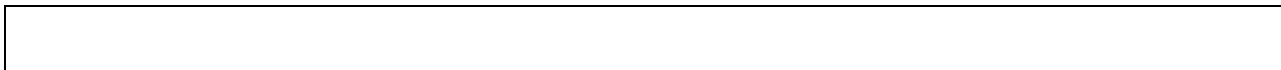


Tabla 5. Comparación de desempeño entre algoritmos		
Criterio	Transformada de Hough	Substracción de fondo
Detección	61.9%	85.71%
Error de distancia entre centros	6.64%	3.04%
Coincidencia de bordes	-	92.33%

Tabla 5 Comparación de desempeño entre algoritmos.

Para esto se debe tomar en cuenta que el algoritmo de substracción de fondo funciona debido a las características de las imágenes que tenemos, entre ellas las más significativas son:

1. No presentan ruido excesivo (la trasformada de Hough sería más robusta o quizá mejor solución en una imagen con ruido mayor).
2. Las estructuras que se presentan tienen formas regulares que pueden ser analizadas mediante métodos de probabilidad de coincidencia estructural.
3. La cantidad de pixeles que componen la imagen es adecuada para hacer el análisis pixel por pixel (en el caso de una imagen muy grande esto generaría un costo de procesamiento excesivo y no viable).

Por las razones anteriores la propuesta de este primer procesamiento del sistema de visión artificial representa una solución a la medida de las necesidades de la misión.

IV.3.d. Monitoreo de movimiento y crecimiento (propuesta y pruebas al algoritmo)

Que la célula se mueva o crezca durante el experimento son los casos que contempla este proceso, el cual forma parte del análisis posterior a la validación de la existencia de una célula en la trampa y debe determinar si dicha célula en un instante de tiempo $t + \Delta t$ cambia su posición o crece. Para ello es importante que la tasa de muestreo en la toma de imágenes sea tal que los cambios en la estructura celular sean graduales.

Siguiendo con la idea de coincidencia estructural en imágenes binarias se propone lo siguiente:

1. Determinar a través de la resta si hay un cambio significativo en la zona donde ya se había detectado presencia de célula
2. Si hay cambio en la zona, calcular el centroide del área de cambio
3. Si el centroide es parecido al de la célula que teníamos inicialmente, significa que ha crecido
4. Si el centroide no es parecido, comparar con la huella de la célula detectada anteriormente para ver la coincidencia estructural
5. Si el valor está por encima de un umbral, determinamos que los nuevos pixeles pertenecen a la célula que se ha movido
6. Si no hay coincidencia significa que hay un error por exceso de ruido en la imagen

Para la evaluación de este algoritmo los criterios que nos interesan son:

Criterio para la evaluación del algoritmo de monitoreo	
Criterio	Método
¿Detecta que hubo un cambio en la célula? ¿Detecta correctamente el caso? (movimiento o crecimiento)	Clasificación de porcentaje correcto

Tabla 6 Criterios de interés para el algoritmo de monitoreo de centros.

Algoritmo propuesto:

Coincidencia estructural por determinación de centros

Descripción de la solución:

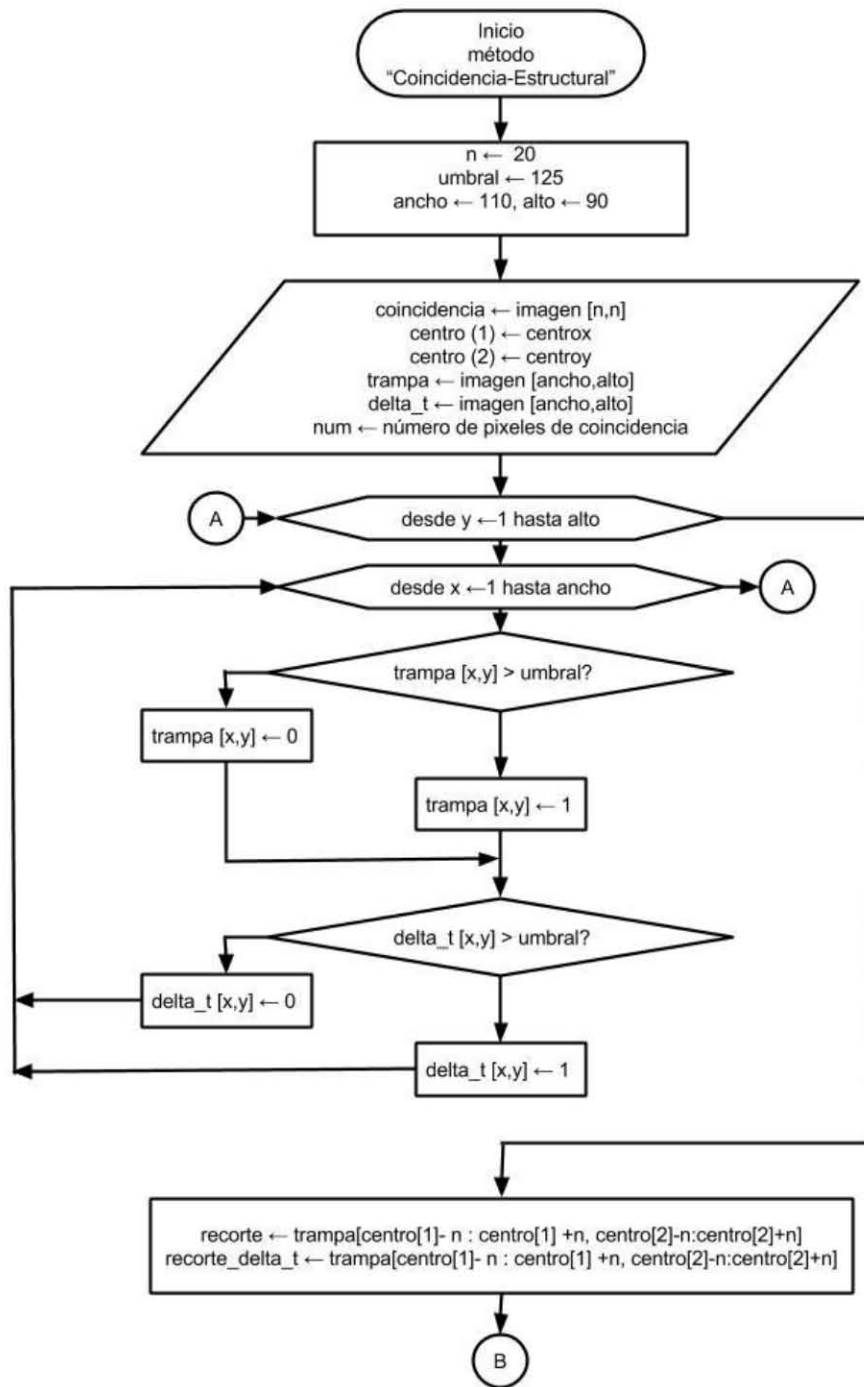
En esta solución se retoman los resultados obtenidos en la solución del proceso anterior, tales como la matriz de coincidencia generada a la cual se le etiqueta como huella de la célula, el número de pixeles que

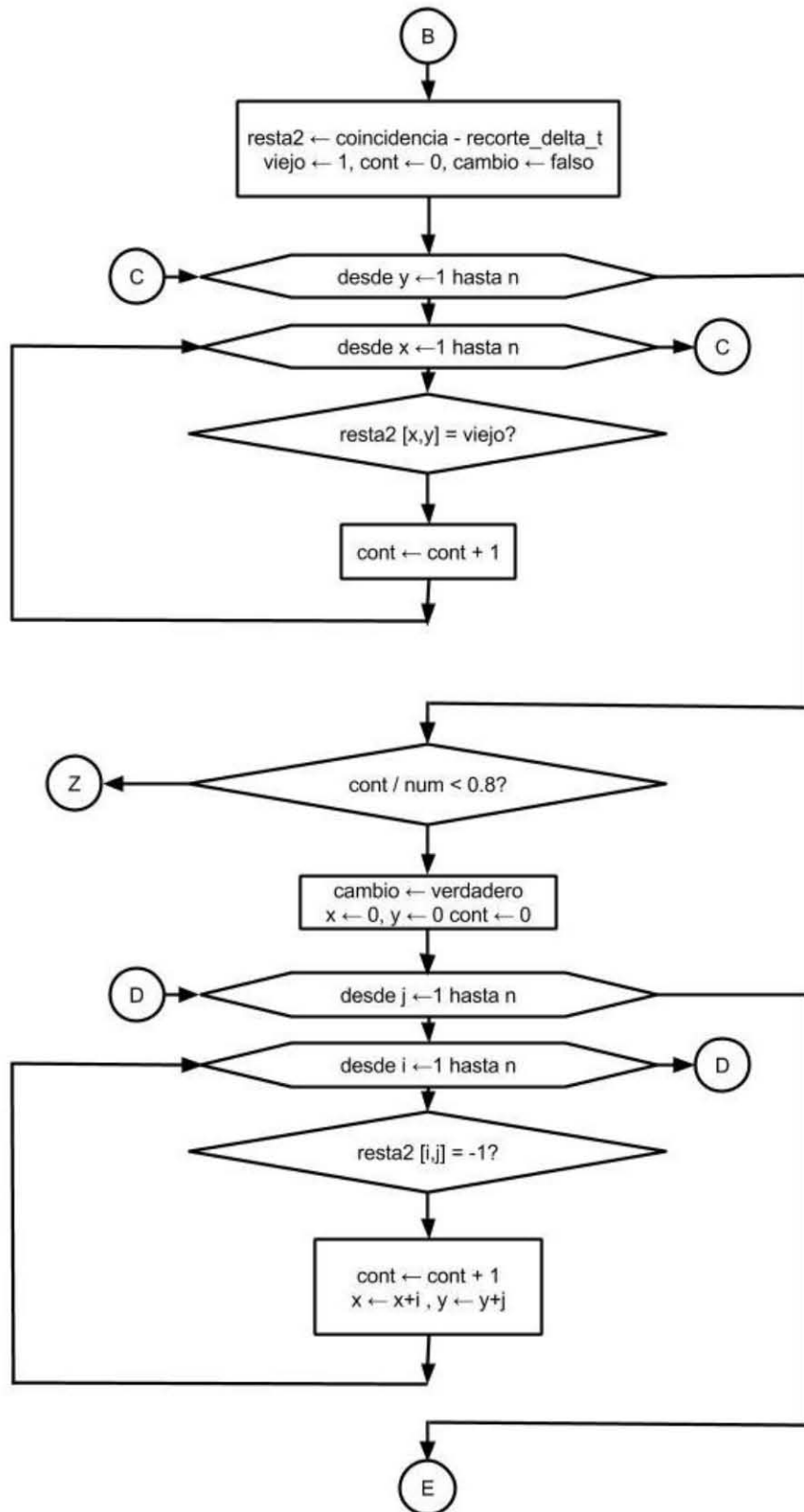
contiene y las coordenadas de su centro en la imagen. Con estos datos se hace un análisis del cambio de posición o tamaño de la estructura en la escena.

Requerimientos:

Al ser un algoritmo basado en diferencias y hacer relaciones directas entre píxeles, se requiere que, al igual que el algoritmo de substracción de fondo, se garantice la estabilidad de los componentes fijos y que el ruido en las imágenes sea de bajo a nulo.

Diagrama de flujo:





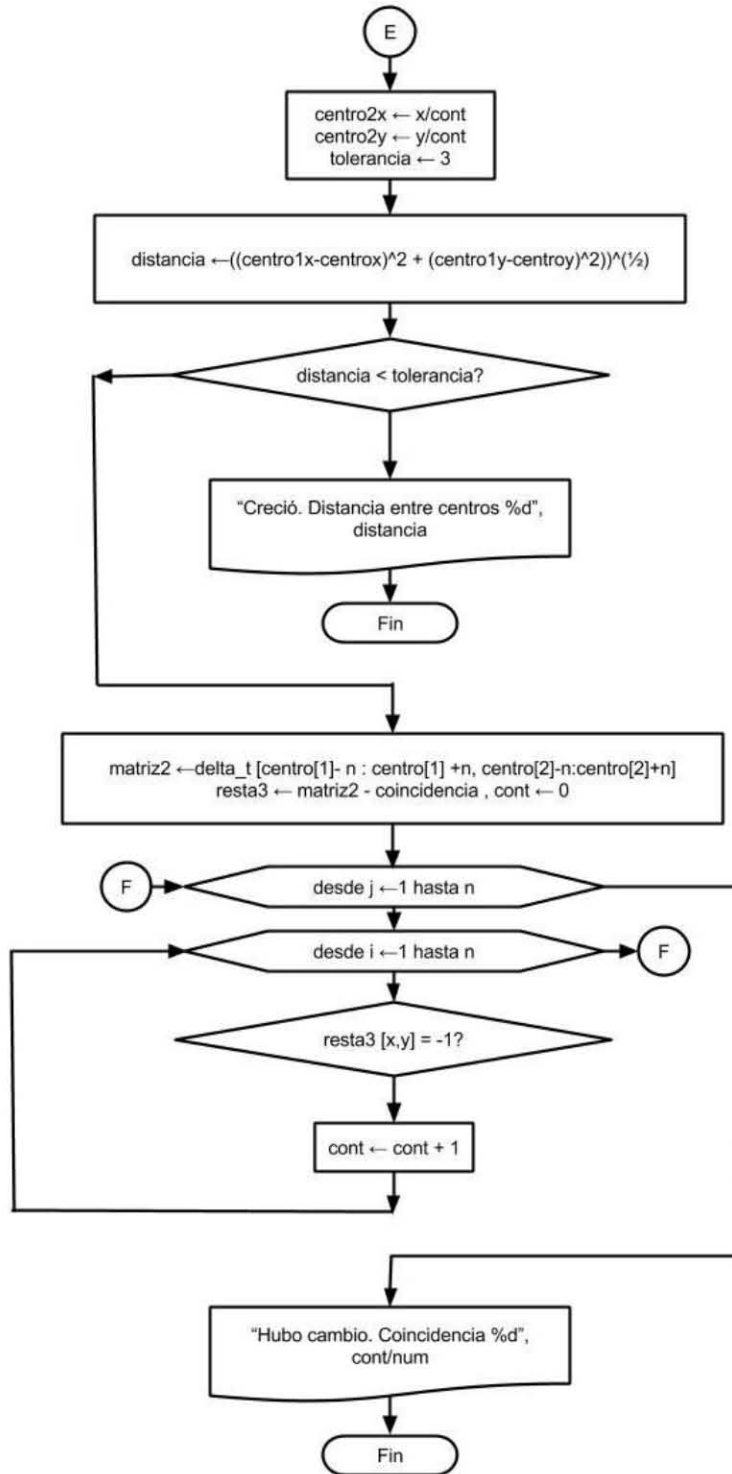
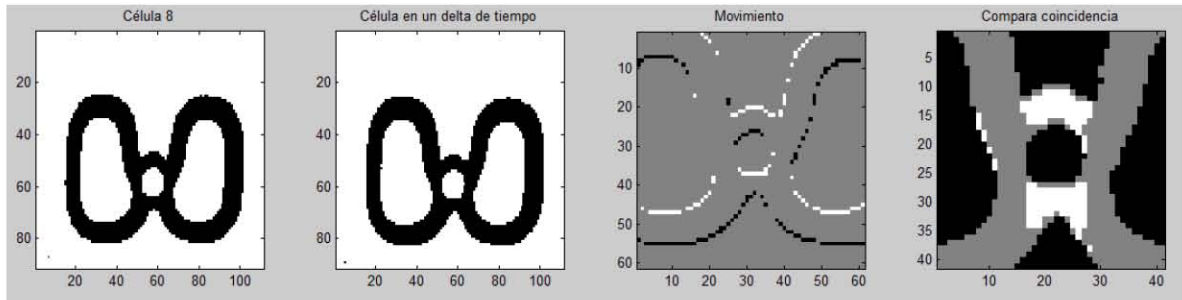


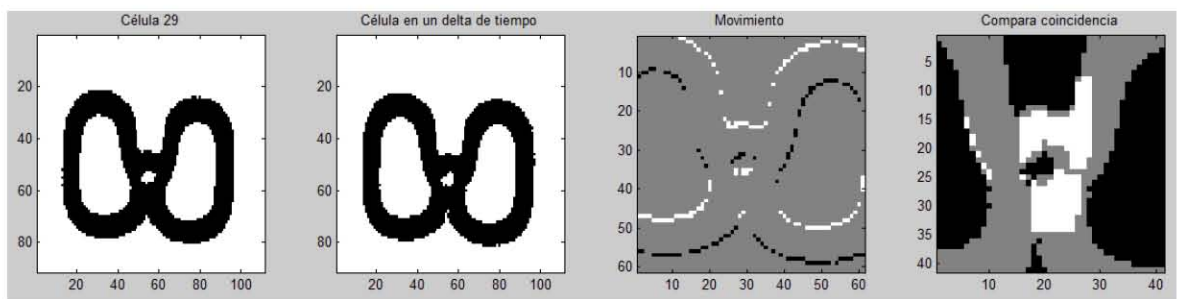
Diagrama 6. Algoritmo de monitoreo de movimiento y crecimiento a través de métodos basados en coincidencia estructural, donde utilizamos la imagen de “coincidencia” y el centro de la densidad de píxeles de cambio obtenidos por el método anterior (método para saber si hay célula en la trampa por algoritmos de sustracción de fondo), y después de hacer una binarización y el recorte del área de interés para el análisis, hacemos una verificación de que al menos el 80% de los píxeles correspondan a la imagen del lapso anterior, con lo que validamos un cambio, dando como resultado final un conteo de píxeles de coincidencia para que en un algoritmo de control externo se determine la acción siguiente del sistema de visión.

Resultados:

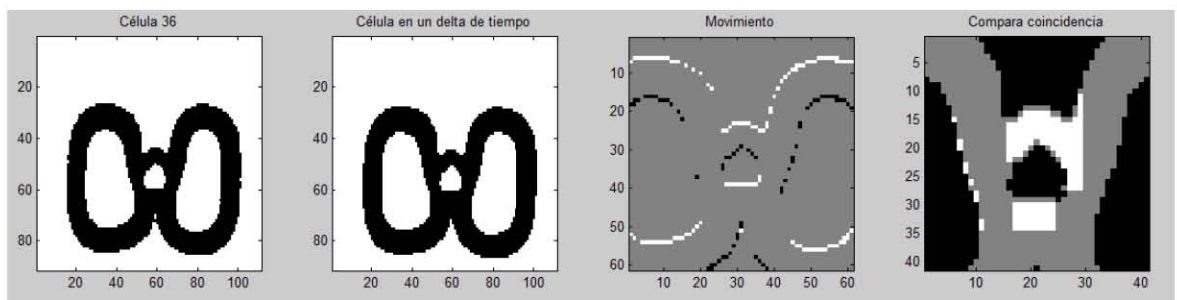
Caso 1. La célula permanece igual. (Trampas 8, 29, 36, 77)



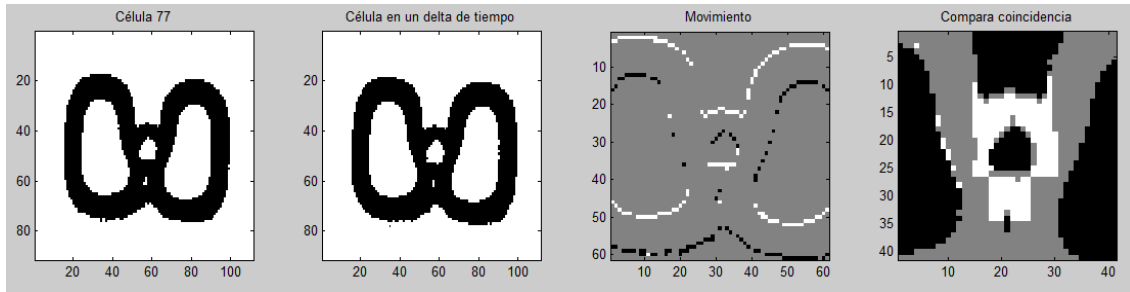
No se movió, coincidencia: 8.740000e+01



No se movió, coincidencia: 91



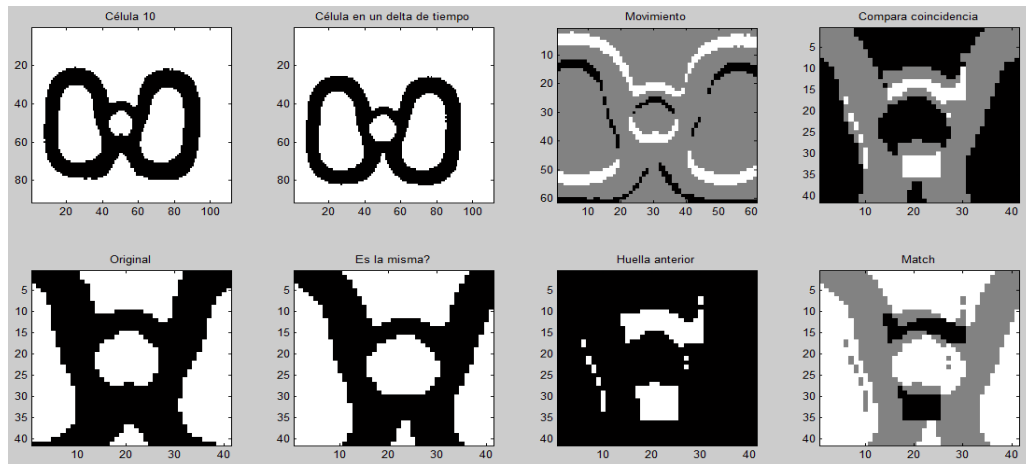
No se movió, coincidencia: 8.720000e+01



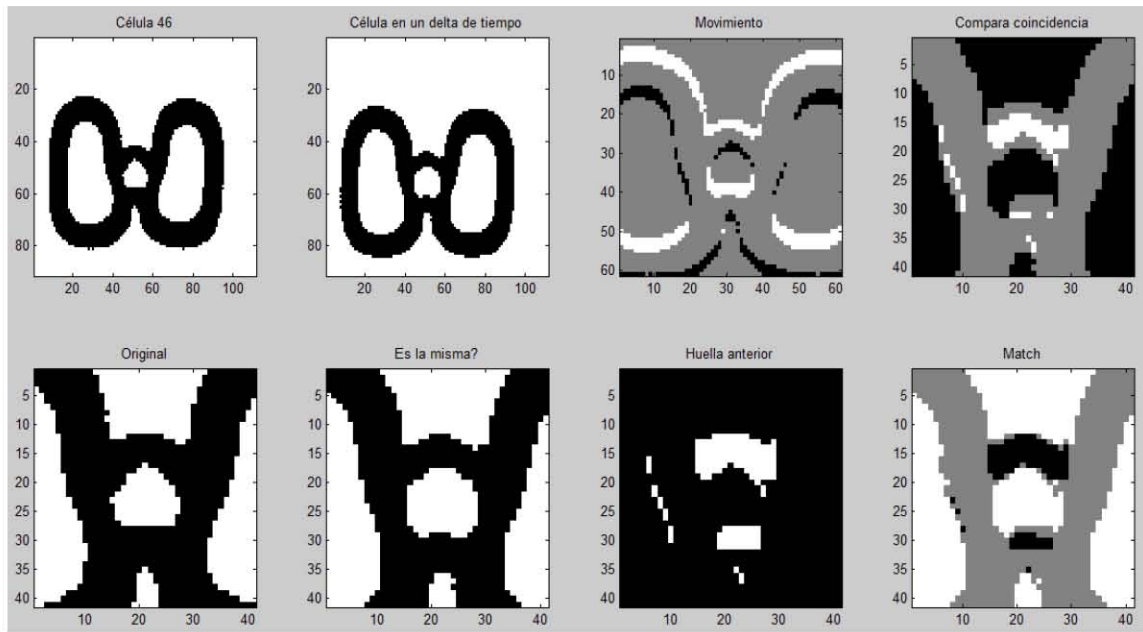
No se movió, coincidencia: 8.920000e+01

Caso 2. La célula crece (Trampas 10, 46, 55, 88)

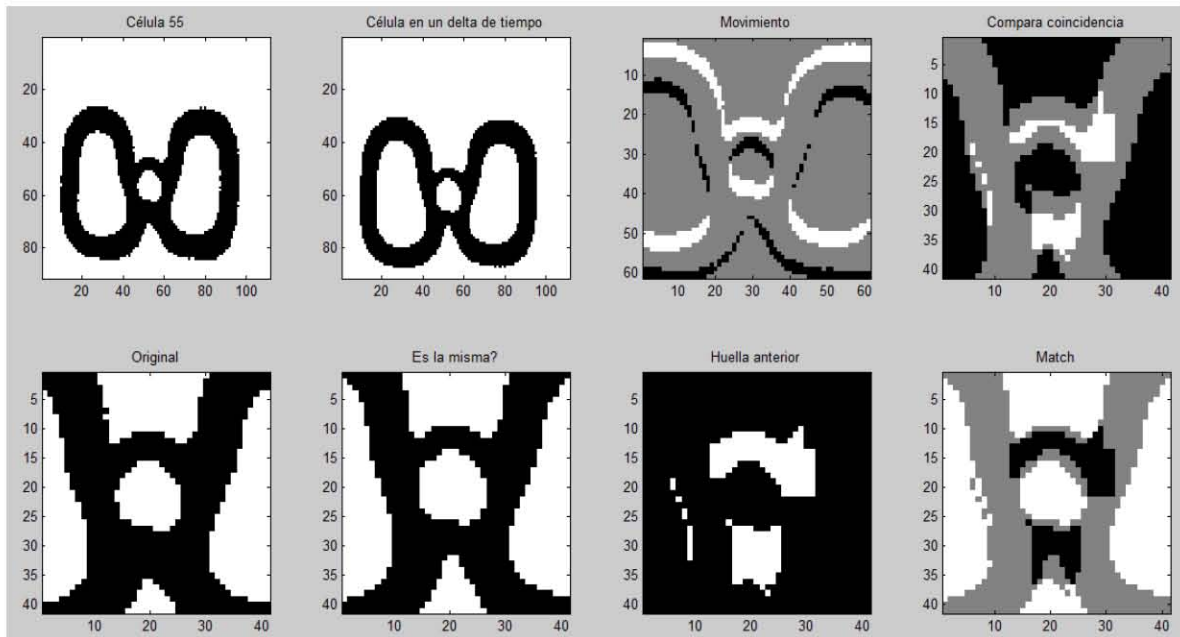
Debido a la escasas de casos en los que varias células crecen significativamente de una imagen a otra, y para hacer un estándar de población de estudio (es decir, analizar varias trampas que estén sujetas a las mismas condiciones), la prueba de este caso se hizo con el cambio entre las tomas 2 y 10 del time lapse, por lo que debemos considerar que estamos tomando con ello una tasa de muestreo 9 veces menor que la requerida, y que afectan los cambios de iluminación no graduales y los cambios de posición de los objetos estáticos con respecto a la cámara, por lo que esto puede servir como primera prueba de robustez, esperando que bajo las condiciones de operación requeridas el algoritmo obtenga mejores resultados que los mostrados.



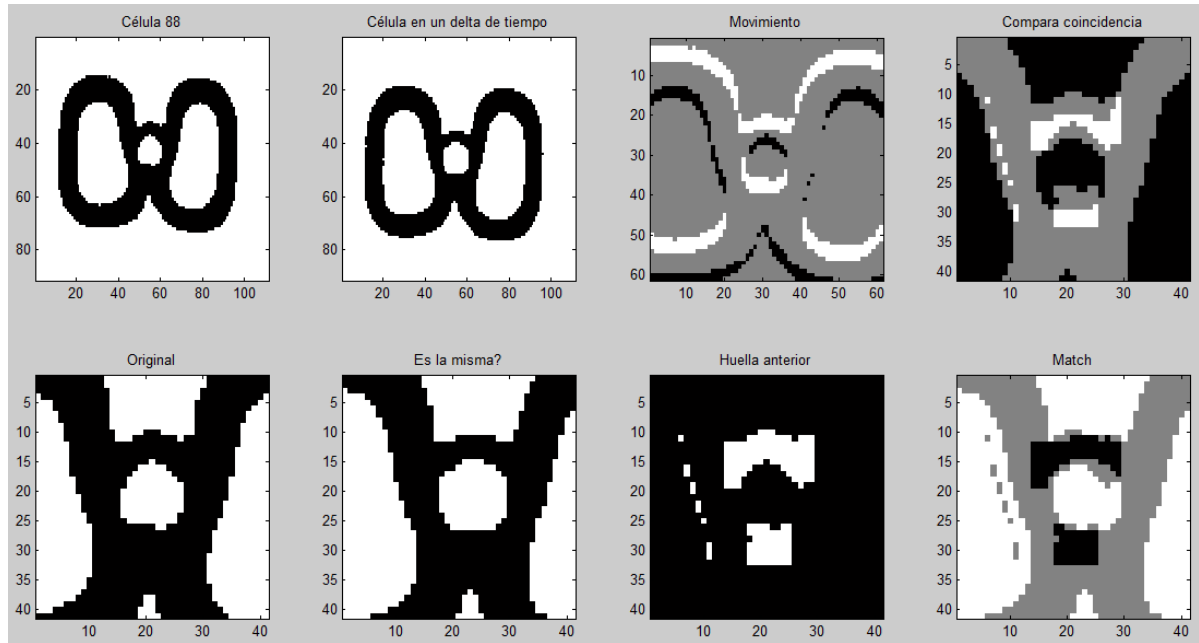
Creció, distancia entre centros: 3.162278e+00>>



Creció, distancia entre centros: 2.236068e+00>>



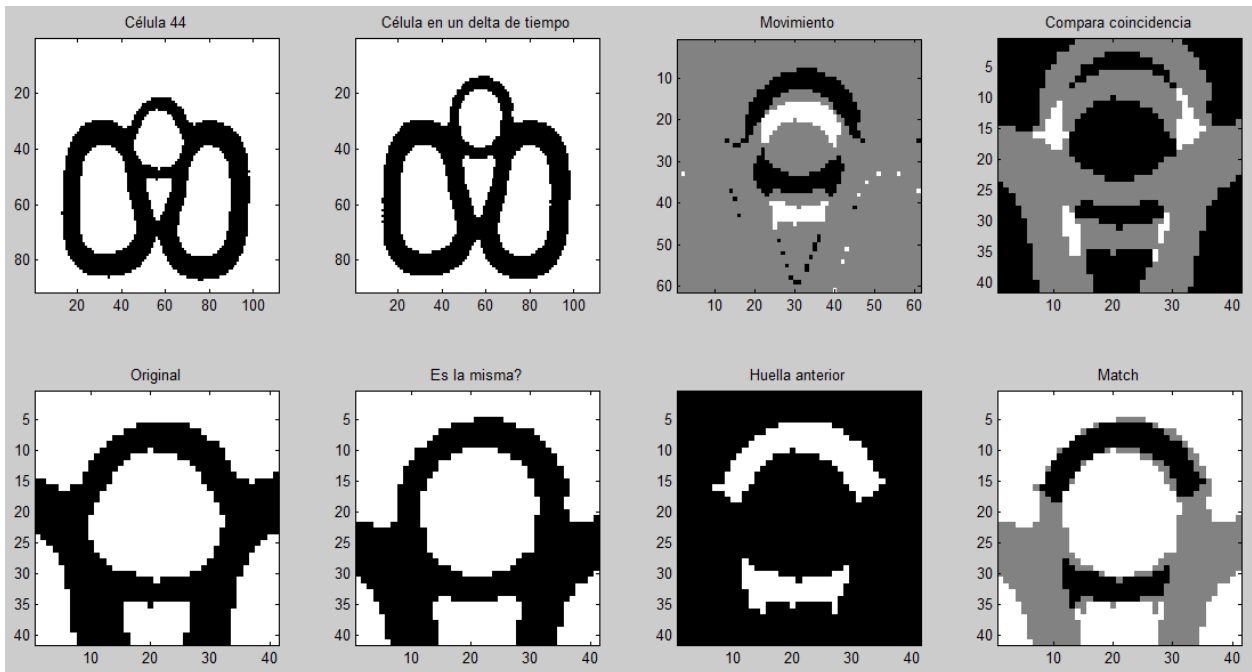
Hubo cambio, coincidencia: 5.910000e+01 Error.>>



Creció, distancia entre centros: 4.242641e+00>>

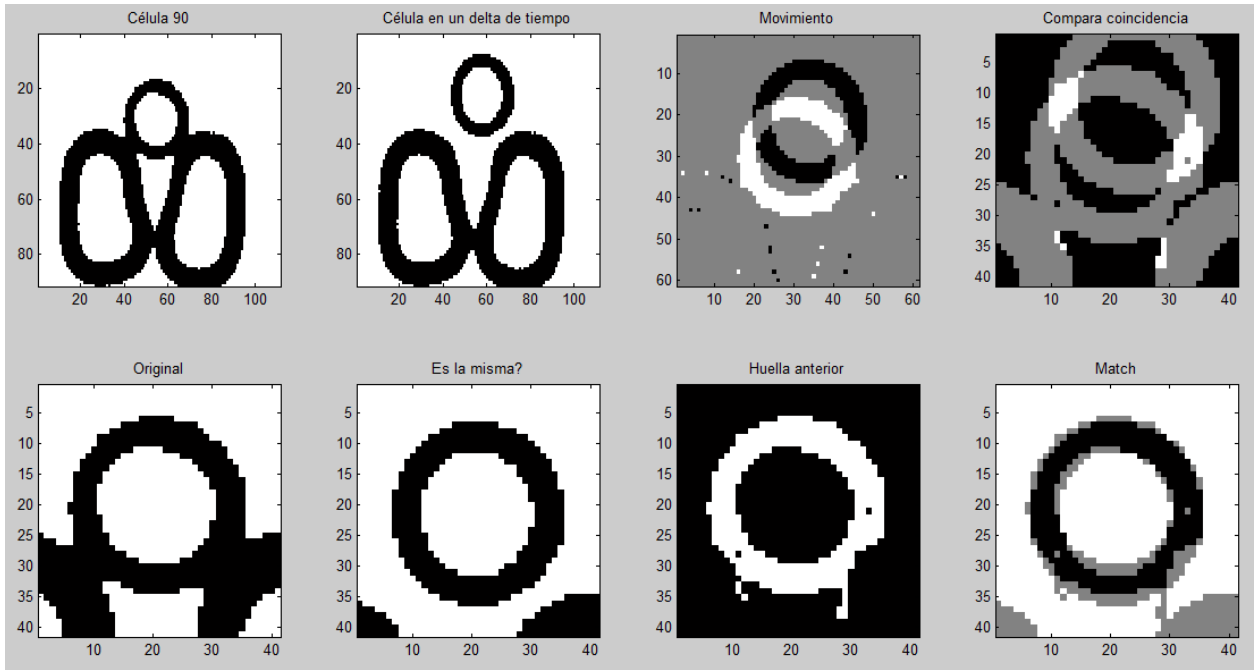
Caso 3. La célula se mueve (Trampas 90, 118, 120)

Trampa 44, tomas 47 y 48.



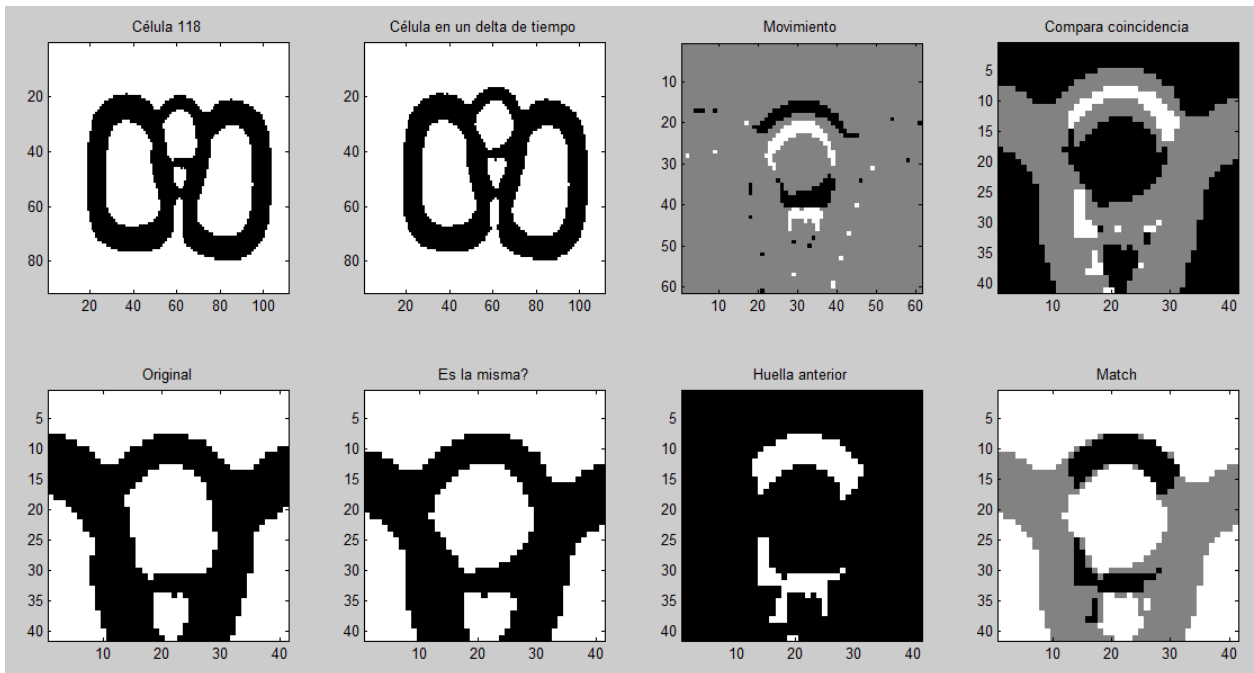
Se movió, coincidencia en match: 8.765432e-01

Trampa 90 tomas 39 y 40



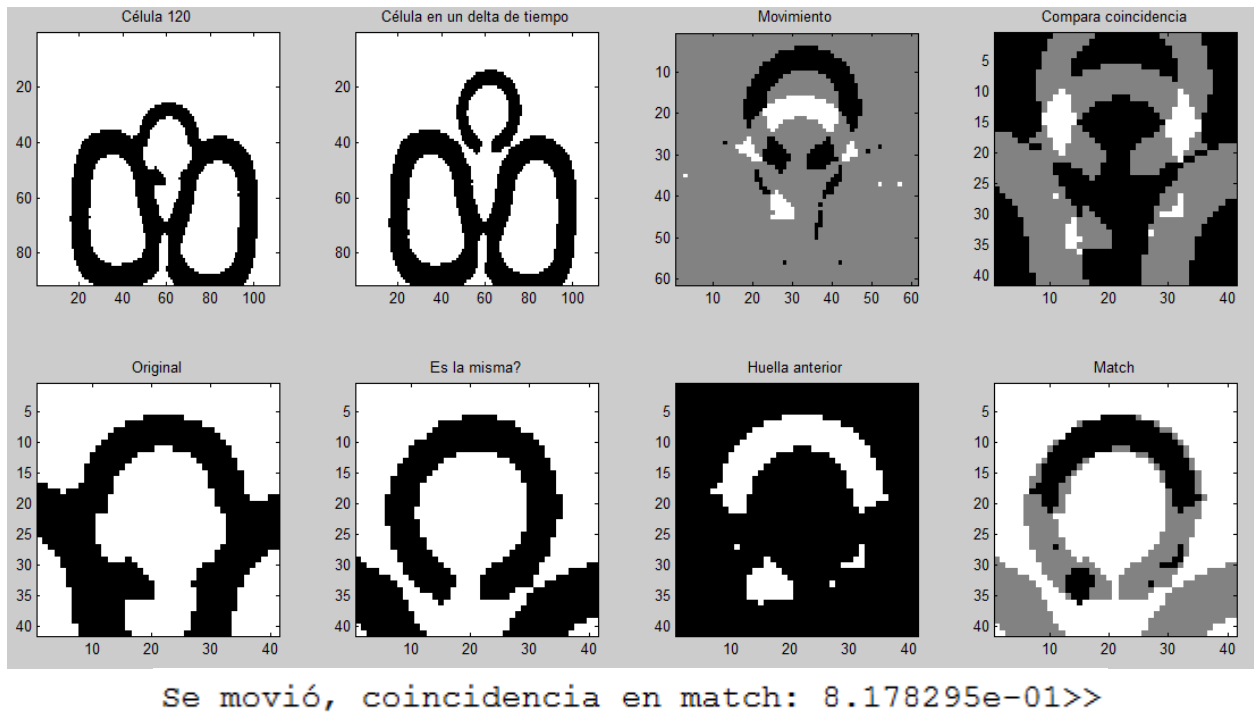
Se movió, coincidencia en match: 8.115183e-01>>

Trampa 118, tomas 55 y 56.



Se movió, coincidencia en match: 8.720930e-01>>

Trampa 120, tomas 42 y 43.



Análisis de Resultados:

Criterio1. Detección de cambio y tipo de cambio

En esta evaluación se consideran dos TP por caso, uno por la detección correcta de cambio y otro por la detección del tipo de cambio, FP y FN se considera sólo en caso de que haya error en la determinación del tipo de cambio, por ejemplo si crece, pero detecta movimiento, contará como un FP para movimiento y FN para crecimiento.

TP=15, TN=4, FP=0, FN=1

$$PCC = \frac{TP + TN}{TP + FP + TN + FN} = \frac{15 + 4}{15 + 4 + 0 + 1}$$

$$PCC = 0.95 = 95\%$$

Conclusiones para la solución del algoritmo de monitoreo por coincidencia estructural:

Observamos que en esta propuesta el error es muy bajo, presentándose un único error en las pruebas de detección de cambio y crecimiento de la célula 55, donde marca error pues su centro no alcanza a estar dentro de la tolerancia (5 píxeles con respecto al centro calculado de la célula anterior) para determinar que la célula ha crecido, lo cual se puede adjudicar al ruido que genera la sombra de la trampa debido en parte al cambio de iluminación gradual entre las tomas que no fueron consideradas (de la toma 3 a la 9), especialmente entre la toma 4 y 5, como se puede observar en el time lapse del documento Anexo.1. Aun así la distancia entre centros es de 5.099 px, por lo que se puede ver que ampliando un poco la tolerancia (0.1 px) se solucionaría este caso, que en realidad no es necesario pues la tasa de muestreo tomada está muy por encima de la real y no se esperan cambios tan bruscos como los que se observan en los casos probados.





 <code>centro1x</code>	61
 <code>centro1y</code>	53
 <code>centrox</code>	56
 <code>centroy</code>	54

Tabla 7 Datos en MATLAB del centro de la célula 55 donde las variables `centro1x` y `centro1y` corresponden al centro de la célula en la toma 2 y `centrox`, `centroy` al centro de la toma 10.

Para la solución de esta parte del algoritmo se exploró también la posibilidad de resolver con el método de *Horn & Schunck* el cual estima el flujo óptico en una imagen pero, debido a que la propuesta actual satisface las necesidades del sistema con un error muy bajo y con una demanda de procesamiento menor que lo que implicaría la implementación de un cálculo de flujo óptico, se optó por seguir con métodos basados en diferencias que, como ya hemos platicado, resultan atractivos para la implementación en el FPGA.

IV.3.e. Determinación y clasificación de reproducción de la célula (propuestas y comparación entre algoritmos)

El planteamiento de esta última tarea representó el reto de diseño más significativo de todo el sistema, debido a que determinar el número de células hijas es el fin último del sistema de visión, por lo que se tuvo la necesidad de tener un conocimiento exhaustivo del comportamiento típico del experimento biológico para llegar a una solución que lograra satisfacer el objetivo.

Como primera posible solución se pensó en detectar a las células hijas con un segundo algoritmo de detección estadística de figuras circulares alrededor de la célula madre, con ello se podrían conocer las posibles células hijas que estén en un rango de distancia del centro de la madre, lo cual funciona en la mayoría de los casos cuando la célula hija tiene una forma circular como las imágenes siguientes:

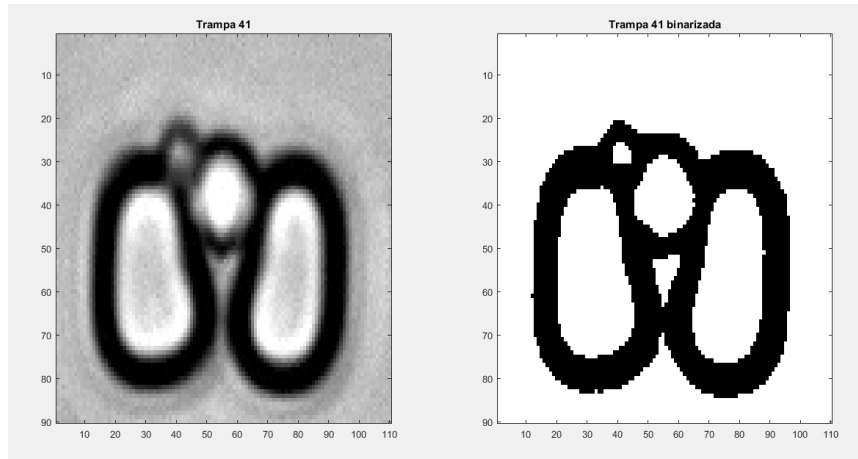


Imagen 29 Caso típico con el que se podría realizar el reconocimiento de célula hija con figuras circulares.

Sin embargo, debido a la naturaleza del experimento se pueden dar casos en los que la célula hija crece de forma alargada o irregular, con lo cual la detección o monitoreo de formas circulares fallaría, y también al haber un flujo constante en el dispositivo microfluídico, puede darse el caso de que una célula ajena o una burbuja sean arrastradas hacia la trampa en análisis y estos casos deben ser considerados para que el algoritmo sea capaz de reconocer si los cambios en la imagen fueron generados por la reproducción de la célula o son agentes que no deben ser considerados como célula hija.

Por estas razones se hizo un análisis de las características de las imágenes de la reproducción de la levadura y se observó que al empezar a reproducirse se produce un ropimiento del borde de la célula madre, mismo que se cierra cuando la célula hija se desprende. Tomando en cuenta esta característica se puede hacer el análisis determinando esta abertura con las siguientes ventajas:

1. Se asegura que la célula se está desprendiendo de la madre, descartando basura o ruido de la imagen.
2. Se descartan los cambios que se podrían producir en la forma o estructura de la célula hija debidas a un cambio de forma o de la reproducción de la misma.
3. Al hacer el análisis de la abertura se hace más fácil reconocer que se trata de la misma célula hija en dos imágenes distintas en caso de que esta se mueva después de desprenderse.

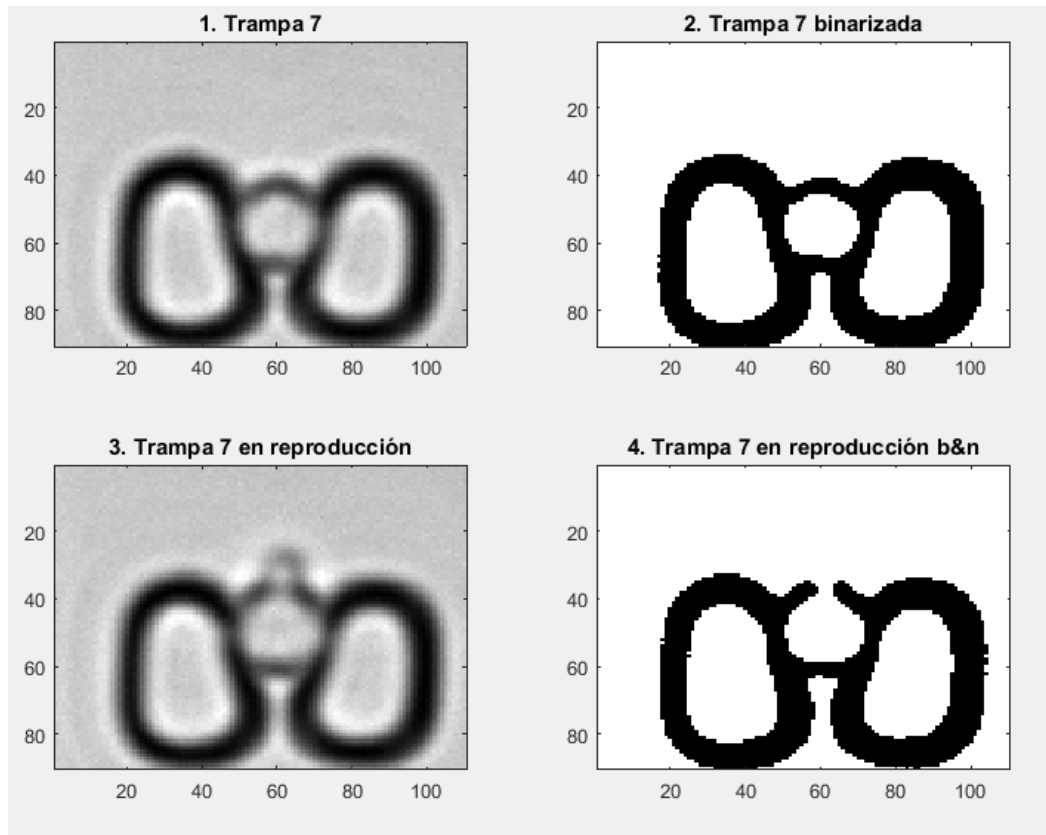


Imagen 30 Se muestra el rompimiento de borde en la imagen binaria (4.) debido a la reproducción de la célula madre.

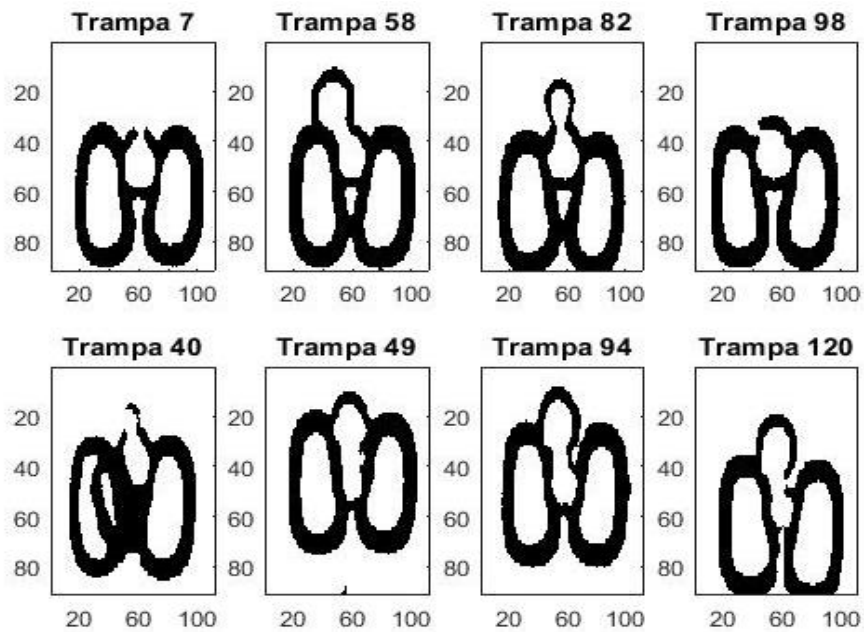


Imagen 31 Se observa el rompimiento de borde en distintos casos de reproducción.

Una vez que definimos que esta característica nos ayuda a determinar la reproducción, se resumió la tarea a monitorear dicha abertura, para lo cual nos valimos del análisis de cambio de regiones, en el que a partir del centro de la célula en estudio (célula madre) se trazan líneas en distintos ángulos y de ellas se determina si hay un cambio en la región con respecto al centro, es decir, si la línea que empieza en el centro no cruza ningún cambio de luminosidad en su trayectoria, indica que hay una abertura en ese ángulo, y por el contrario si hay un cambio en las coordenadas donde se espera el borde, indicará que en ese ángulo no hay abertura o ruptura de borde.

Para utilizar esta solución que parece sencilla y efectiva, se tiene que hacer un procesamiento previo a la imagen que detallaremos a continuación.

Algoritmo propuesto: Determinación de reproducción por abertura de borde

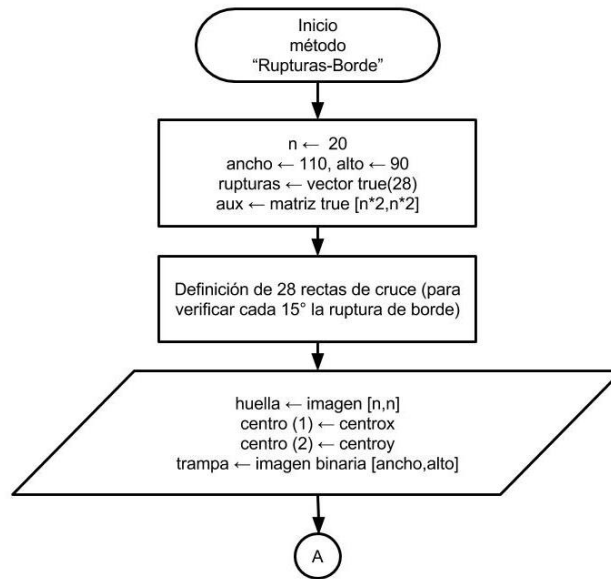
Descripción de la solución:

En esta solución se pretende determinar la cantidad y dirección de las aberturas en el borde de una célula de centro conocido mediante el procesamiento previo de monitoreo de movimiento, se propone hacer una revisión del borde con rectas cada 15° .

Requerimientos:

Para el análisis de la abertura se requiere una serie de rectas que determinarán el cambio de región (que cambie de blanco a negro o viceversa) con respecto al centro (cuyas coordenadas han sido calculadas por el proceso de monitoreo de movimiento previo), al ser un procesamiento directo de píxeles, es posible su realización tomando en cuenta que el ruido en la imagen es escaso. También se considera que el proceso de reproducción en promedio dura entre 4-10 tomas, es decir, el intervalo de toma de fotografía debe ser tal que se alcance a apreciar este proceso en al menos esa cantidad de imágenes para asegurar la detección de ruptura de borde.

Diagrama de flujo:



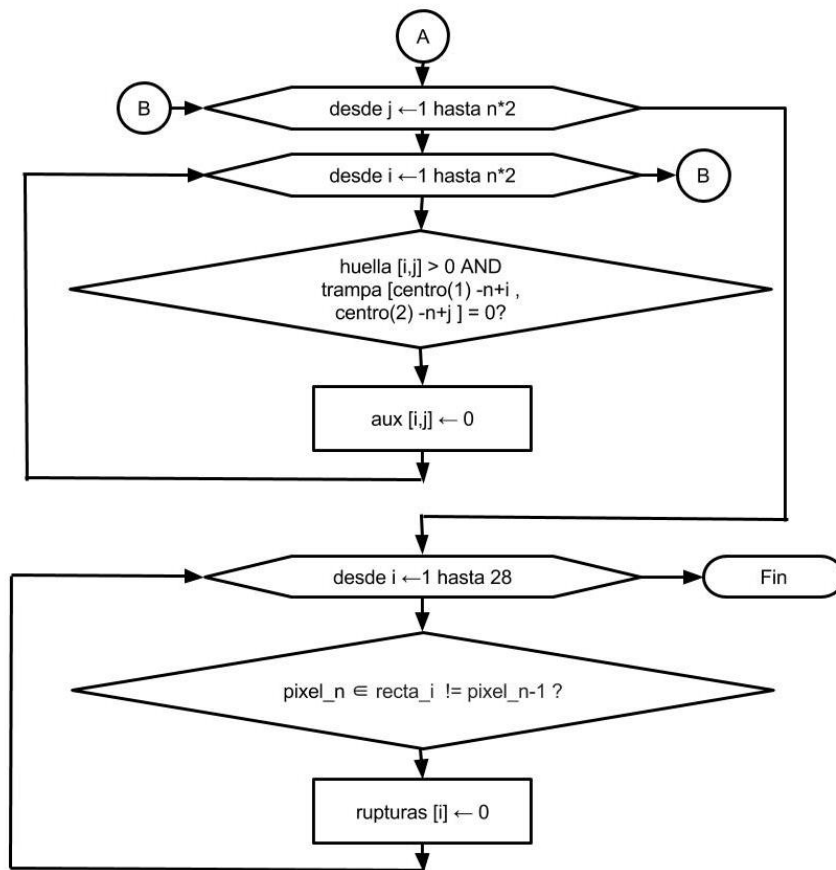


Diagrama 7. En este método a partir de la imagen de coincidencia y el centro obtenidos de los métodos de procesamiento anteriores, se hace coincidir la huella y los píxeles negros de la imagen original, lo cual se vacía en una matriz auxiliar (*aux*) que contendrá únicamente la imagen binaria de la célula madre con el menos ruido posible. A partir de esta imagen se procede a determinar los cruces de regiones, los cuales son guardados en el arreglo *rupturas* de acuerdo al ángulo de la recta en la que se determinó una ruptura de borde.

Conociendo de esta manera los ángulos donde se presenta una abertura, se puede determinar el estado de la reproducción, si se trata de una misma célula hija (dos aberturas en ángulos cercanos), si se reproduce más de una vez al mismo tiempo (caso atípico) y la dirección en la que se encuentra la célula hija, aun cuando en la mayoría de los casos la abertura es detectada antes de que la célula hija pueda observarse de manera clara en la imagen, lo que nos da un margen un poco más amplio para el tiempo de muestreo de toma de fotografía en caso de ser necesario.

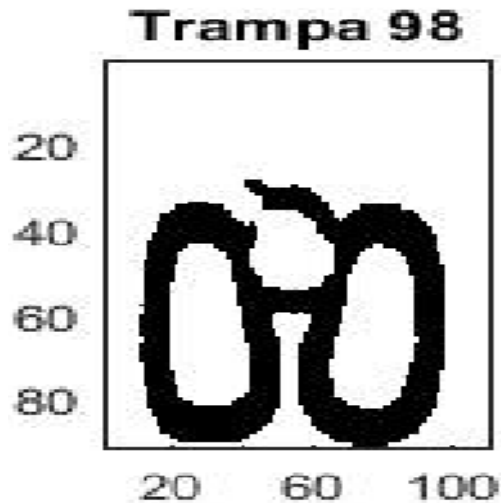


Imagen 32 Aquí se muestra cómo en la binarización se pierden detalles de la célula hija, sin embargo por detección de rompimiento de borde aún en esta etapa temprana se puede determinar y monitorear la reproducción de la célula.

Resultados:

Caso 1. Células en reproducción:

Los siguientes cuadros corresponden al análisis de 5 células en reproducción, cada una con seis imágenes que muestran la trampa completa, la red de rectas separadas cada 15 grados, la imagen auxiliar que aísla la célula del fondo, la detección de la madre como referencia, la matriz de rupturas que es donde vemos el o los ángulos donde se detectó una abertura (se multiplica por 15), y por último vemos, a modo de verificación, una recta del centro hacia el ángulo donde se detectó la abertura.

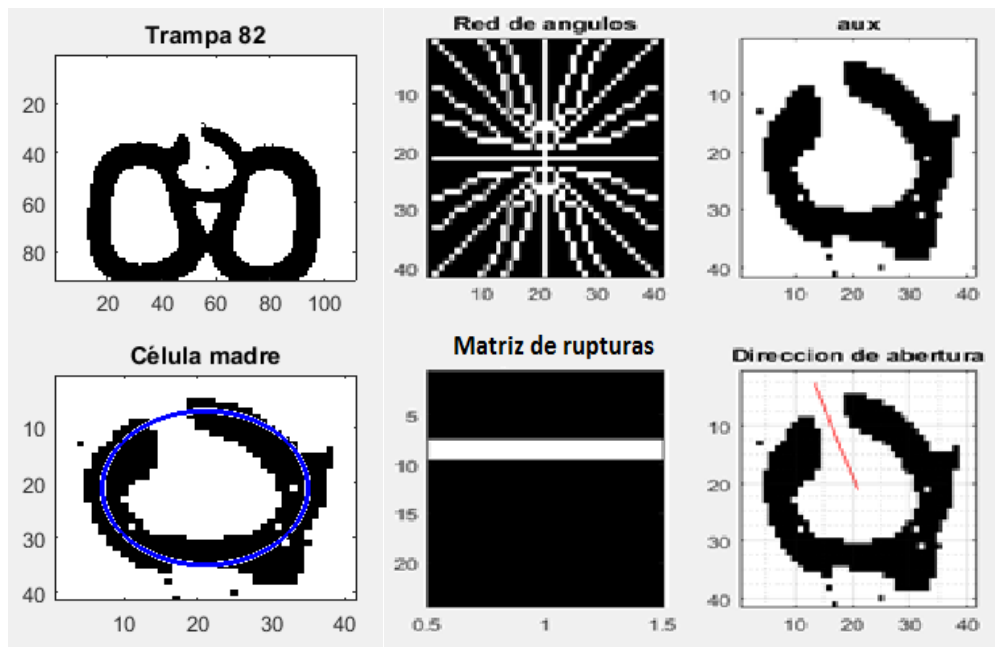


Imagen 33 Análisis de la dirección de abertura en la trampa 82.

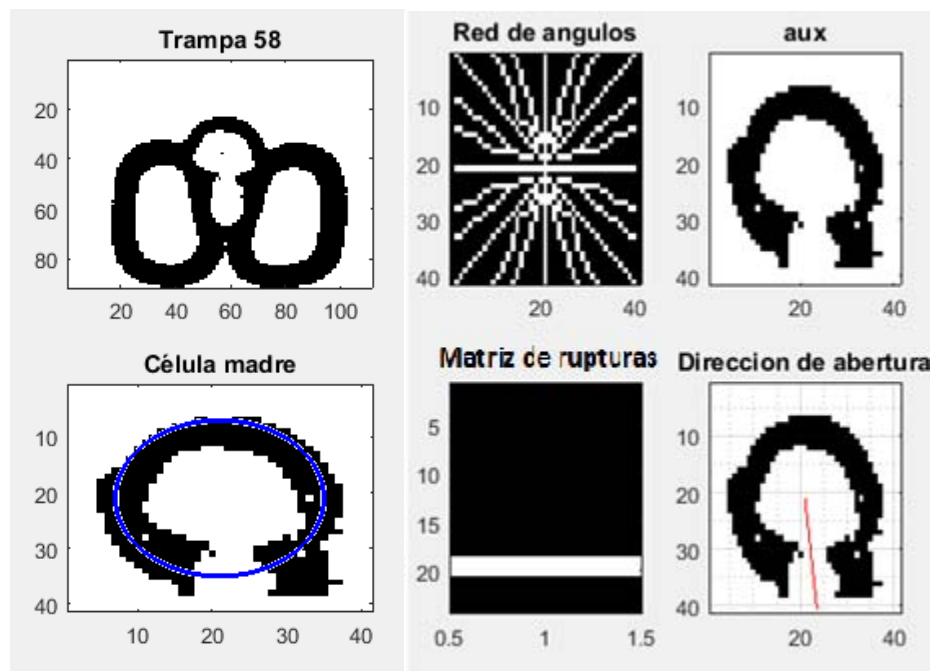


Imagen 34 Análisis de la dirección de abertura en la trampa 58.

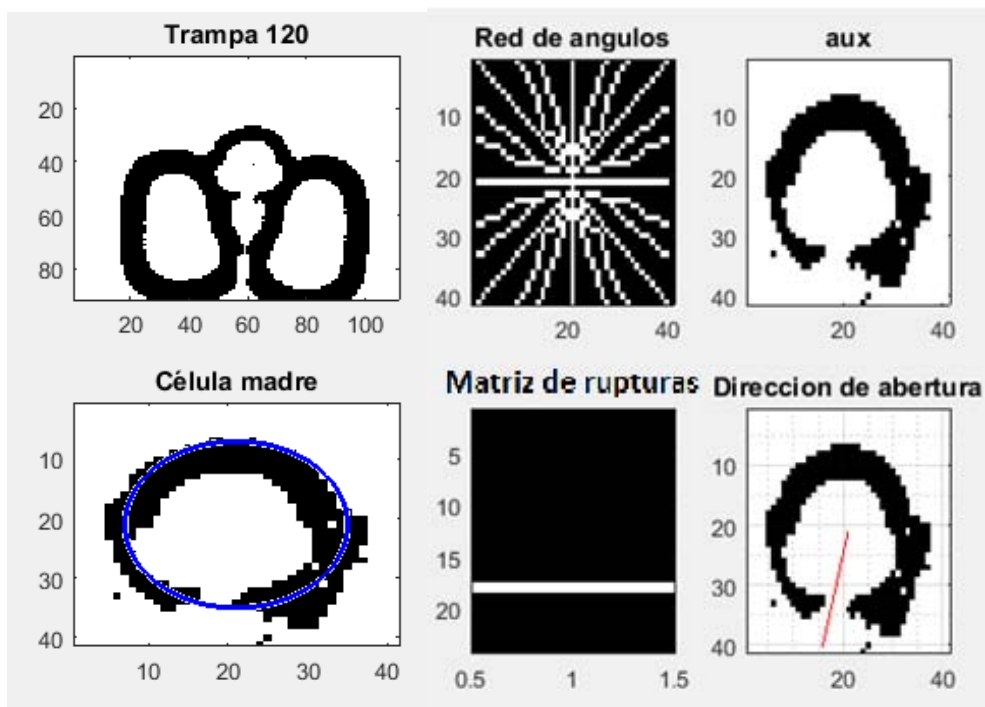


Imagen 35 Análisis de la dirección de abertura en la trampa 120.

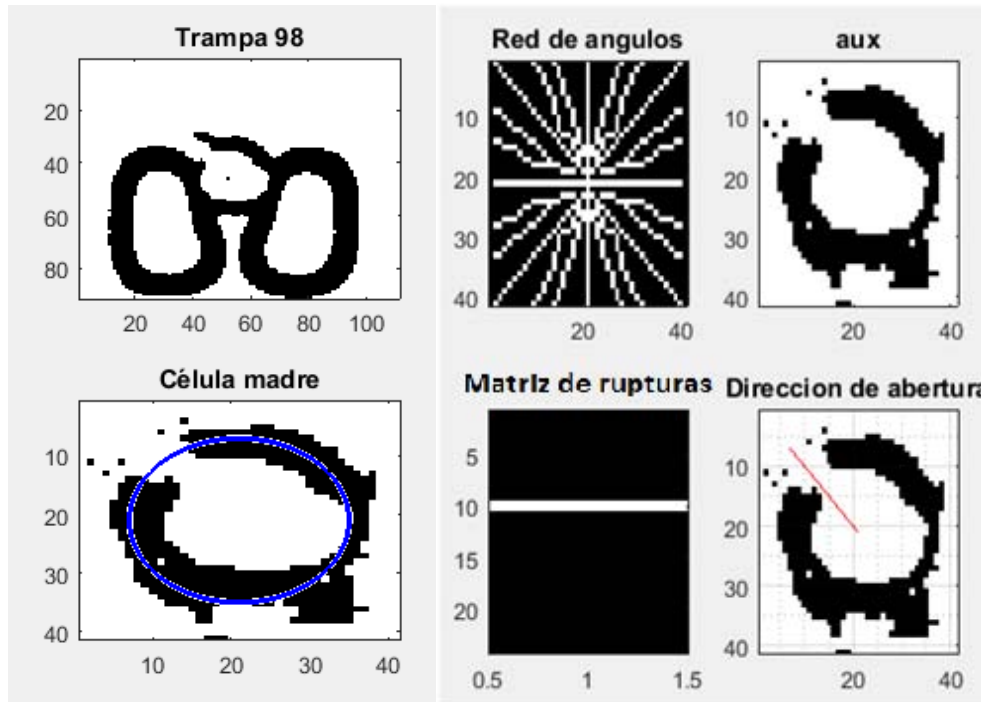


Imagen 36 Análisis de la dirección de abertura en la trampa 98.

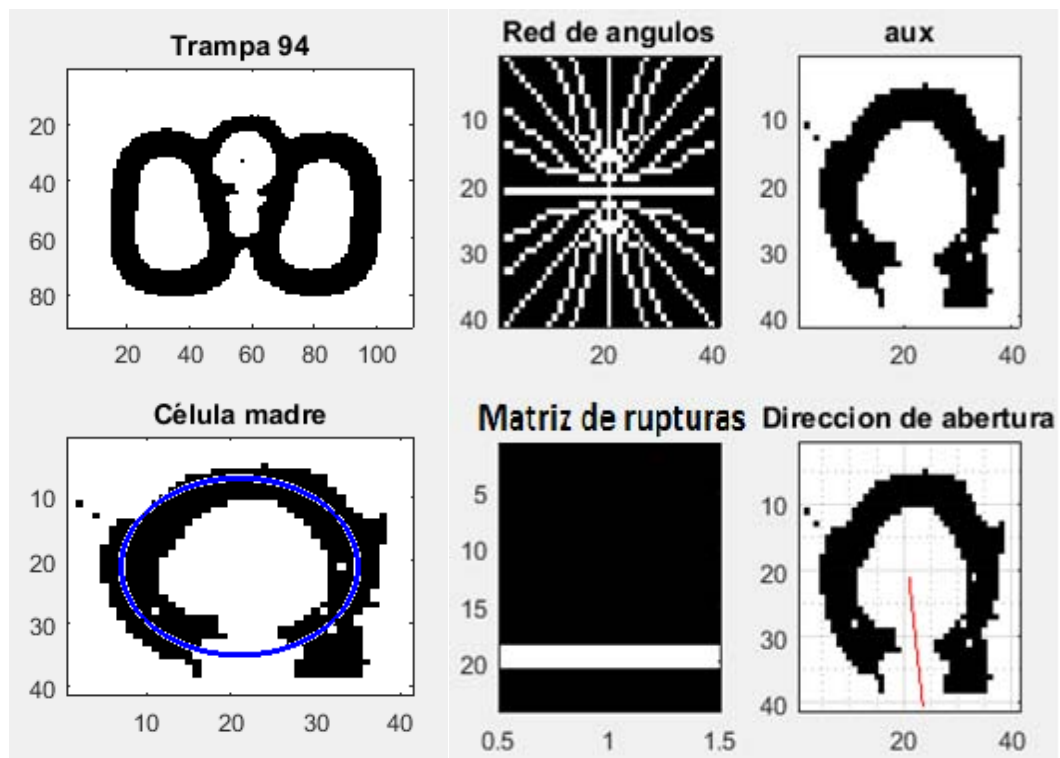


Imagen 37 Análisis de la dirección de abertura en la trampa 94.

Caso 2. Células que no están en reproducción

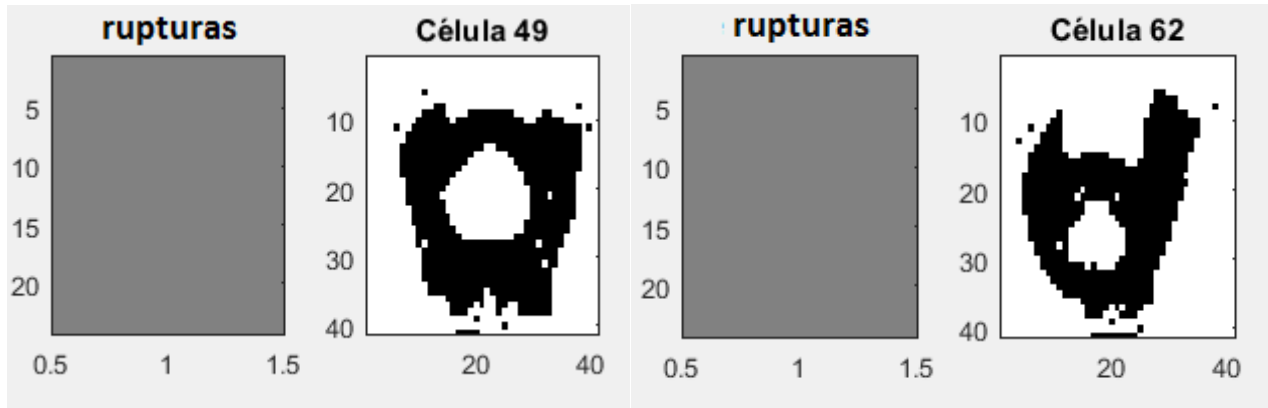


Imagen 38 Vemos que la matriz de rupturas al estar todos sus valores en cero indica que no hay abertura en las trampas 49 y 62.

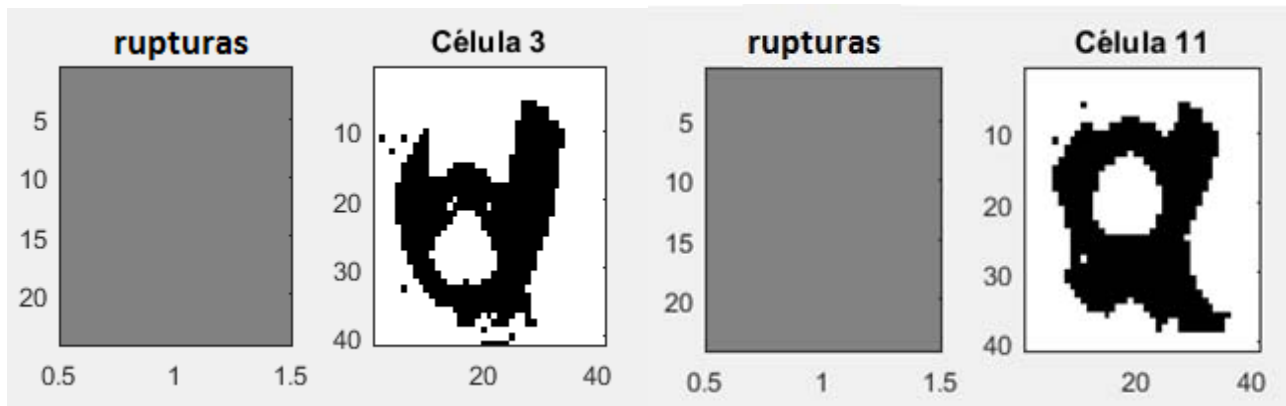


Imagen 39 Vemos que la matriz de rupturas al estar todos sus valores en cero indica que no hay abertura en las trampas 3 y 11.

Análisis de Resultados:

En esta evaluación con el método PCC, se considera TP cuando detecta correctamente el ángulo de la abertura, FP cuando detecta una abertura que no existe, TN cuando no hay abertura y deja la matriz de rupturas sin cambios y FN se considera en caso de que determine que no existe abertura y la célula esté en reproducción.

$$TP=5, TN=4, FP=0, FN=0$$

$$PCC = \frac{TP + TN}{TP + FP + TN + FN} = \frac{5 + 4}{5 + 4 + 0 + 0}$$

$$PCC = 1 = 100\%$$

Conclusiones para la solución del algoritmo para la determinación de reproducción a través del análisis de ruptura de borde:

Observamos que en esta propuesta se logra el objetivo de detección de rompimiento de borde, y esto funciona debido a dos condiciones importantes: primero que se tienen imágenes con poco ruido, lo que permite hacer un análisis en el dominio espacial sin filtros previos y en segundo lugar resulta crucial el correcto cálculo del centro de la célula madre, tarea que recae sobre el algoritmo de monitoreo de movimiento del que este proceso recibe los parámetros de entrada.

V. INTEGRACIÓN DEL SISTEMA E IMPLEMENTACIÓN DE LOS MÓDULOS EN FPGA

V.1. Propuesta de diseño de la arquitectura para el Algoritmo de Visión Artificial

Para que el Sistema de Visión Artificial pudiera ser descrito en hardware se tomó en cuenta que el módulo generado trabajará en conjunto con diferentes sistemas embebidos en el mismo FPGA, entre los cuales estará el propio módulo de control del algoritmo, que indicará cuándo debe actuar, así como interpretar y manejar los resultados arrojados. Una característica importante que se consideró en la propuesta de diseño de la arquitectura para el Sistema de Visión, es que en el FPGA pueden ser descritas diferentes herramientas de las que podemos hacer uso, para ello encontramos conveniente el uso de un microprocesador en el que nos apoyaremos para realizar los procesamientos que demanden demasiados recursos para ser descritos en hardware. Para hacer la propuesta del microprocesador a utilizar se hace necesario definir los requerimientos del sistema de visión, lo cual será discutido más adelante en este capítulo, todo con el fin de seguir el principio de diseño de un sistema a la medida en el que tengamos total control de cada componente para tener la posibilidad de aplicar técnicas de tolerancia a fallas en una segunda etapa del proyecto, así como satisfacer los requerimientos de consumo energético.

Al haber trabajado los algoritmos en un esquema de procesamiento por trampa, se tiene la ventaja de que el sistema se puede ajustar al número de trampas necesarias en la carga útil de la misión, y al mismo tiempo esto nos da la facilidad de proponer un sistema de módulos que haga el procesamiento de la imagen de una o varias trampa a la vez. El sistema puede ser alimentado de los datos necesarios según el procesamiento requerido, a lo cual le encontramos las siguientes ventajas:

1. Se pueden describir varios módulos dependiendo de los recursos disponibles de la tarjeta FPGA que se seleccione al final para la misión, esto permitiría analizar las trampas en paralelo con lo que disminuye el tiempo de procesamiento y se genera redundancia en hardware.
2. Se hace posible ajustar el número de módulos activos de acuerdo a los recursos energéticos.
3. La interpretación de los resultados arrojados, la comparación con la base de datos generada y la toma de decisiones serán procesadas en un microprocesador embebido, el cual puede ser diseñado a la medida de acuerdo con los requerimientos determinados por el algoritmo de control del análisis de imagen, o definido por la suma de requerimientos de procesamiento generales con lo que el control del Sistema de Visión Artificial se integraría al flujo de procesamiento general de los diversos sistemas que componen al proyecto.

Explicado lo anterior, el Sistema de Visión se propone como se ilustra en el Diagrama 8:

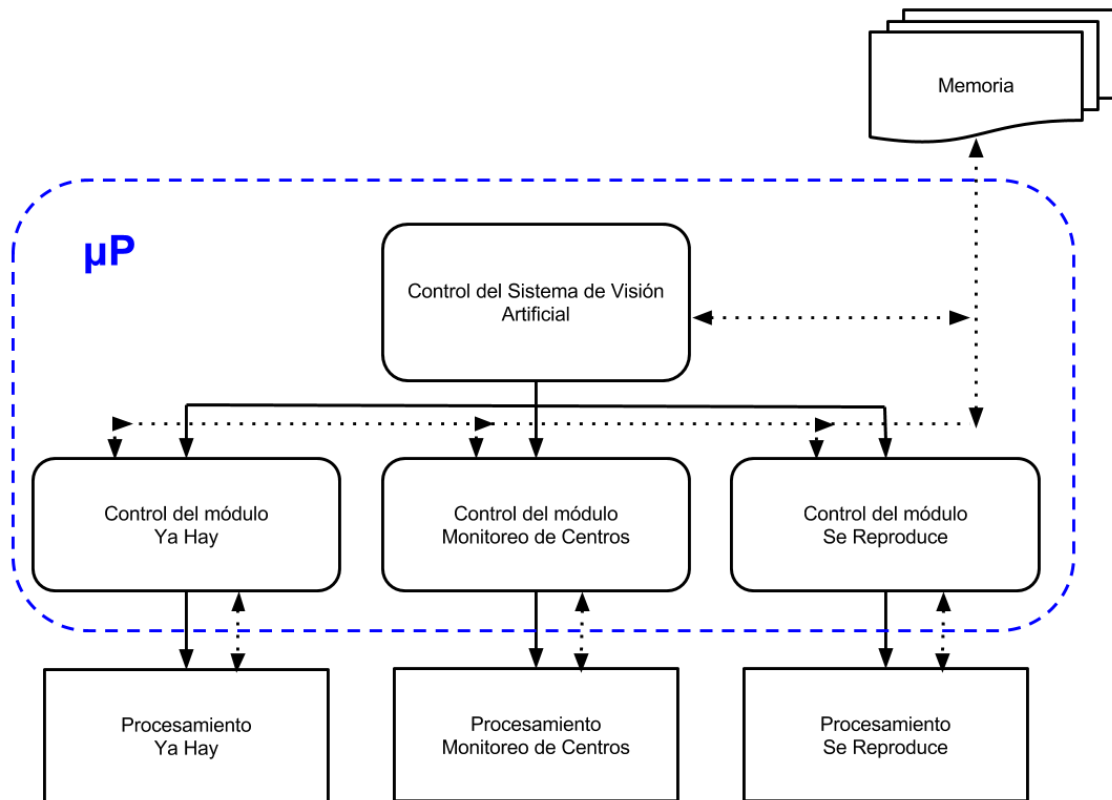


Diagrama 8 Concepto general del Sistema de Visión, donde el sentido de las flechas indica el sentido del control de procesos y el sentido de las flechas punteadas indica el sentido de flujo de datos.

En el Diagrama 8 observamos los tres módulos de procesamiento de imagen, donde de acuerdo a la jerarquía de los procesos (de arriba hacia abajo, marcados por flechas sólidas), primero dentro de un microprocesador se programa el algoritmo de control o toma de decisiones general, el cual manda ejecutar las rutinas de control de los módulos de procesamiento de imagen, todo lo cual tiene acceso a la lectura y escritura de una memoria externa. Las rutinas de control secundario son las que interactúan con los módulos de procesamiento, que deben ser descritos en hardware.

A continuación se presenta la propuesta de las arquitecturas descritas en el FPGA para los tres módulos de procesamiento de imagen del sistema:

Proceso “¿Ya hay célula?”:

La propuesta de arquitectura es tener un módulo síncrono con tres puertos de entrada, dos para leer los pixeles de las imágenes a comparar y como auxiliares en otros procesos (*px1_in*, *px2_in*), así como el puerto mediante el que se le indicará al módulo la rutina que debe hacer (*bitEscritura*), en cuanto a las salidas se tienen dos canales para el envío de diferentes datos (*ch1_out*, *ch2_out*), así como un puerto para conocer el estado general de procesamiento (*status_out*), además de los puertos de reloj (*clk*) y *reset* como se muestra en el Diagrama 9.

El significado de las entradas y salidas del módulo que realiza el proceso “¿Ya hay célula?” se interpretará de acuerdo a la Tabla 8.

“Ya hay” Métodos	Entradas		Salidas				Interpretación status
	<i>px1_in</i>	<i>px2_in</i>	<i>bitEscritura</i>	<i>ch1_out</i>	<i>ch2_out</i>	<i>status_out</i>	
Reset variables	~	~	0	0	0	2	Terminó
Lectura y binarización	<i>px_trampa</i>	<i>px_deltat</i>	1	0/x	0/y	0/1	Procesando/ Terminó
Aproximación de índices	<i>pxX_deltat</i>	<i>pxY_deltat</i>	2	0/x	0/y	0/1	Procesando/ Terminó
Determinación de centros	<i>pxX_deltat</i>	<i>pxY_deltat</i>	3	0/centrox	0/centroy	0/1	Procesando/ Terminó
Pasar centros anteriores	<i>centrox</i>	<i>centroy</i>	4	0	0	1	Ok
Procesamiento con Huella	<i>px_Huella</i>	~	5	0	0	0/1/2	Procesando/ Si Hay/No Hay
Envío de matriz Coincidencia	~	~	6	<i>ind_x</i>	<i>ind_y</i>	0/1	Bit de matriz de coincidencia
Procesamiento con ruido	~	~	7	0/centrox	0/centroy	0/1	Procesando/ Terminó

Tabla 8. Relación de entradas y salidas de la arquitectura para el módulo “Ya Hay”, donde el símbolo ~ significa que el valor presente a la entrada del puerto es indiferente pues no se toma en el procesamiento en ese método.

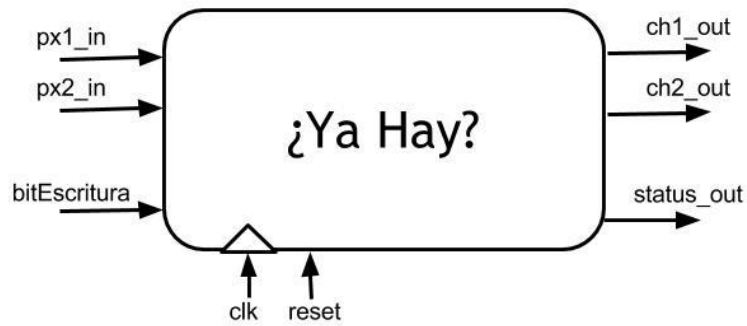


Diagrama 9. Arquitectura del módulo de procesamiento de imagen “Ya Hay”, donde el sentido de las flechas indica si es un puerto de entrada (dirección hacia el módulo) o salida (dirección hacia fuera del módulo).

Más adelante explicaremos el diseño del algoritmo de control de los parámetros de entrada de acuerdo a los diferentes casos que pudieran presentarse a la salida.

Proceso “Monitoreo de centros”:

La propuesta de arquitectura es tener un módulo síncrono con dos puertos de entrada (*px1_in*, *px2_in*) para leer los pixeles de las imágenes a comparar y también como auxiliares, así como el puerto por el que se le indica al módulo la rutina que debe ejecutar (*bitEscritura*). Como salidas se tienen dos canales, uno para el envío de diferentes datos (*ch1_out*) y el segundo para conocer el estado general de procesamiento (*status_out*), además de los puertos de reloj (*clk*) y *reset* como se muestra en el Diagrama 10.

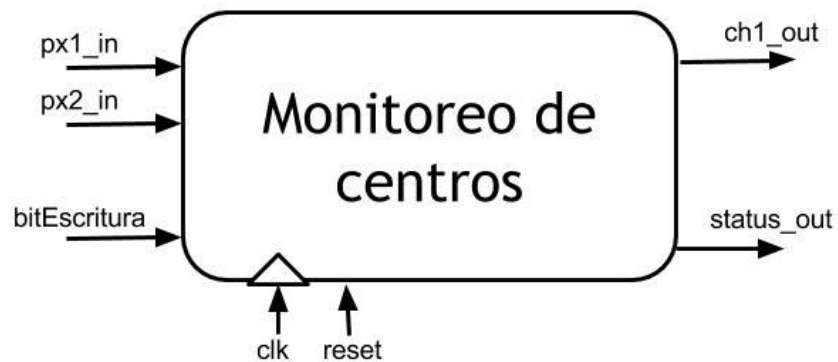


Diagrama 10. Arquitectura del módulo de procesamiento de imagen “Monitoreo de Centros”, donde el sentido de las flechas indica si es un puerto de entrada (dirección hacia el módulo) o salida (dirección hacia fuera del módulo).

El significado de las entradas y salidas del módulo para monitoreo de centros se interpretará de acuerdo a la Tabla 9.

“Monitoreo de Centros” Métodos	Entradas			Salidas		Interpretación de salidas
	px1_in	px2_in	bitEscriura	chl_out	status_out	
Reset de variables	~	~	0	0	0	Terminó
Leer matriz de coincidencia y recorte	px_coincid	px_recorte	1	0/(1/2/3)	0/1	Procesando/ (Cambio Pequeño/ Centros Similares/ Cambio Grande)
Leemos los centros	~	~	2	cenx/ceny	0/1	Bit de cambio entre lectura de cenx y ceny
Error de datos de entrada	~	~	otro	0	1	~

Tabla 9. Relación de entradas y salidas de la arquitectura para el módulo “Monitoreo de Centros”, donde el símbolo ~ significa que el valor presente a la entrada del puerto es indiferente pues no se toma para el procesamiento en esa rutina.

Proceso “¿Se Reproduce?”:

La propuesta de arquitectura es tener otro módulo síncrono con tres puertos de entrada, dos para leer en paralelo los pixeles de las imágenes a analizar (*px1_in*, *px2_in*), así como el puerto “*indice*” por el que, al igual que en los casos anteriores con el puerto “*bitEscriura*”, se le indica al módulo la rutina que debe ejecutar. Como salidas se tienen dos canales, uno para el envío de diferentes datos, entre los cuales están los pixeles de la matriz de rupturas (*ruptura_out*) y el segundo para conocer el estado general de procesamiento (*status_out*), además de los puertos de reloj (*clk*) y *reset*.

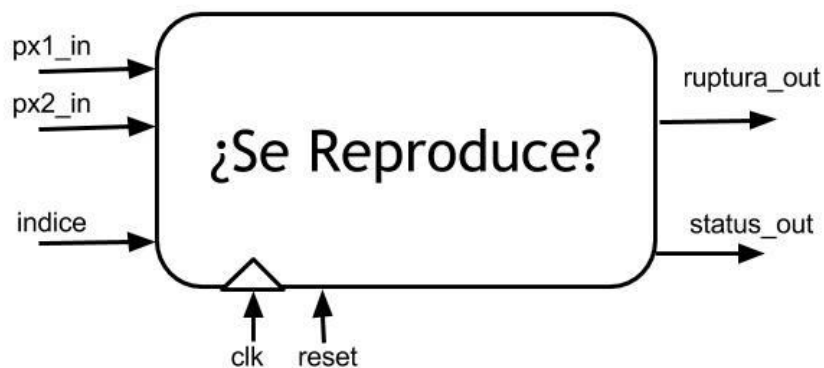


Diagrama 11. Arquitectura del módulo de procesamiento de imagen “Se Reproduce”, donde el sentido de las flechas indica si es un puerto de entrada (dirección hacia el módulo) o salida (dirección hacia fuera del módulo).

A continuación presentamos la Tabla 10 con la que deberán interpretarse las entradas y salidas:

¿Se Reproduce? Métodos	Entradas			Salidas		Significado status
	px1_in	px2_in	indice	paridad out	status_out	
Reset de variables	~	~	0	0	0	
Huella sobre imagen binaria	px_dtByN	px_huella	1	0	0/1	Procesando/ Terminó

Determinar ruptura	~	~	2	0	0/1	Procesando/ Terminó
Enviar matriz de rupturas	~	~	3	(0/1)/~	0/1	Procesando/ Terminó

Tabla 10. Relación de entradas y salidas de la arquitectura para el módulo “Se Reproduce”, donde el símbolo ~ significa que el valor presente a la entrada del puerto es indiferente pues no se toma para el procesamiento en esa rutina.

Donde, al igual que en los casos anteriores, el símbolo ~ significa que el valor que toma el puerto es indiferente.

V.2. Integración del Sistema

V.2.a. Primera integración de los módulos del Sistema de Visión, orientada a facilitar la implementación en hardware reconfigurable

Para hacer la descripción en el FPGA se decidió utilizar la herramienta HDL Coder de MATLAB, por dos razones, la primera es que facilita la creación y relleno de las localidades de memoria utilizadas para los arreglos de matrices que representan las imágenes, sin esta herramienta la declaración manual de las localidades de memoria que ocupan las imágenes hubiera sido compleja (en el menor de los casos sería definir un arreglo de 1600 localidades para las imágenes de recorte).

En segundo lugar gracias a la metodología de diseño con la que se desarrolló el sistema, la cual permitió obtener los algoritmos de procesamiento basados en funciones lógicas, prescindiendo de ciclos y operaciones de difícil manejo en hardware, es factible utilizar la herramienta HDL Coder como un traductor de código manteniendo el control de las características de la arquitectura, es decir, en las funciones basadas en sentencias lógicas se migra de una función condicional a la sentencia equivalente con la sintaxis de VHDL, y el código generado se puede interpretar y manipular para verificar que corresponde a lo diseñado en el código M (lenguaje de MATLAB), en contraste con una función compleja o de alto nivel, donde la herramienta genera su “interpretación” de arquitectura basada en las reglas de diseño definidas por MATLAB, que además de restar control sobre el hardware generado, muchas veces no genera una arquitectura eficiente para la aplicación específica.

Hay algunos desafíos importantes al pasar código de MATLAB a hardware. El código MATLAB trabaja por procedimientos y puede ser sumamente abstracto; puede utilizar datos de punto flotante, no tiene noción de tiempo y la implementación de bucles complejos se puede tornar impráctica, por lo que en resumen podemos decir que la implementación de código MATLAB en hardware a través de la herramienta HDL Coder en un esquema general implica:

- Hacer una conversión del código de punto flotante a punto fijo para definir tamaños de palabra adecuados para la generación eficiente de hardware.
- Introducción del concepto de tiempo mediante el uso de ciclos de reloj para programar las operaciones en hardware.
- Creación de arquitecturas de recursos compartidos para implementar operadores costosos como multiplicadores y cuerpos de ciclos *for*.

Hasta esta parte del proyecto el algoritmo de visión se tenía estructurado de la siguiente manera:

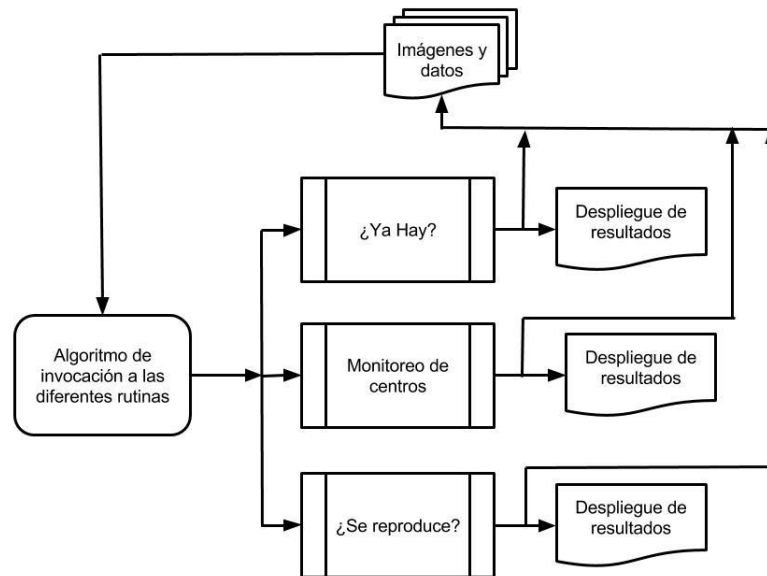


Diagrama 12. Estructura del algoritmo, donde el sentido de las flechas indica la dirección del flujo de datos entre los diferentes elementos del sistema.

En el Diagrama 12 se ilustra el funcionamiento general del sistema: en el algoritmo de control de procesos se definen los parámetros iniciales y se cargan las imágenes en el ambiente de trabajo, con estos datos se hace una llamada a las subrutinas que analizan los datos e interpretan por sí mismas los resultados guardando los datos en archivos de texto a los que se puede acceder a través del algoritmo de control en la siguiente iteración desde diferentes rutinas, las cuales pueden modificar dichos archivos que representan las variables del sistema (simulando una memoria externa a la arquitectura).

Para utilizar la herramienta de MATLAB HDL Coder, el algoritmo debe cumplir ciertos requerimientos, por lo que tuvo que ser sometido a una reestructuración en la que se le hicieron los cambios que enlistaremos a continuación:

- Se revisaron y modificaron algunos métodos para que se tuviera compatibilidad con las funciones soportadas por HDL Coder.
- En lugar de invocar los métodos y hacer un procesamiento único y secuencial, se generó un selector que, según los parámetros de entrada, indica a cada módulo la rutina que debe seguir de acuerdo a la interpretación de los datos arrojados por el método que le antecede.
- En el código de MATLAB se carga una imagen desde un archivo en la computadora, para hacer esta lectura compatible con el concepto de descripción de hardware se generó un proceso de lectura en el cual se lee pixel por pixel (simulando la entrada de datos a través de un puerto de la arquitectura) y se va procesando al tiempo que se lee. Estos primeros resultados (generalmente binarizados) se van vaciando en una memoria interna que definimos como un arreglo

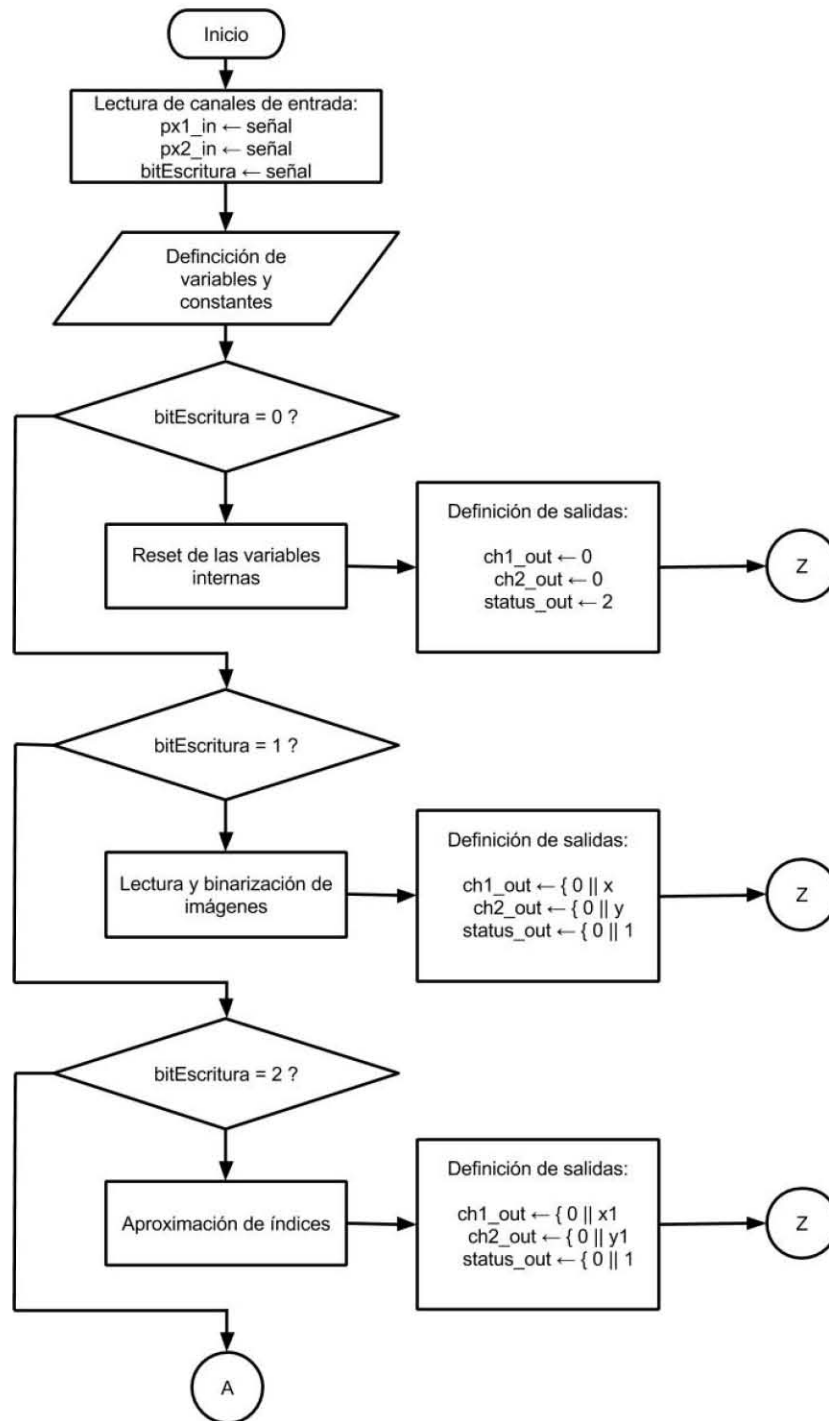
- bidimensional del tamaño del recorte que representa el área de interés de cada trampa (40 x 40 px).
- Se eliminaron todas las estructuras *for*, cuya implementación, a pesar de ser posible, era costosa en recursos y se diseñaron en su lugar métodos por contadores para el control de los índices de las matrices, lo que arrojó resultados exitosos.

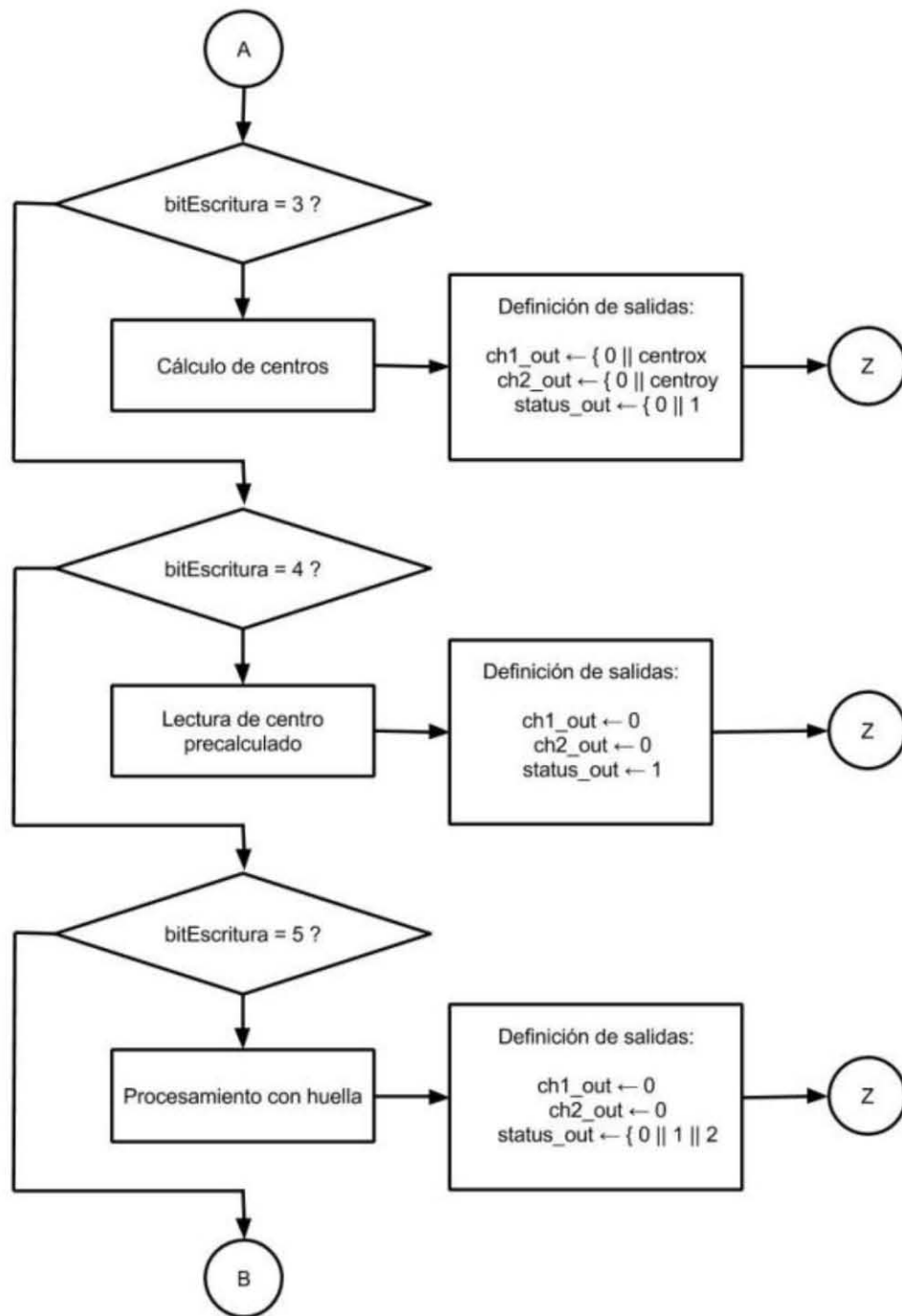
El proceso de conversión entre los diseños de MATLAB a hardware constó de los siguientes pasos:

1. Se hizo el diseño en MATLAB para depurar, probar y optimizar los módulos que conforman el sistema de visión.
2. Se cambió la estructura de subrutina que presentaba cada módulo por una estructura de función. La diferencia que hace esto es que en la primera se ejecutaba un método independiente de la rutina principal que, como vimos en el Diagrama 1, interactuaba sólo con los archivos de salida y hacía el procesamiento de la imagen y la toma de decisiones de manera secuencial en un solo bloque. En la nueva definición se separó el procesamiento de la imagen (lo que describiremos en hardware) de la lógica de toma de decisiones a partir de la cual se diseñó el algoritmo de toma de decisiones, llamado testbench (que puede ser programado en un microprocesador embebido).
3. Se diseñó un testbench para cada módulo de procesamiento, los cuales tienen dos funciones principales: la primera es la de hacer la interpretación de resultados arrojados por cada módulo de procesamiento y, a través de esta interpretación, controlar los parámetros de entrada al módulo en la siguiente iteración, y la segunda es la validación de que este procesamiento se está llevando de forma correcta, es decir, al invocar a la función (módulo de procesamiento de imágenes) con el testbench se valida que los resultados obtenidos de la parte de procesamiento corresponden con los resultados del primer diseño del algoritmo.
4. En la nueva estructura del sistema la función debe ser llamada un cierto número de veces con los parámetros de entrada correspondientes a la rutina que se requiere ejecutar, esto equivale a enviar las señales de entrada por cada pulso de reloj lo cual se lleva a cabo por el testbench con la ayuda de ciclos. Esto nos indica que la velocidad de procesamiento de los módulos estará sujeta a la señal de reloj, cuya frecuencia puede ser modificada por el microprocesador, o estar fija a una señal de reloj de frecuencia constante.
5. Se generó un banco de datos que contiene imágenes e información referida a éstas, que alimenta al sistema y que es representativo de los diferentes casos que se pudieran presentar, con lo que se definieron los tipos de variable usados y los resultados que se esperaban obtener del procesamiento.
6. En MATLAB se procesó el código del módulo de procesamiento de imagen con el método de Punto fijo y con ello se generó posteriormente el código HDL para la creación de prototipos en el FPGA.
7. Se validó que la simulación en el FPGA y el código MATLAB arrojaron los mismos resultados reutilizando el banco de datos generado para las pruebas.
8. Se hizo un análisis con el software ISE versión 14.7 de Xilinx de los recursos en hardware que ocupa cada módulo de procesamiento de imagen del sistema de visión en un FPGA.

Generación de funciones para la conversión de lenguaje

Método “¿Ya Hay?”: Los procesos que componen este método ya se han explicado exhaustivamente en el capítulo de diseño. En el Diagrama 13 se presenta de manera general la forma en que la subrutina quedó estructurada en una función.





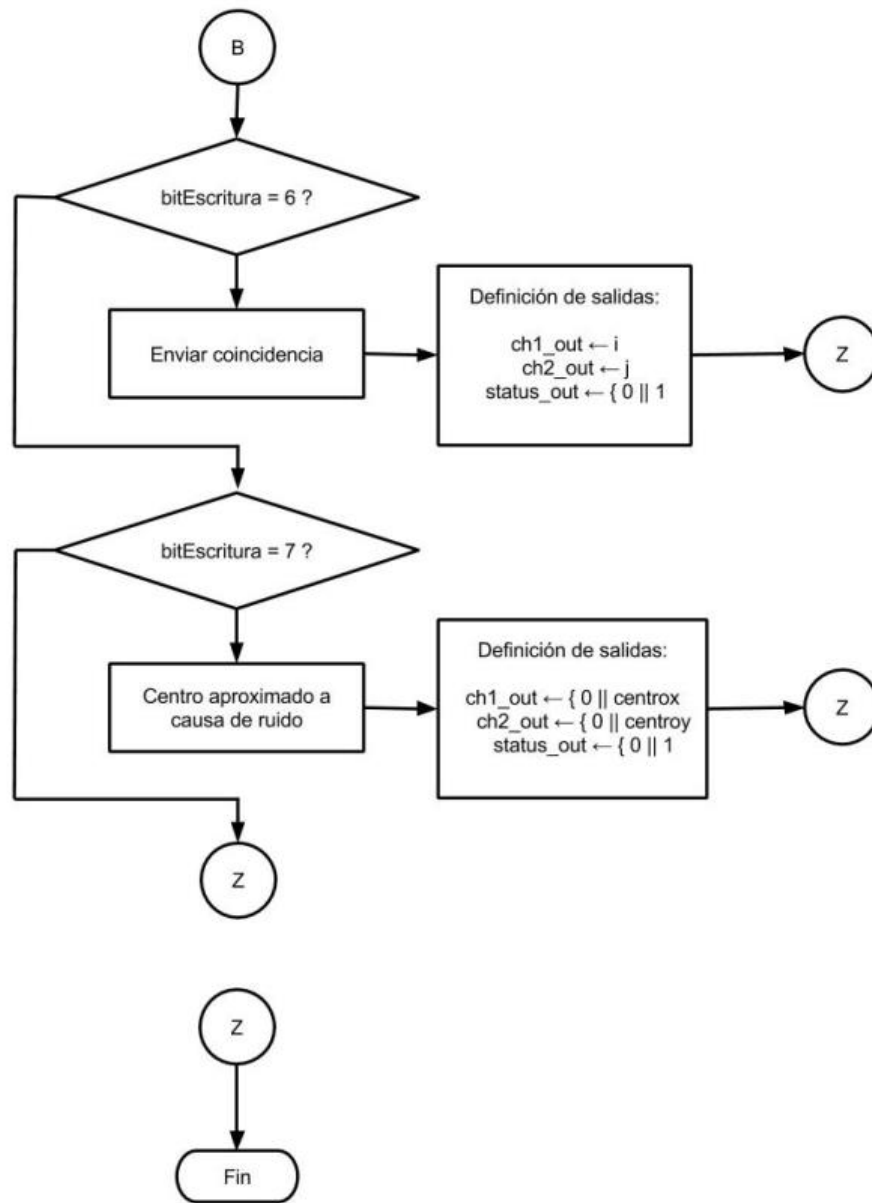
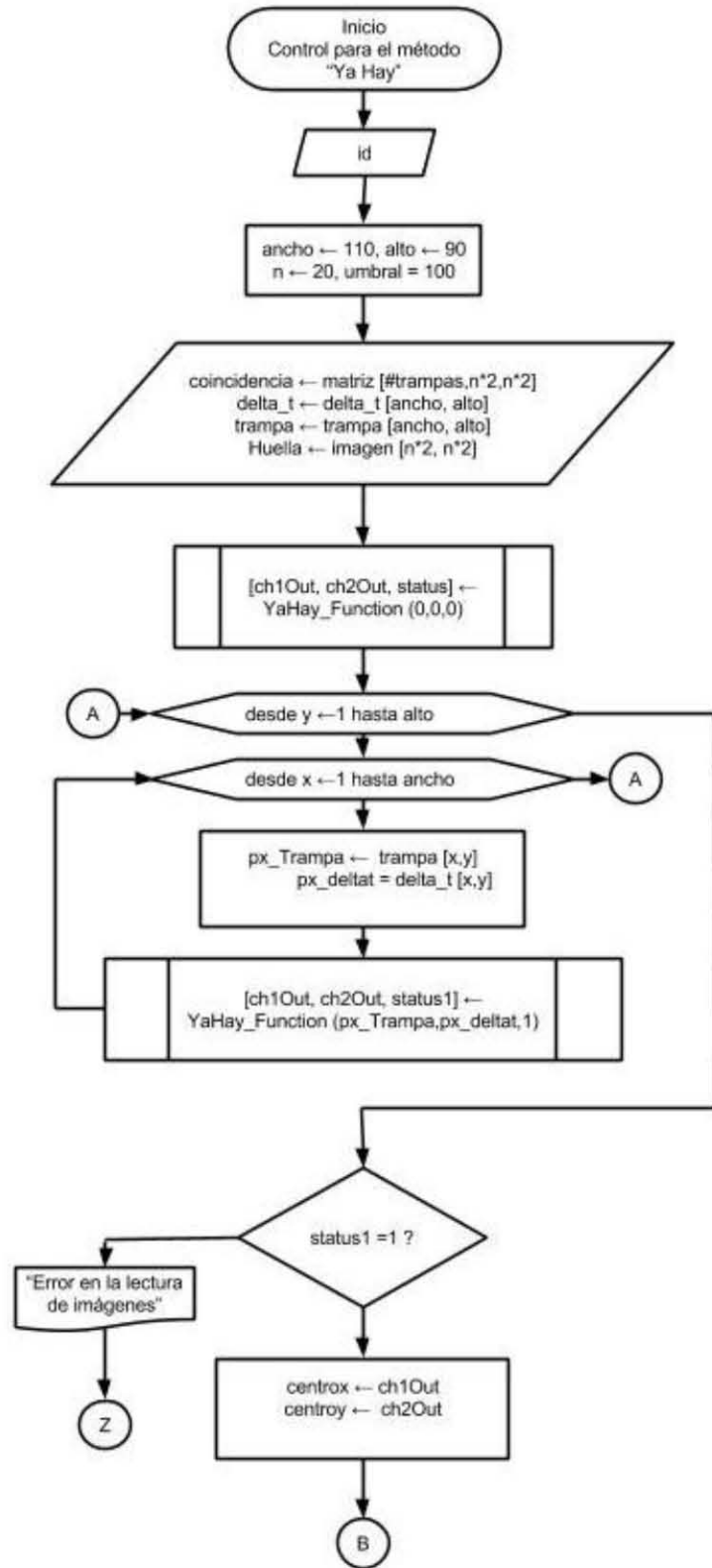
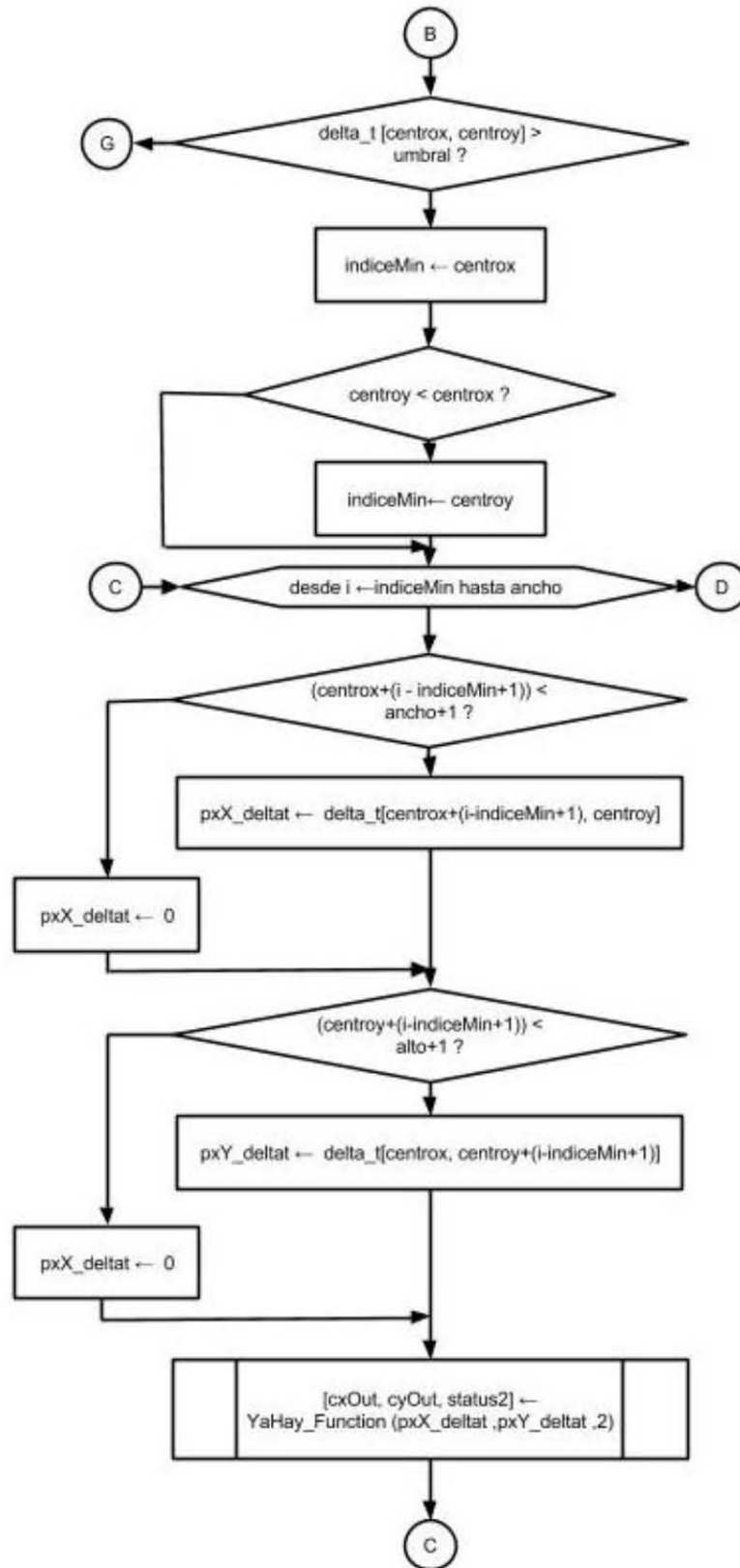
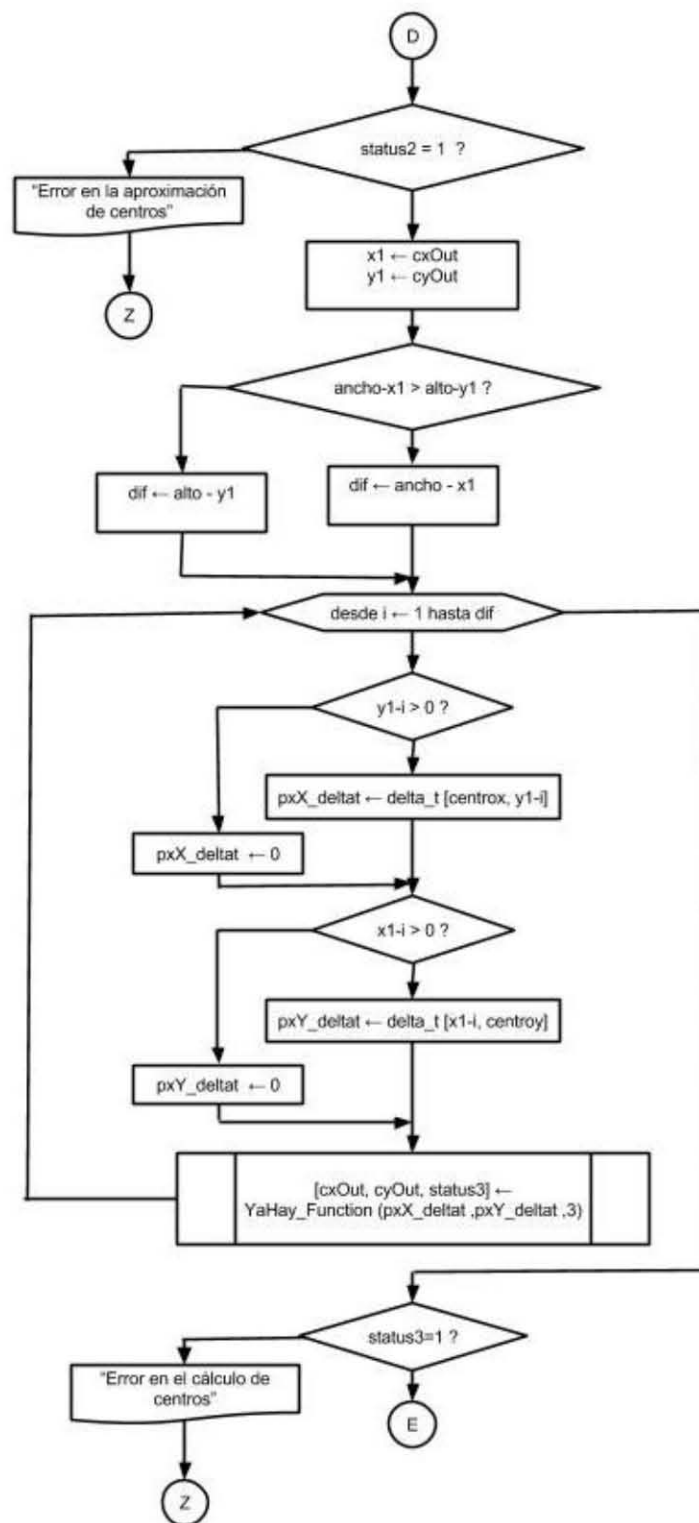


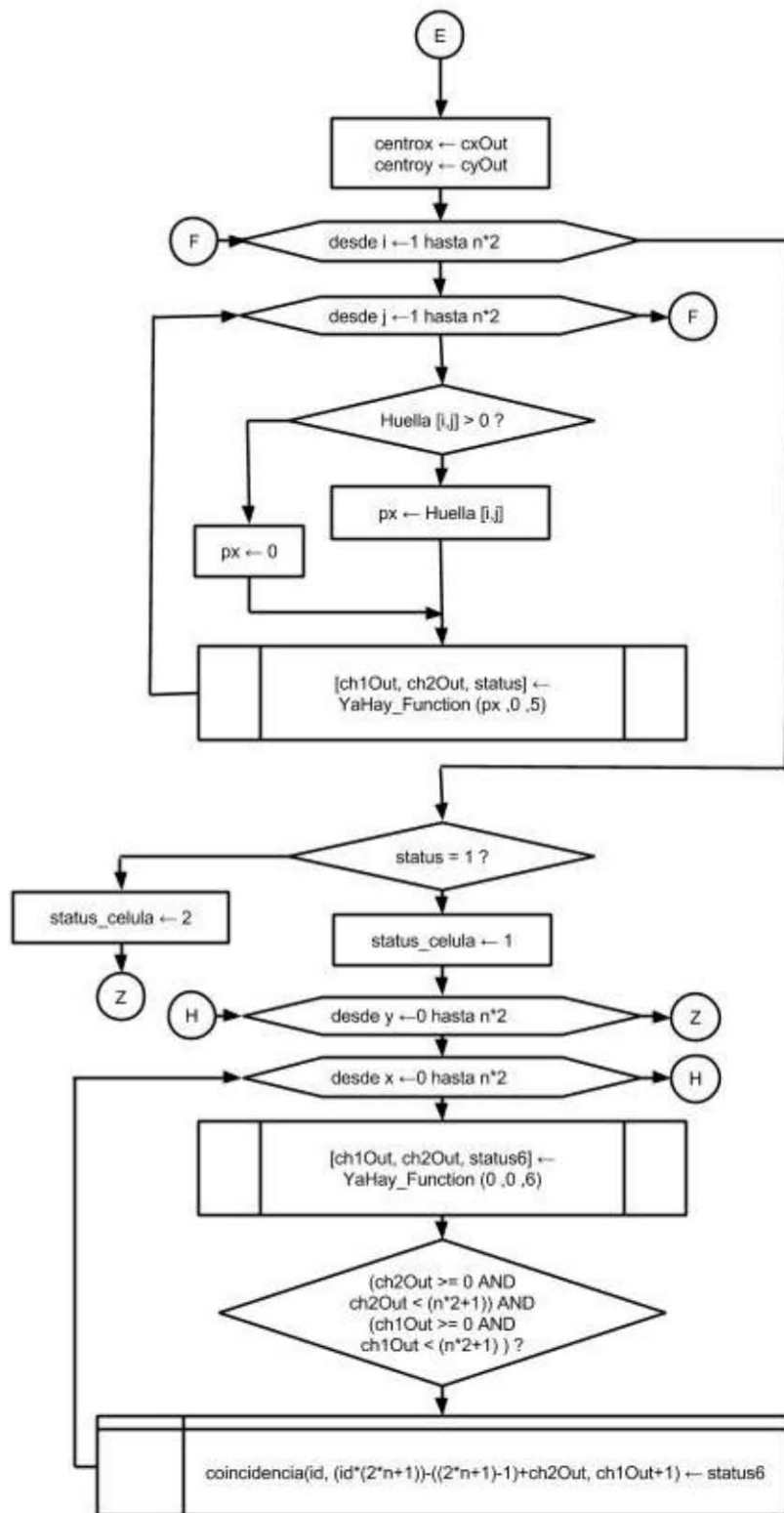
Diagrama 13. Método “Ya Hay”, donde se muestra que dependiendo del bitEscritura, que funciona como selector, se ejecuta alguna de las rutinas secundarias dentro del módulo.

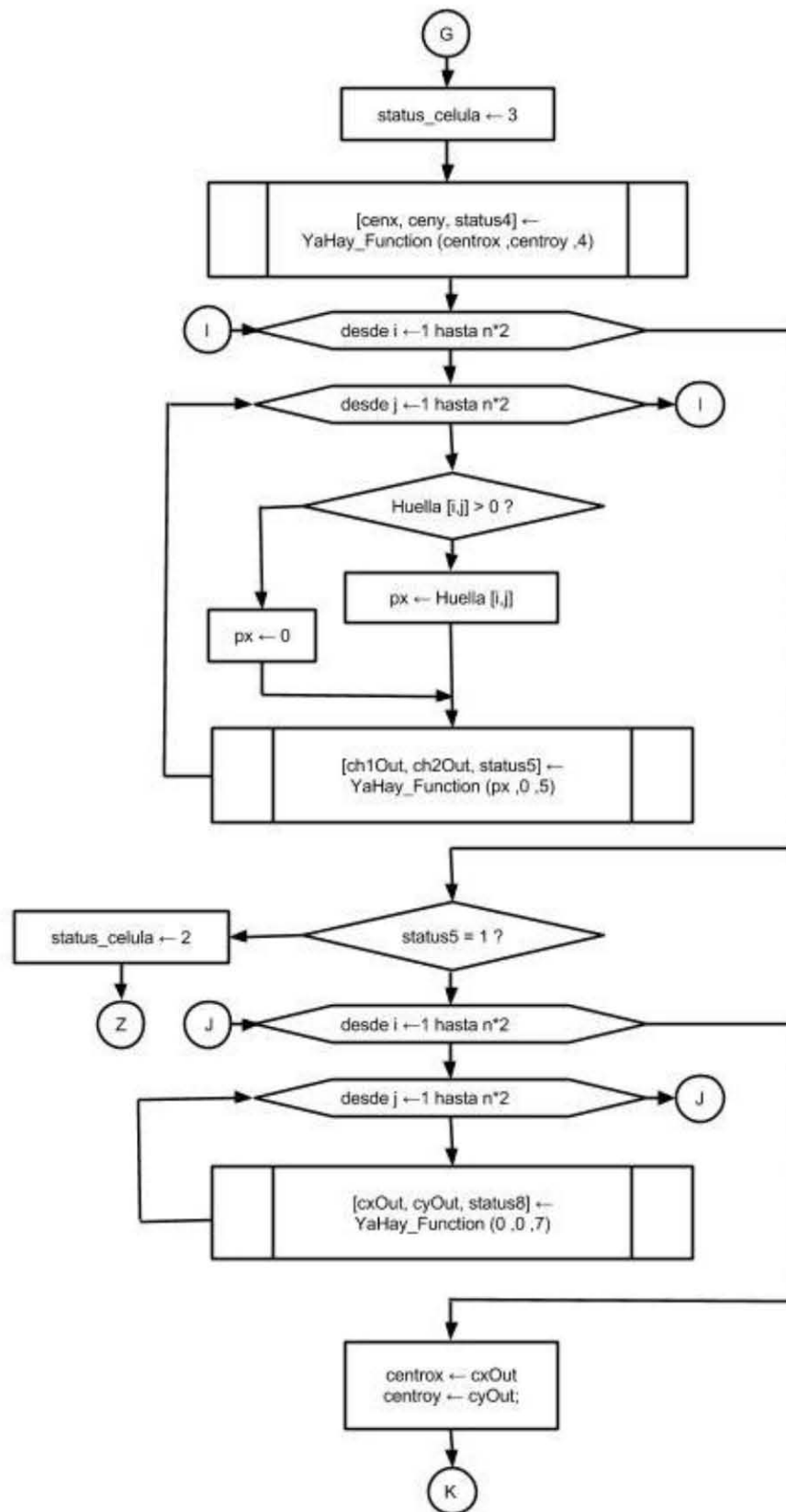
Una vez convertida la función se procedió a hacer un algoritmo de toma de decisiones *testbench* que fuera capaz de interpretar las salidas para generar nuevas entradas para el algoritmo. Su diseño, como ya hemos explicado con anterioridad, fue de gran importancia pues esta lógica es la que controla e interpreta los datos generados, es decir, en el *testbench* se resumen las acciones de control que se deben tomar una vez terminado el procesamiento de la imagen. El algoritmo para controlar el proceso para saber si ya hay célula en la trampa queda descrito en el Diagrama 14.











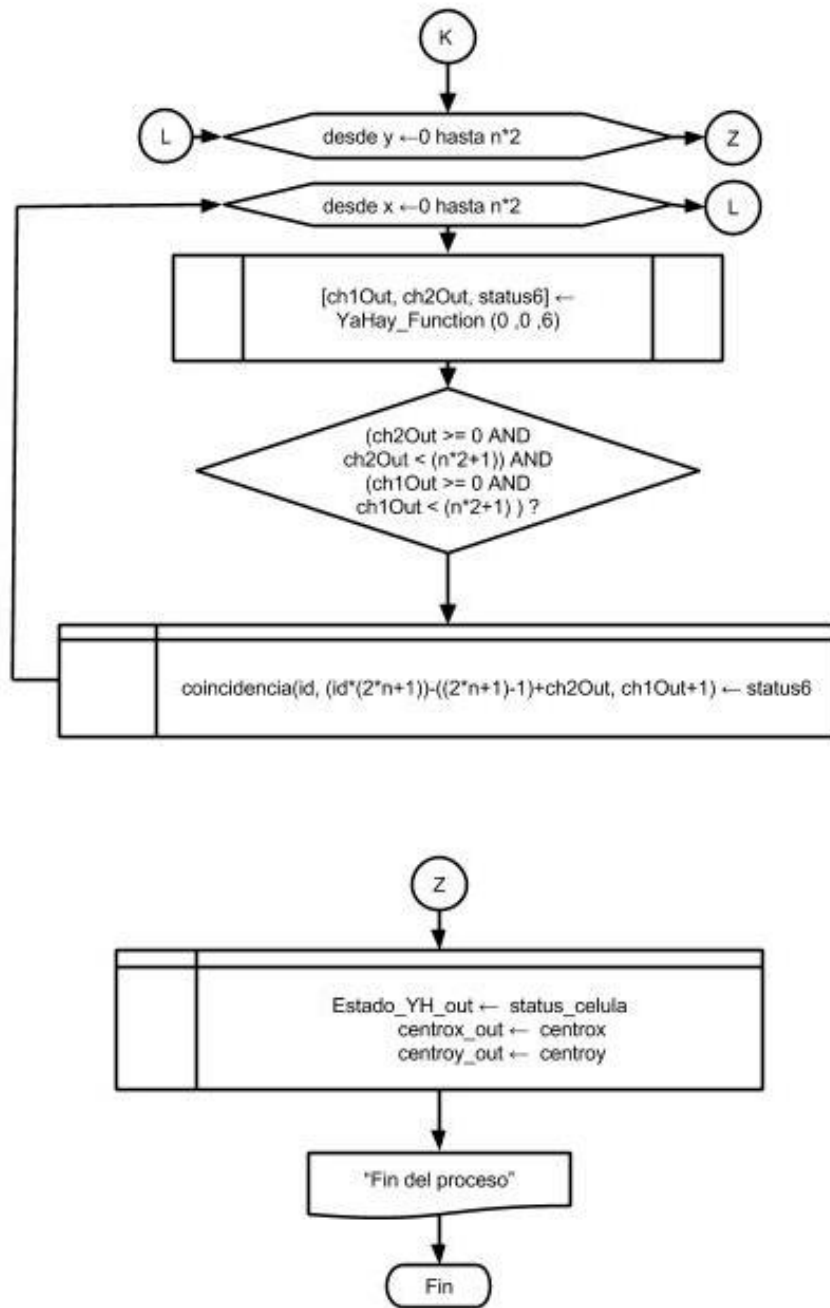


Diagrama 14. Lógica del testbench, o algoritmo de control para el método “Ya Hay”.

De este proceso de control o *testbench* al final nos interesan tres únicos datos que resultan de todo el procesamiento: el estatus final del análisis del módulo que indicará el estado de la trampa, la matriz de coincidencia de pixeles que representan la célula y los valores en coordenadas rectangulares del centro en caso de que el resultado fuese positivo. A continuación se muestra la interpretación de estas tres salidas:

Datos de salida del análisis “¿Ya Hay?”			
Estatus	Centro	Matriz de coincidencia	Interpretación
0	~	~	No ha inicializado el análisis
1	(cx, cy)	Matriz [40,40]	Sí hay célula en la trampa
2	~	~	No hay célula
3	(cx, cy)	Matriz [40,40]	Hay ruido y posible célula

Tabla 11. Variables de salida del testbench para el método “Ya Hay” y su respectiva interpretación.

Una vez teniendo este algoritmo genérico para el control de la función “¿Ya Hay?”, para probarlo se procedió a escoger diferentes trampas que representarían los tres diferentes casos en los que puede caer este algoritmo:

Caso 1. Sí hay célula: Para probar este caso se utilizó la trampa número 4 por presentar una célula de dimensiones más pequeñas que el promedio, lo cual representa un reto para un algoritmo basado en estadística. Se utilizaron las capturas 0 (la trampa sin célula) y 2 (trampa con célula en un intervalo de 8 minutos aproximadamente), donde se observaron los siguientes resultados en MATLAB:

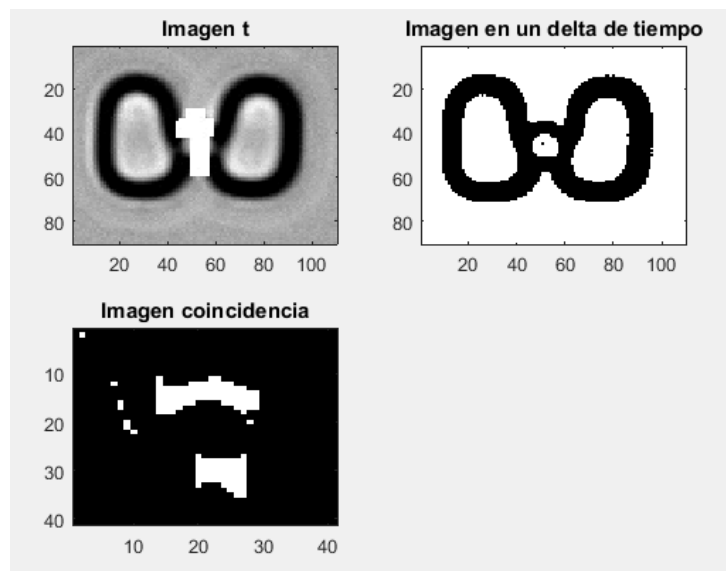


Figura 28. En la imagen arriba a la izquierda observamos la trampa 4 vacía, a la derecha la misma trampa después de un delta de tiempo donde detecta la presencia de una célula y despliega un pixel auxiliar (en el centro de la célula) para visualizar el primer

cálculo del centro que hace este algoritmo y muestra los píxeles que detecta como parte de esta célula en la imagen de coincidencia.

```
Estatus: 1, cx: 51, cy: 45. >>
```

Figura 29. Resultado de la consola de MATLAB para el análisis de la trampa 4 en el delta de tiempo de la captura 2, donde el Estatus igual a 1 indica que hay presencia de célula y nos da sus coordenadas cartesianas en el eje horizontal (cx) y vertical (cy).

Caso 2. No hay célula: Para este caso se utilizó la trampa número 30, las capturas 0 y 5, que representan un intervalo aproximado de 40 min, lo que podría generar ruido en la imagen. Al correr la función con el testbench se generaron los siguientes resultados:

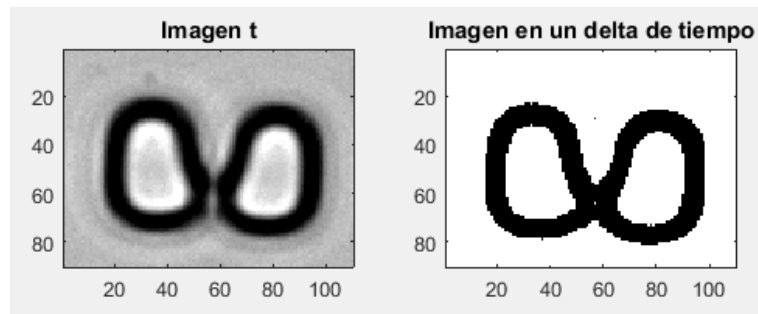


Figura 30. En la imagen a la izquierda observamos la trampa 30 vacía, a la derecha la misma trampa después de un delta de tiempo donde no se detecta presencia de una célula.

```
Estatus: 2, cx: 57, cy: 29. >>
```

Figura 31. Resultado de la consola de MATLAB para el análisis de la trampa 30 en el delta de tiempo de la captura 5, donde el Estatus igual a 2 indica que no hay célula y nos da las coordenadas cartesianas de la densidad de píxeles de cambio en el eje horizontal [cx] y vertical [cy], (éstas a pesar de no ser utilizadas, son de ayuda para la visualización de que el algoritmo está funcionando correctamente).

Caso 3. Hay ruido en la imagen, posible célula: Para este caso se utilizó la trampa 40, los frames 0 y 25, que tienen una distancia temporal de 1 hora 40 minutos, lo que puede representar ruido en la imagen. Se observaron los siguientes resultados:

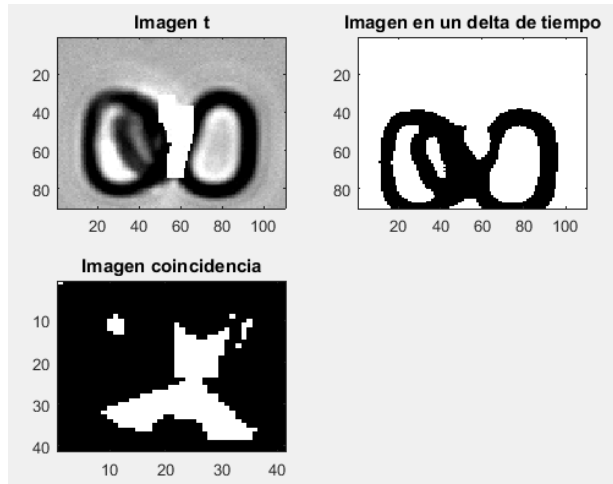


Figura 32. En la imagen arriba a la izquierda observamos la trampa 40 vacía, a la derecha la misma trampa después de un delta de tiempo donde detecta ruido en la imagen y muestra los pixeles que podrían pertenecer a una célula en la imagen de coincidencia para ser analizada en la siguiente iteración.

`Estatus: 3, cx: 32, cy: 51. >>`

Figura 33. Resultado de la consola de MATLAB para el análisis de la trampa 40 en el delta de tiempo de la captura 25, donde el Estatus igual a 3 indica que hay ruido en la imagen y nos da las coordenadas cartesianas de la densidad de pixeles de cambio en el eje horizontal [cx] y vertical [cy], para poder reanudar el análisis en la siguiente iteración.

Una vez que validamos que tanto el testbench como la función generan los resultados esperados en los diferentes casos posibles del algoritmo, procedimos a generar su código en VHDL con la herramienta HDL Coder. Se sintetizó en el software ISE versión 14.7 de Xilinx, de donde la arquitectura sintetizada se muestra en la Figura 34.

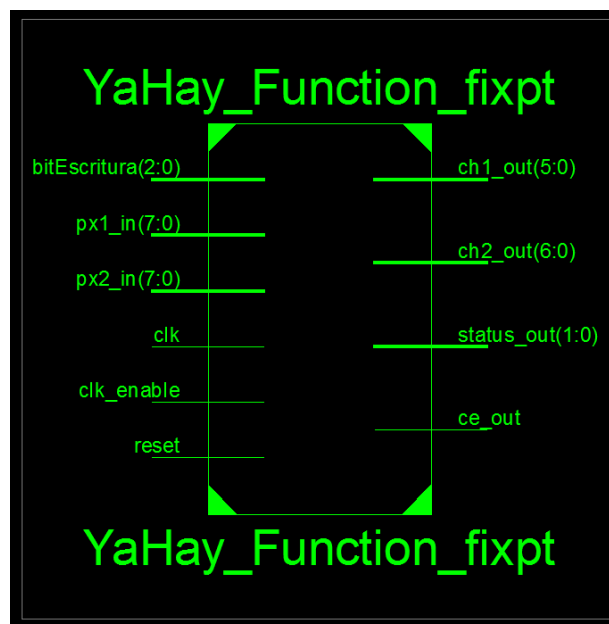


Figura 34. Arquitectura generada para la función “¿Ya Hay?”.

Como vemos se respetaron los puertos que se definieron en el diseño de la arquitectura y se agregaron los puertos *clk_enable* y *ce_out*, el primero es un puerto de entrada que, como su nombre lo indica, permite la activación del ciclo del reloj y con ello funciona como el *enable* del módulo, y el puerto de salida *ce_out* que simplemente indica cuando el módulo está activado.

Método “Monitoreo de centros”: La función modificada en un esquema general quedó estructurada como se muestra en el Diagrama 15.

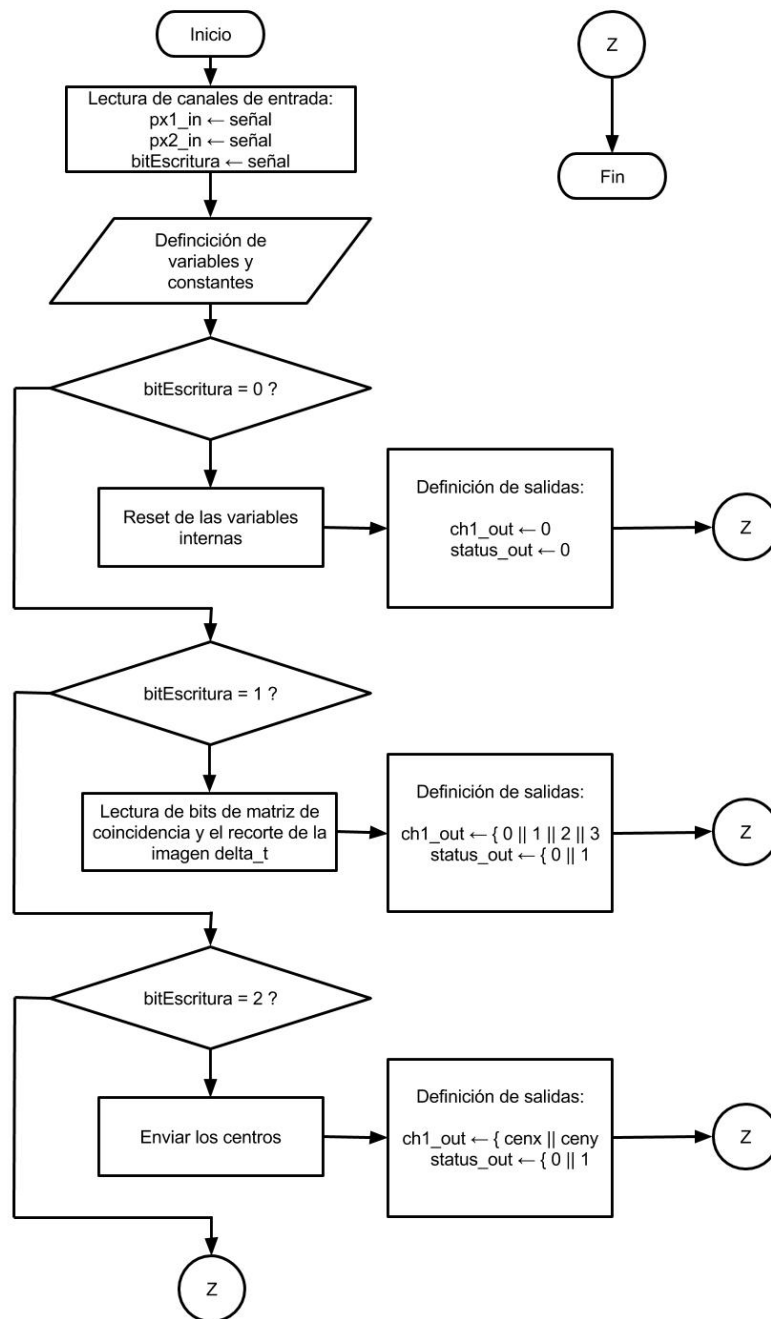
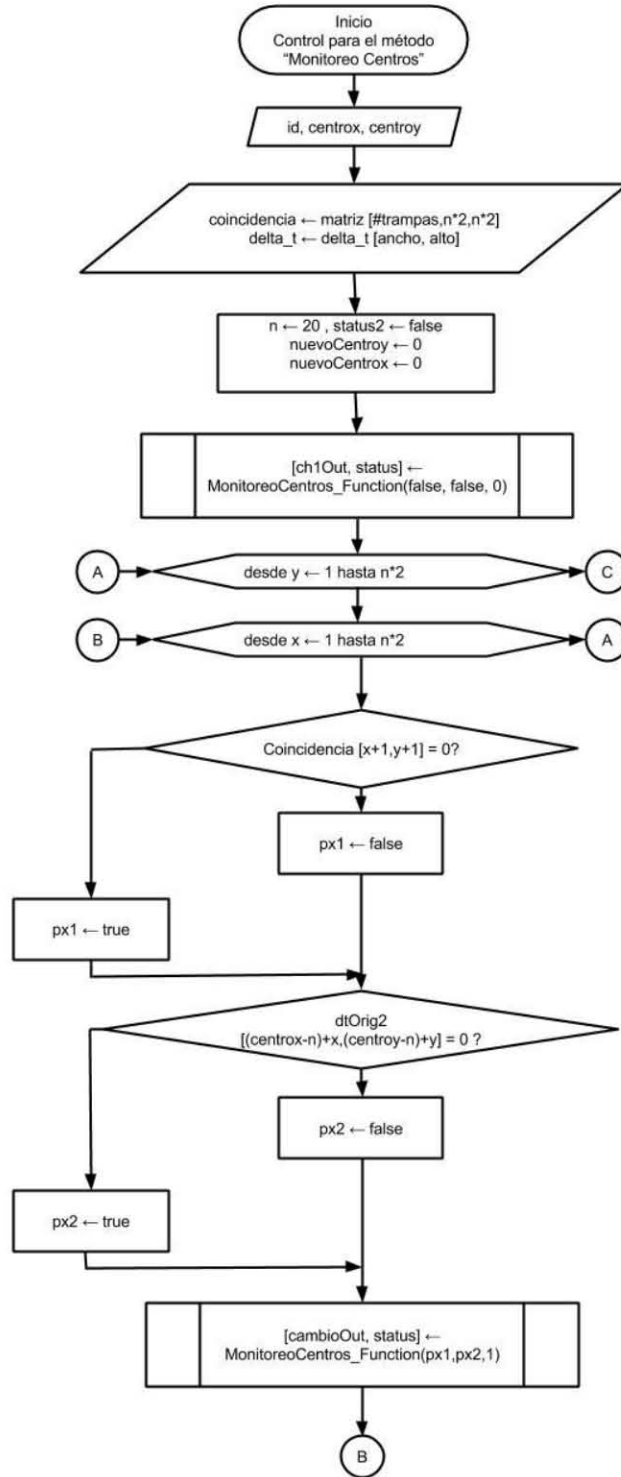
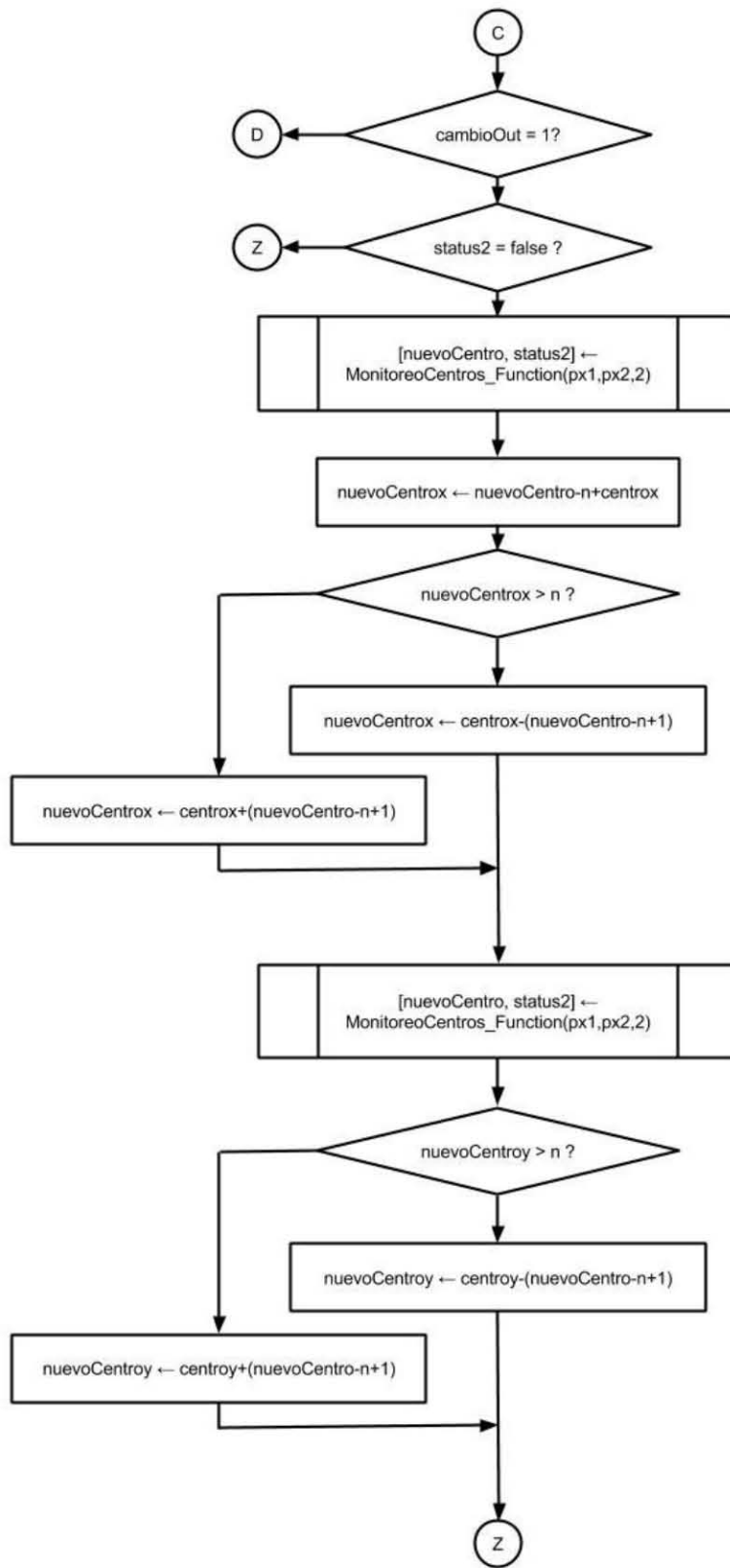
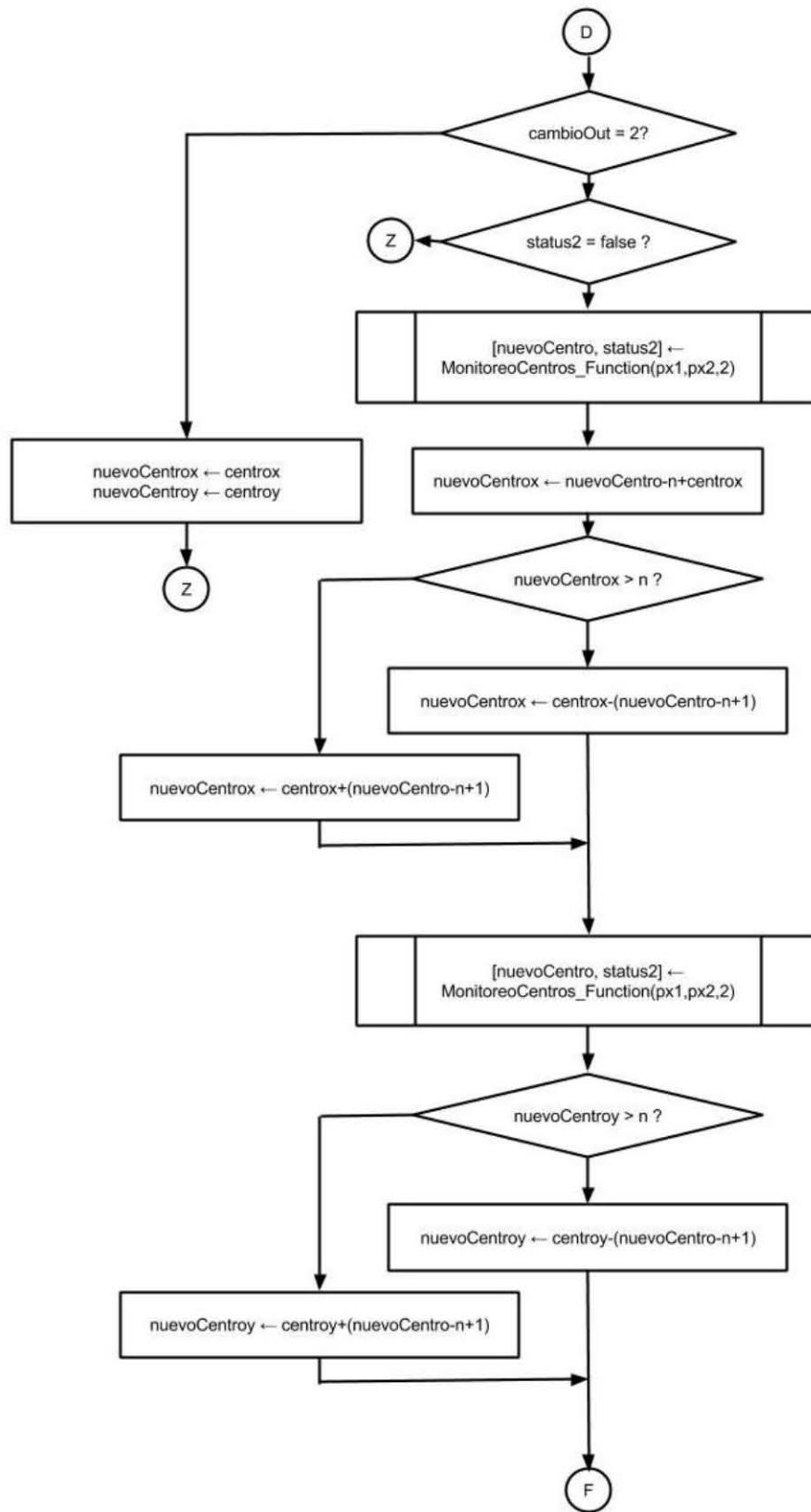


Diagrama 15. Método “Monitoreo de centros”, donde se muestra que dependiendo del bitEscritura, que funciona como selector, se ejecuta alguna de las rutinas secundarias dentro del módulo.

Teniendo esta función se procedió a hacer el método de testbench para interpretar las salidas y generar nuevas entradas para la siguiente iteración del algoritmo, es decir, las acciones de control que se deben tomar una vez obtenidos los resultados del análisis. El algoritmo para el monitoreo de centros queda sintetizado en el Diagrama 16.







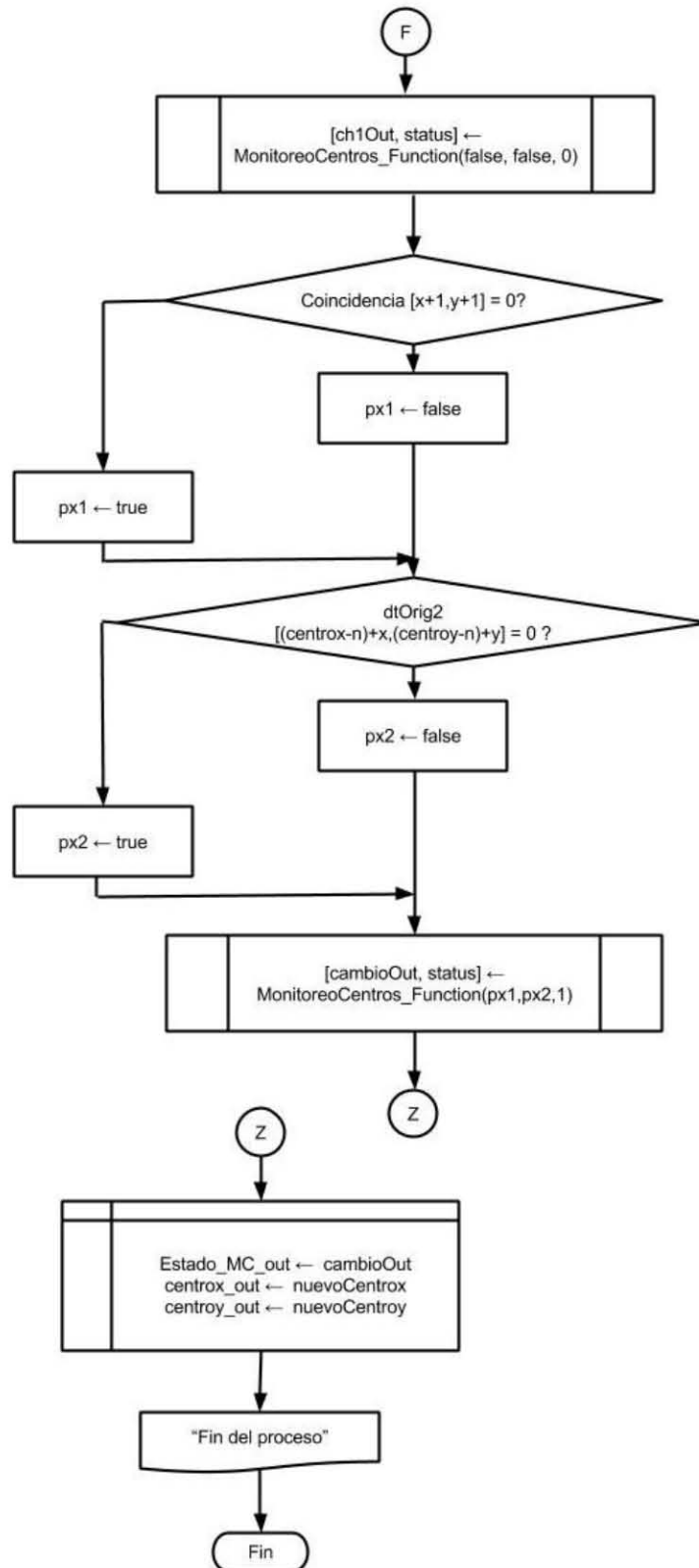


Diagrama 16. Lógica del testbench, o algoritmo de control para el método “Monitoreo de Centros”.

De este proceso de control de variables de entrada o testbench al final nos interesan dos datos que resultan de todo el procesamiento: el estatus final del análisis del módulo que indicará el estado del cambio en la estructura de los pixeles que componen a la célula, y los valores en coordenadas rectangulares del centro de la misma. A continuación se muestra la interpretación de estas salidas:

Datos de salida del análisis "Monitoreo de centros"		
Estatus	Centro	Interpretación
0	~	No ha inicializado el análisis
1	(cenx, ceny)	Cambio pequeño o nulo en la estructura celular
2*	(cenx, ceny)	Existe un cambio en la estructura, pero el centro es similar *En este caso se hace un nuevo análisis para forzar la salida a (1 3)
3	~	Hay un cambio significativo en la estructura, se debe hacer nuevamente el proceso "¿Ya Hay?"

Tabla 12. Variables de salida del testbench para el método "Monitoreo de Centros" y su respectiva interpretación.

Una vez teniendo la interpretación de resultados y el algoritmo de control para el monitoreo de centros, se procedió a escoger las imágenes representativas de los diferentes casos en los que puede caer el procesamiento:

Caso 1. Cambio pequeño o nulo en la estructura celular: en este caso los pixeles que representan la célula no tienen gran variación en el intervalo de las imágenes, por lo que el centro se corrige ligeramente y se continúa el análisis con la misma matriz de coincidencia estructural para la siguiente iteración del algoritmo.

Para las pruebas se utilizó la trampa 94, las capturas 2 y 4, lo que representa el cambio en la trampa en un tiempo de 8 minutos.

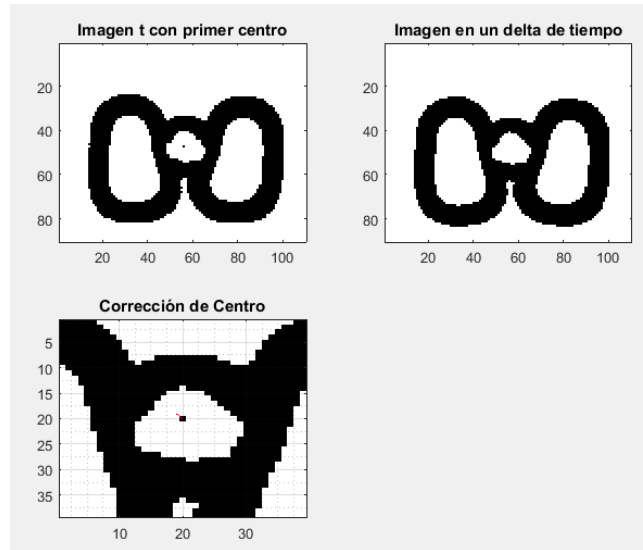


Figura 35. En la imagen arriba a la izquierda vemos la imagen de la trampa 94 con el centro de la célula marcado con un pixel auxiliar, a la derecha la misma trampa después de un tiempo delta donde la célula presenta movimiento y en la última imagen abajo a la izquierda el vector de movimiento en rojo del nuevo centro calculado con respecto al anterior.

```
Cambio= 1, cenx= 57, ceny= 48.
Centros anteriores cenx= 56, ceny= 47.
>> |
```

Figura 36. Resultado de la consola de MATLAB para el análisis de monitoreo de centros de la trampa 94, donde vemos que la variable de Cambio al estar en 1 nos indica que hubo movimiento de la célula y nos da el nuevo centro calculado en coordenadas cartesianas en el eje horizontal (cenx) y vertical (ceny), además de las coordenadas del centro anterior a modo de comparación.

Caso 2. Cambio en la estructura celular: este caso se presenta cuando hay un cambio en la estructura celular debido a un ligero movimiento o al crecimiento de la célula, lo que generalmente produce que no coincida por completo la huella de pixeles que se presenta en imágenes anteriores, pero que la distancia entre los centros calculados sea pequeña, por lo que el algoritmo después de calcular el nuevo centro realiza nuevamente el análisis de coincidencia estructural comparando la nueva imagen (delta_t) con la imagen de coincidencia calculada en los procesos anteriores (esto con el fin de disminuir el error generado por un desplazamiento), lo que termina generando una determinación clara de si el cambio de pixeles es aceptable para continuar el análisis con la misma imagen de coincidencia, en cuyo caso el estatus arroja un 1, o si se debe hacer de nuevo la determinación de esta matriz mediante el método “¿Ya Hay?”, lo cual se indica con el estatus 3.

Para la realización de estas pruebas se utilizó la trampa 94, las capturas 2 y 10, que representan una distancia temporal de 48 minutos.

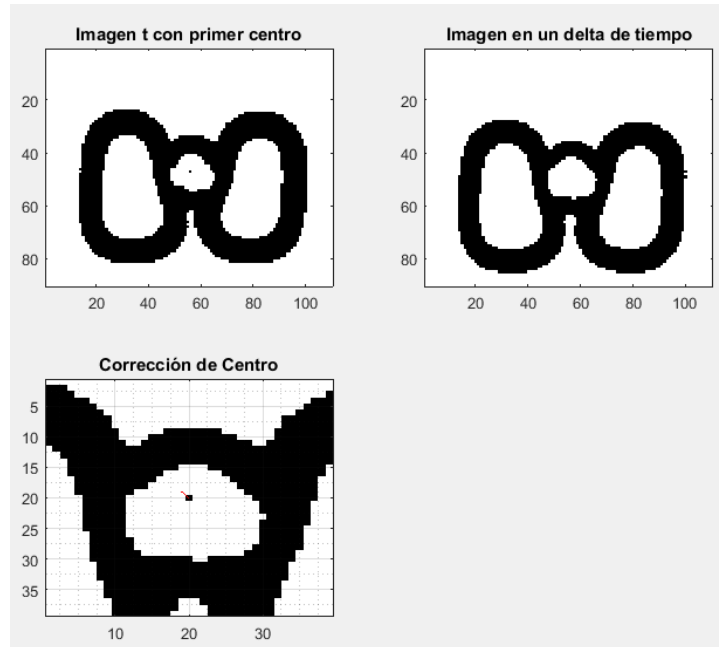


Figura 37. En la imagen arriba a la izquierda vemos la imagen de la trampa 94 con el centro de la célula marcado con un pixel auxiliar, a la derecha la misma trampa después de un tiempo delta donde la célula presenta además de movimiento un cambio en su estructura, y en la última imagen abajo a la izquierda el vector de movimiento en rojo del nuevo centro calculado con respecto al anterior.

```

Hay un cambio en la estructura de pixeles. (status 2)
Cambio= 1, cenx= 57, ceny= 48.
Centros anteriores cenx= 56, ceny= 47.
>> |

```

Figura 38. Resultado de la consola de MATLAB para el análisis de monitoreo de centros de la trampa 94, donde vemos un mensaje de que se ha detectado un cambio en la estructura de la célula, y nos da el nuevo centro calculado en coordenadas cartesianas en el eje horizontal (cenx) y vertical (ceny), además de las coordenadas del centro anterior a modo de comparación.

Caso 3. Demasiado cambio en la estructura: en este caso el cambio en la estructura de pixeles con respecto a la imagen de coincidencia calculada en capturas anteriores ya no es aceptable para este método y se debe ejecutar nuevamente la función “¿Ya Hay?” para actualizar la coincidencia estructural que se use en las siguientes iteraciones del algoritmo.

Para estas pruebas se utilizaron las capturas 2 y 47 de la trampa 94, lo que supone un intervalo de 6 horas.

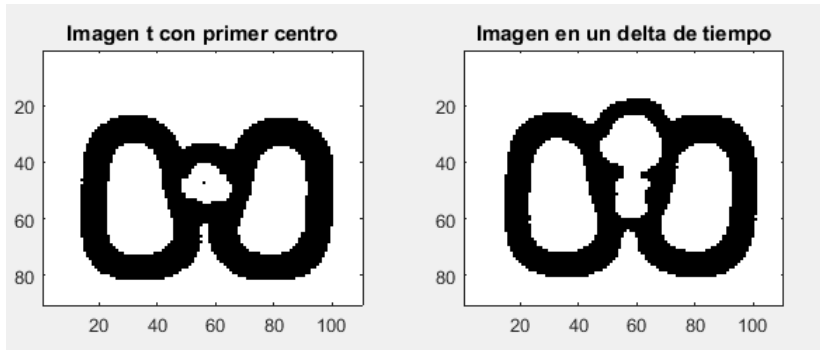


Figura 39. . En la imagen a la izquierda observamos la trampa 94 con la célula detectada, y a la derecha la misma trampa en un delta de tiempo equivalente a seis horas, en el que el cambio de la estructura es lo suficientemente grande para que no lo detecte como la misma estructura calculada con anterioridad.

```
Cambio -> 3, Hacer de nuevo YaHay.
Cambio= 3, cenx= 56, ceny= 47.
Centros anteriores cenx= 56, ceny= 47.
>> |
```

Figura 40. Resultado de la consola de MATLAB para el análisis de monitoreo de centros de la trampa 94, donde vemos el mensaje de que se ha detectado un cambio grande en la estructura de la célula, por lo que deja los datos del centro sin cambios y manda la señal para que vuelva a efectuarse el método “Ya Hay”.

Una vez que validamos que tanto el *testbench* como la función de monitoreo de centros generan los resultados esperados en los diferentes casos posibles del algoritmo, procedimos a generar su código en VHDL con la herramienta HDL Coder. Se sintetizó en el software ISE versión 14.7 de Xilinx y generó la arquitectura como se muestra en la Figura 41.

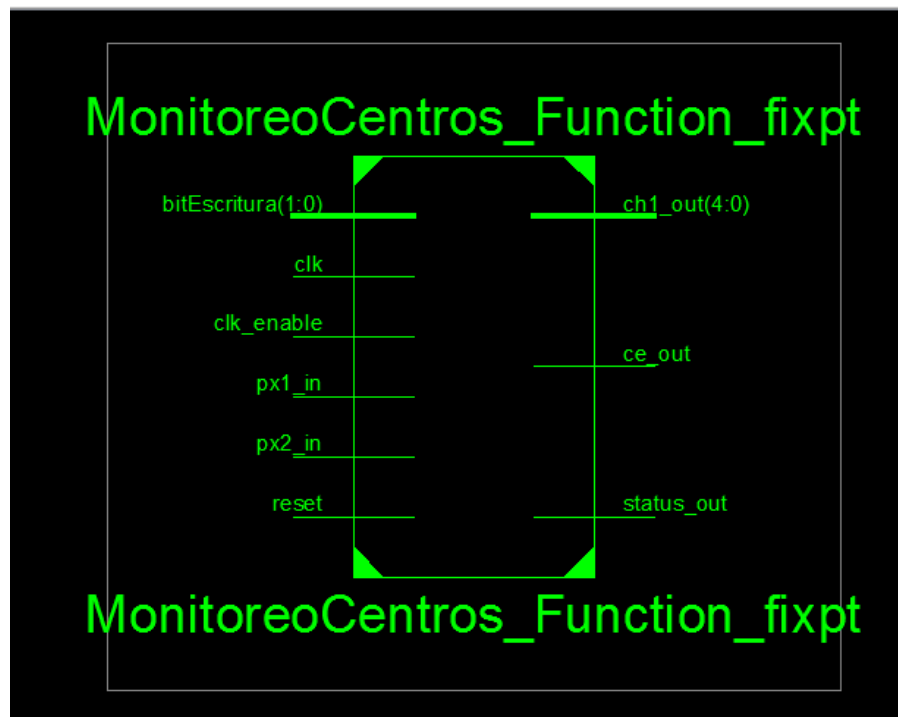


Figura 41. Arquitectura del módulo Monitoreo de centros sintetizado.

Método “¿Se Reproduce?”: La función modificada en un esquema general quedó estructurada como se muestra en el Diagrama 17.

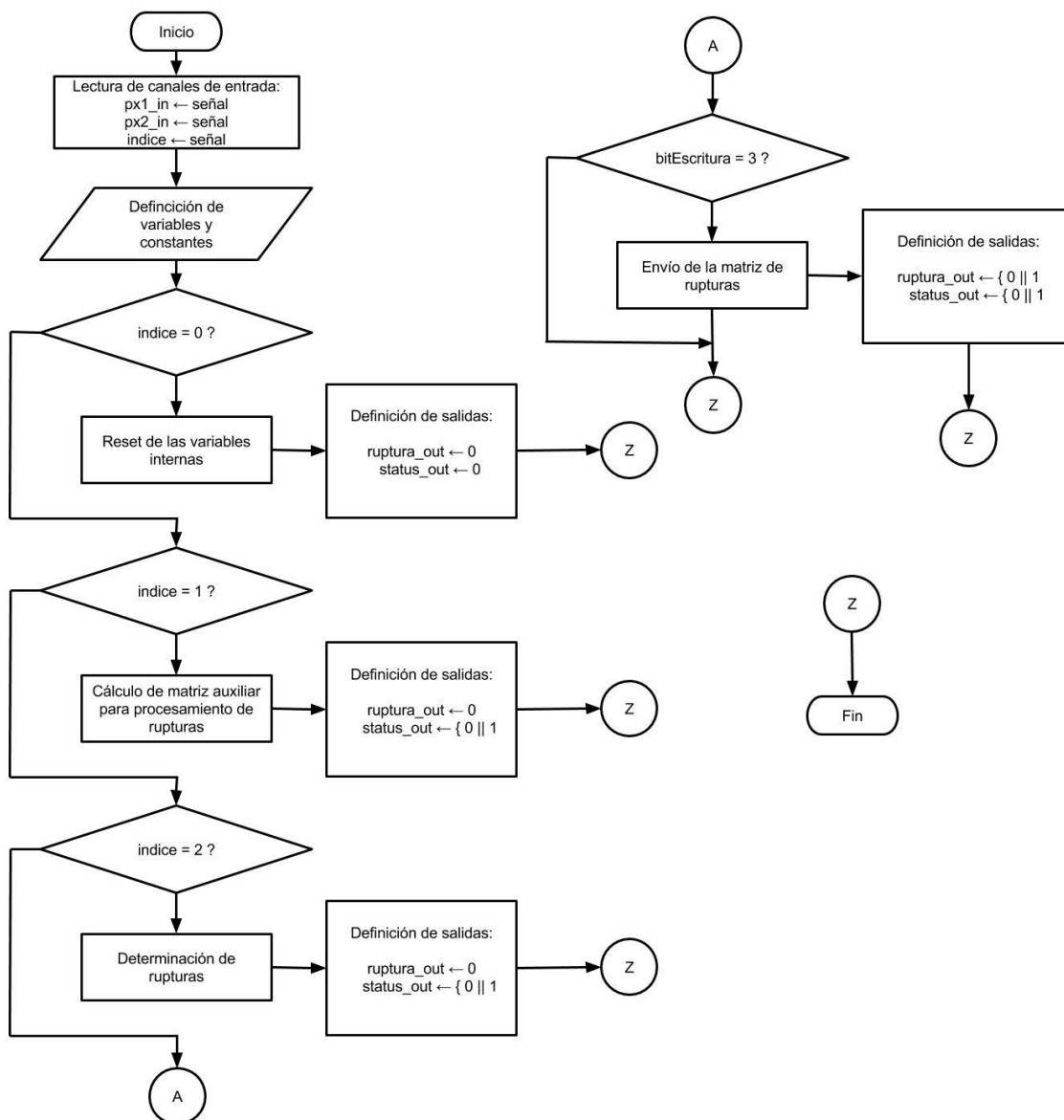
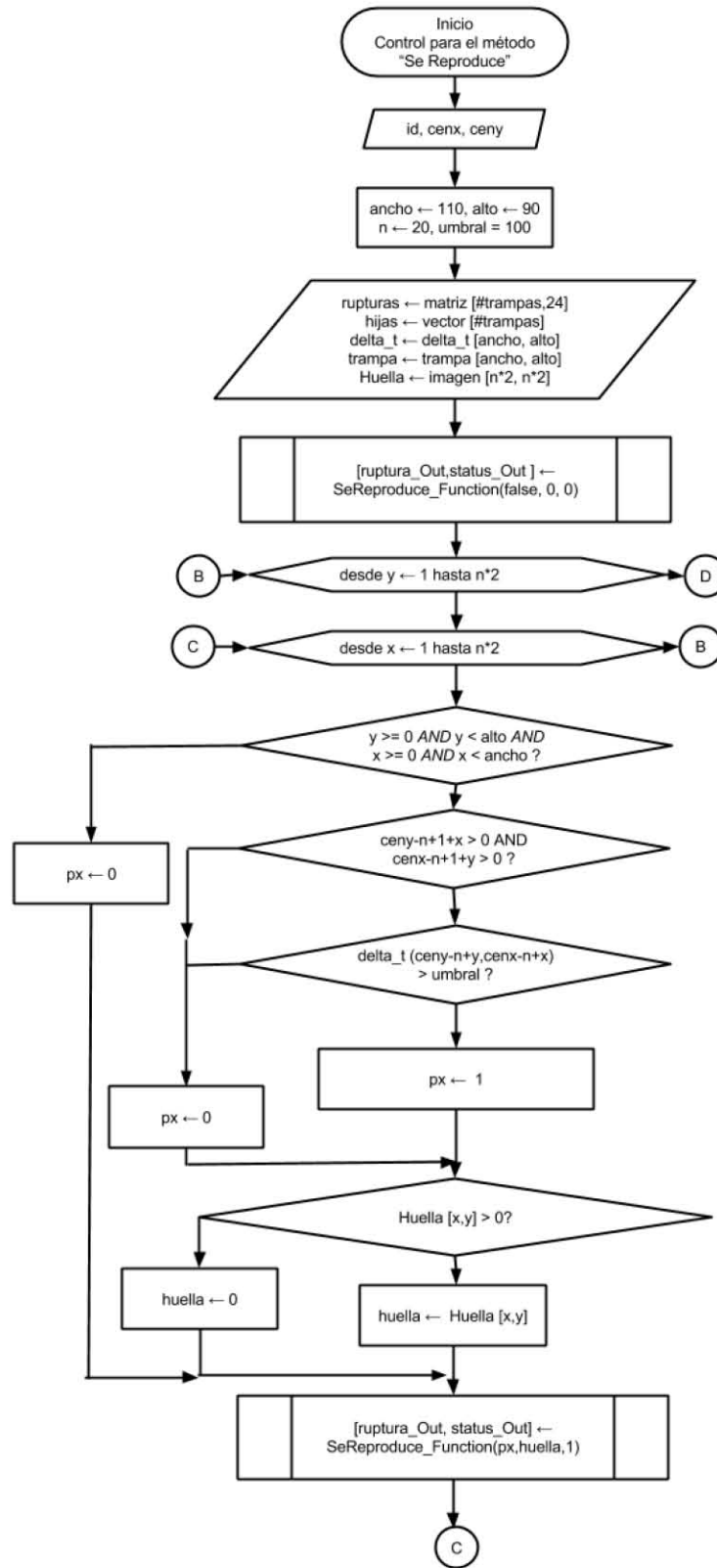
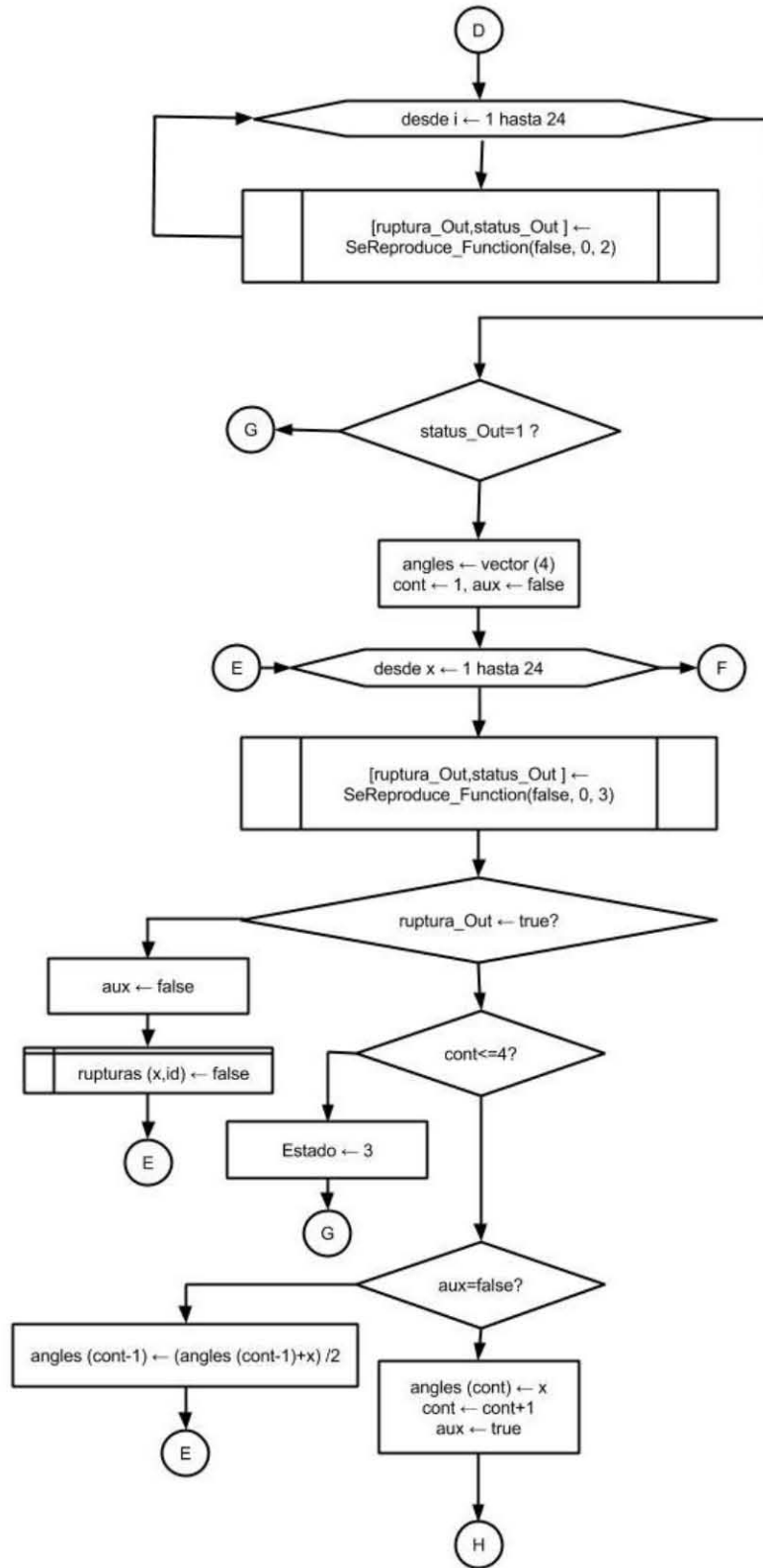


Diagrama 17. Método “Se Reproduce”, donde se muestra que dependiendo de la variable *indice*, que funciona como selector, se ejecuta alguna de las rutinas secundarias dentro del módulo.

Teniendo la función se procedió a hacer el *testbench* para interpretar las salidas y generar nuevas entradas para el algoritmo. Recordemos que esta lógica resume las acciones de control que se deben tomar una vez interpretados los resultados. En el Diagrama 18 se muestra el algoritmo para controlar el proceso con el fin de saber si la célula se está reproduciendo a través de la determinación de la ruptura de sus bordes:





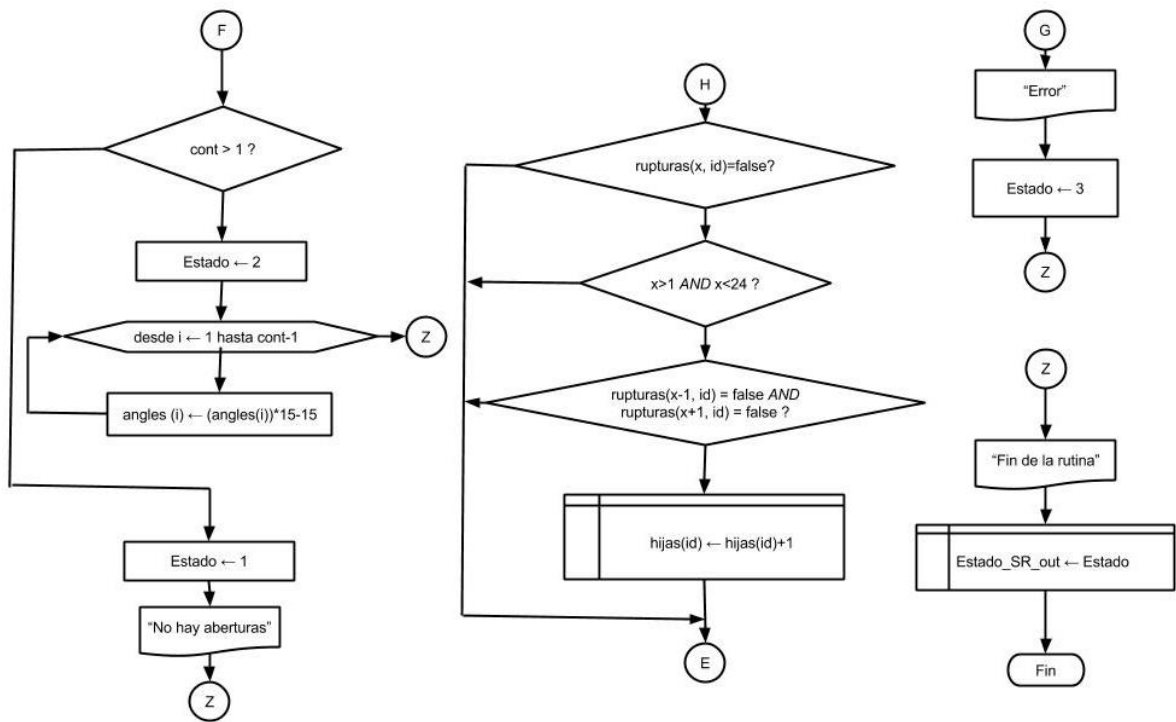


Diagrama 18. Lógica del testbench, o algoritmo de control para el método “Se Reproduce”.

De este proceso de control de variables de entrada o *testbench* al final los datos de interés son el estatus final del análisis del módulo y los valores en grados de las aberturas del borde de la célula. En la Tabla 13 se muestra la interpretación de estas salidas.

Datos de salida del análisis “¿Se Reproduce?”		
Estatus	Ángulos de ruptura	Interpretación
0	~	No ha inicializado el análisis
1	~	No hay aberturas
2	Matriz de ángulos	Hay aberturas
3	~	Error

Tabla 13. Variables de salida del método “Se Reproduce” y su interpretación. El símbolo ~ indica que el valor entregado en ese puerto para ese caso es indiferente para el análisis.

Una vez teniendo la interpretación de resultados y el algoritmo de control para el monitoreo de centros, se procedió a escoger las imágenes representativas de los diferentes casos en los que puede caer el procesamiento:

Caso 1. Una ruptura en el borde de la célula: para este caso se utilizaron los datos de la trampa 34, captura 5.

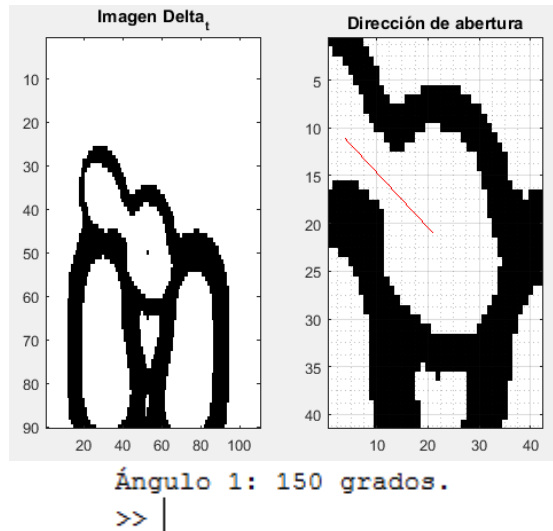


Figura 42. En la imagen de la izquierda se observa la célula madre identificada con un pixel auxiliar en el centro calculado, a la derecha el análisis de la abertura con una línea roja auxiliar indicando la dirección calculada y debajo de estas imágenes vemos la salida de la consola de MATLAB con los datos numéricos arrojados.

Caso 2. No hay rupturas: para este caso se utilizaron los datos de la trampa 98, captura 5.

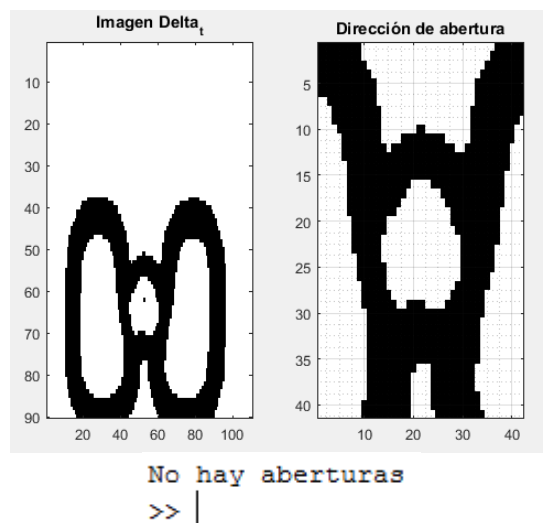


Figura 43. En la imagen de la izquierda se observa la célula madre identificada con un pixel auxiliar en el centro calculado, a la derecha el análisis que no detecta aberturas, lo cual es corroborado con la salida de la consola de MATLAB en la imagen inferior.

Caso 3. Varias rupturas en el borde de la célula : para este caso se utilizaron los datos de la trampa 34, captura 47.

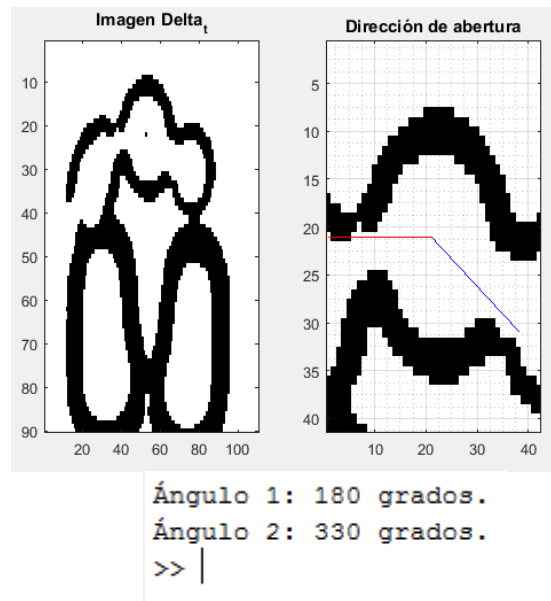


Figura 44. En la imagen de la izquierda se observa la célula madre identificada con un pixel auxiliar en el centro calculado por el proceso de monitoreo, a la derecha el análisis de detección de aberturas, que al detectar dos rupturas de borde marca sus direcciones con las líneas que parten del centro hacia los ángulos calculados. Lo anterior es corroborado con la salida de la consola de MATLAB mostrada en la parte inferior de las imágenes.

Una vez que validamos que tanto el testbench como la función que determina las rupturas en el borde de la estructura celular generan los resultados esperados en los diferentes casos posibles procedimos a generar su código en VHDL con la herramienta HDL Coder. Se sintetizó en el software ISE versión 14.7 de Xilinx y se generó una arquitectura como la que se muestra en la Imagen 40:

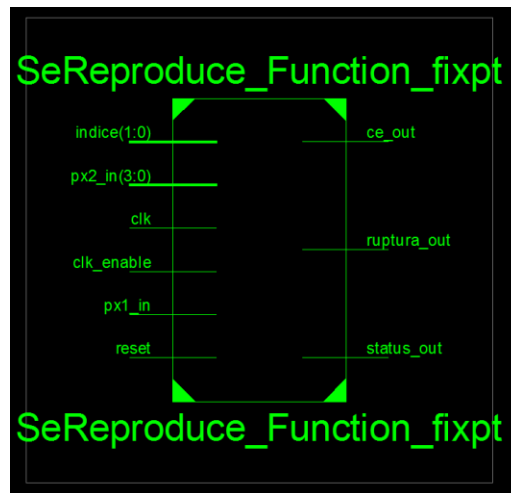
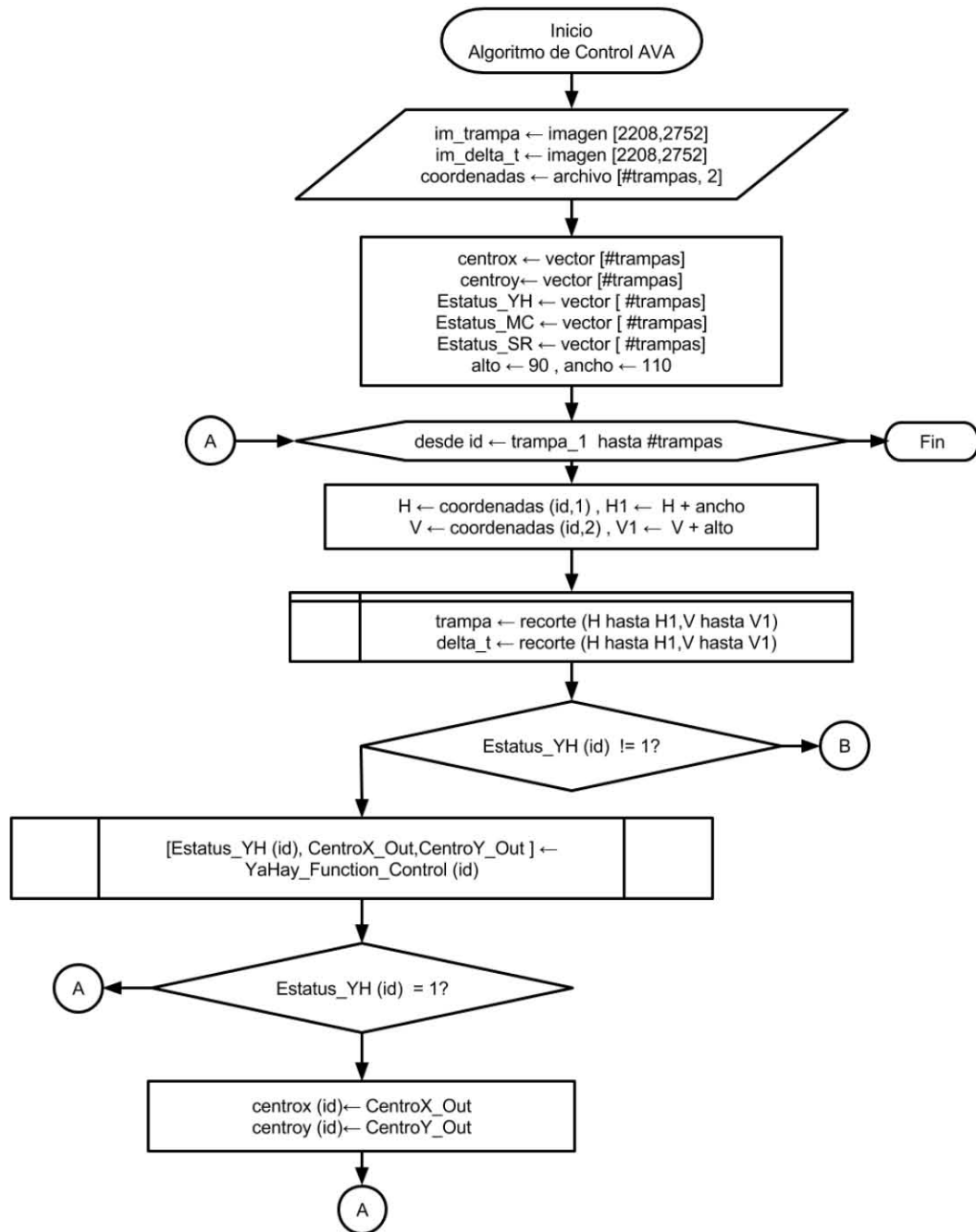


Imagen 40 Arquitectura generada para la función “¿Se Reproduce?”.

Finalmente a continuación se describe la lógica de interacción entre las salidas de los tres métodos: Ya Hay, Monitoreo de Centros y Se Reproduce, lo cual representa el más alto nivel de control del algoritmo,

que debe ser ejecutado por trampa y cuyo resultado final será el contador de las hijas que tuvo la célula a lo largo del número de capturas:



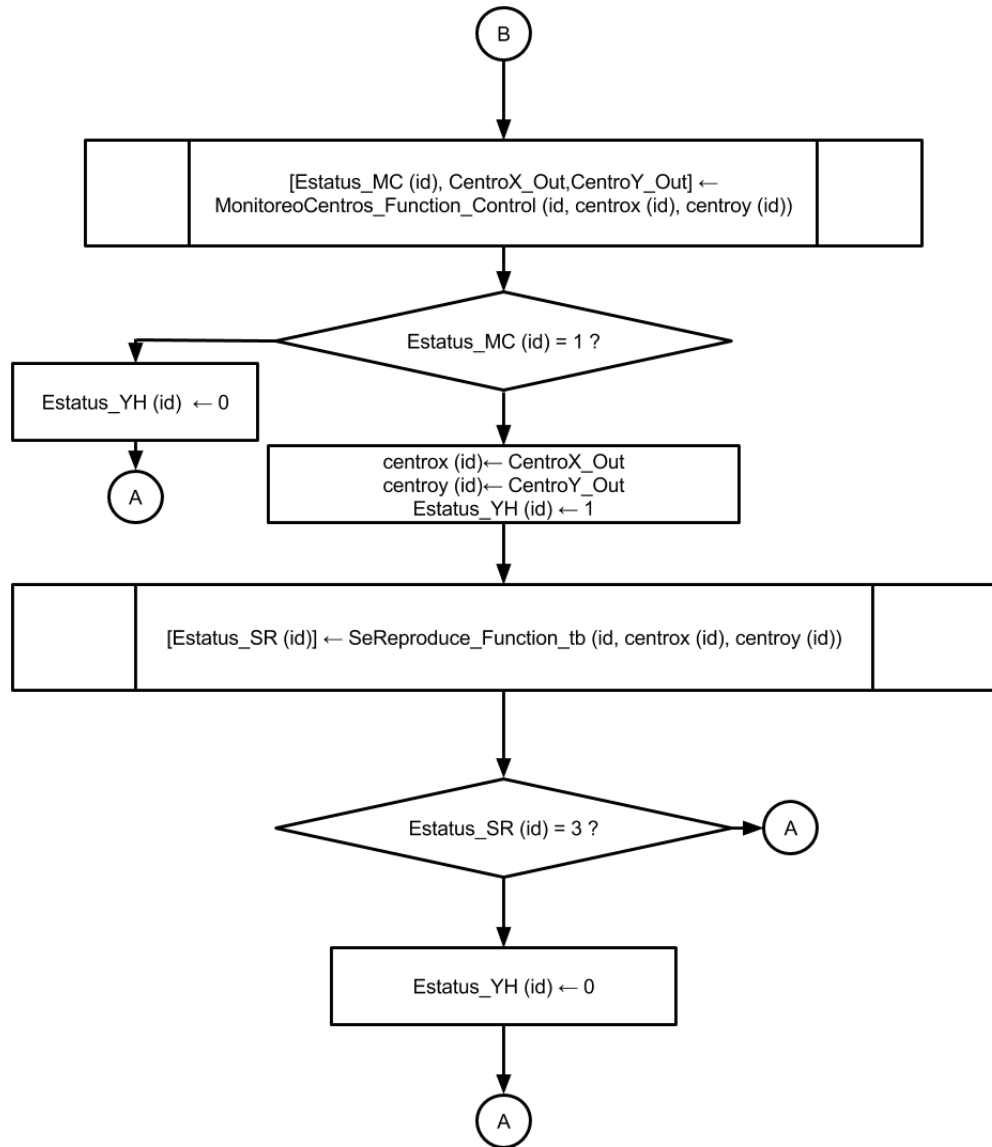


Diagrama 19. Diagrama de control general de procesos para el Sistema de Visión Artificial.

V.2.b. Comprobación del funcionamiento del sistema

Para corroborar que la integración de los módulos del algoritmo y su controlador funcionan correctamente se consideró un intervalo de la secuencia de imágenes del experimento, estas imágenes fueron tomadas por el laboratorio de microfluídica del CINVESTAV-Monterrey. La secuencia original es de 58 tomas, sin embargo en ellas se presenta movimiento de la cámara con respecto a la escena, lo cual es un factor que afecta los resultados del algoritmo. Debido a esto con el fin de comprobar el desempeño se optó por tomar las primeras 10 imágenes para la prueba de funcionamiento general ya que en este intervalo el movimiento relativo de la cámara con respecto a la escena no es significativo.

La prueba consistió en correr el algoritmo y analizar cada dos imágenes (delta_t) los resultados obtenidos del algoritmo por cada una de las trampas y compararlos contra los resultados esperados en ése mismo delta de tiempo con el fin de obtener el total de verdaderos positivos, falsos negativos, falsos positivos y verdaderos negativos, esto es, si en una misma trampa en 5 diferenciales de tiempo, por ejemplo, detecta erróneamente una célula hija, aunque la sumatoria sea sólo una célula hija incorrecta, se tomarán en cuenta 5 falsos positivos, o en su caso, hasta el delta_t correspondiente al ciclo de análisis en el que el algoritmo sea capaz de corregir el resultado. Cabe mencionar que de las 124 trampas a analizar, se tomaron en cuenta sólo 121, debido a que las 3 restantes se encuentran en los bordes de la imagen y no pueden ser procesadas incompletas, por lo que el programa las saca del análisis automáticamente.

A continuación se documentan los resultados:

$$TP \rightarrow 4$$

$$TN \rightarrow 583$$

$$FP \rightarrow 20$$

$$FN \rightarrow 0$$

$$PCC = \frac{4 + 583}{4 + 20 + 583 + 0}$$

$$PCC = 96.7\%$$

De lo cual al hacer las pruebas se observó que los falsos positivos se producen casi en su totalidad en las células que tienen un tamaño más pequeño que la media, por lo que en una segunda propuesta de mejora del presente algoritmo se propone poner especial interés en estos casos.

V.3. Simulación de los módulos de procesamiento en VHDL

V.3.a. Método “¿Ya Hay?”:

Para corroborar que los resultados de la arquitectura son los esperados, se procedió a simular en el ISE de Xilinx el testbench del caso 1, trampa 4 capturas 0 y 2, al mismo tiempo que se comparó su comportamiento con la depuración del mismo caso en MATLAB para la validación de la arquitectura generada.

Bit de Escritura 0: Reset de las variables internas de módulo:

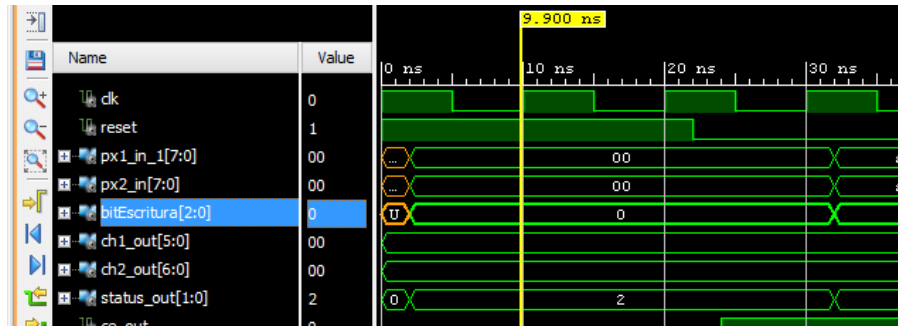


Figura 45. Simulación del ISE de Xilinx.

```

%% Reset de las variables globales
[ch1Out, ch2Out, status] = ...
YaHay_Function(uint8(0), uint8(0), 0);

```

```

K>> [ch1Out, ch2Out, status]
ans =
     0     0     2

```

Figura 46. Observamos el código en MATLAB (izquierda) y los resultados de la depuración en la consola de MATLAB (derecha), lo cual corresponde a lo mismo que vemos en la Simulación del ISE de Xilinx en la Figura 18.

Bit de Escritura 1: Lectura de las imágenes

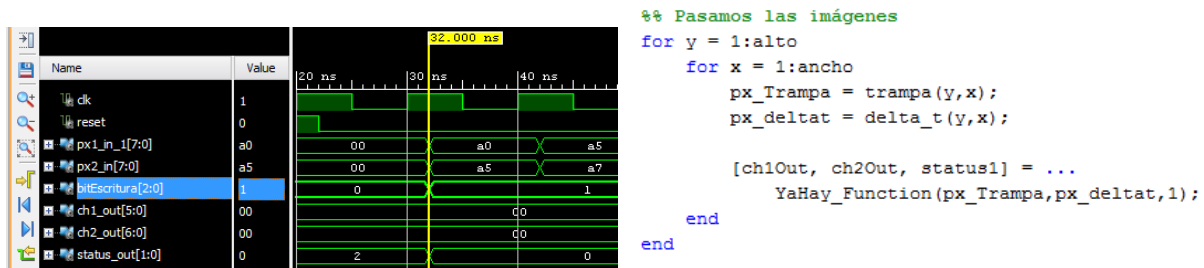


Figura 47. A la izquierda vemos que en la simulación se empiezan a mandar los bits de las imágenes a través de los canales *px1_in* y *px2_in*, lo cual corresponde a lo programado en MATLAB en los ciclos de la derecha.

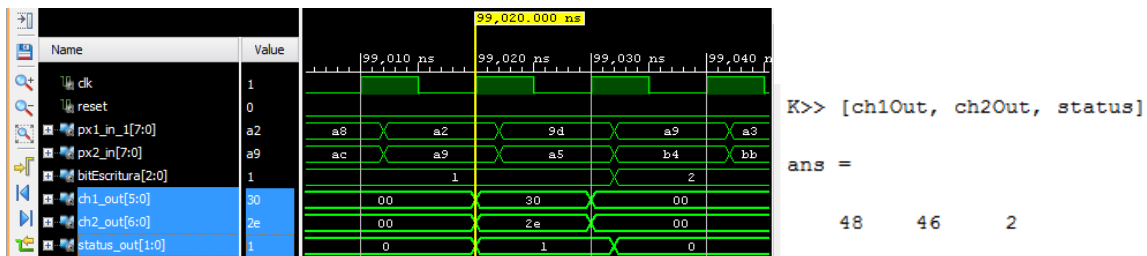


Figura 48. Término de la lectura de imágenes. Observamos que al terminar de enviar las imágenes el bit de status de salida *status_out* se pone en alto y los puertos *ch1_out* y *ch2_out* nos dan los valores en hexadecimal correspondientes a los resultados que nos arroja la depuración en MATLAB. (30 hex = 48 dec, 2e hex=46 dec).

Bit de Escritura 2: Aproximación de los límites de la célula.

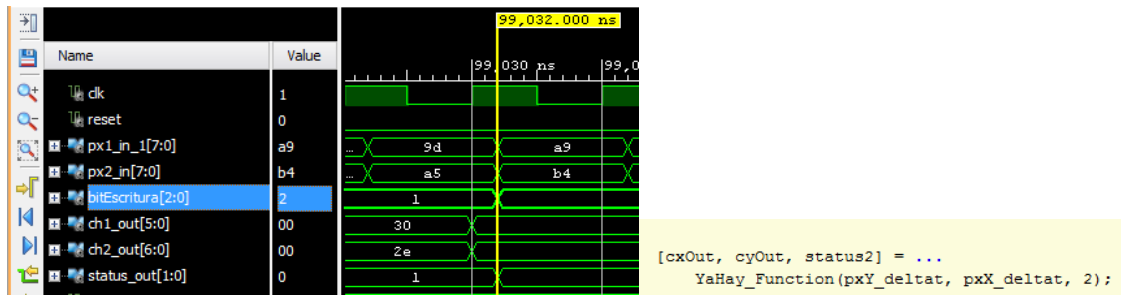


Figura 49. Vemos que durante el procesamiento (función en MATLAB mostrada a la derecha), los valores de los puertos de salida permanecen en cero.

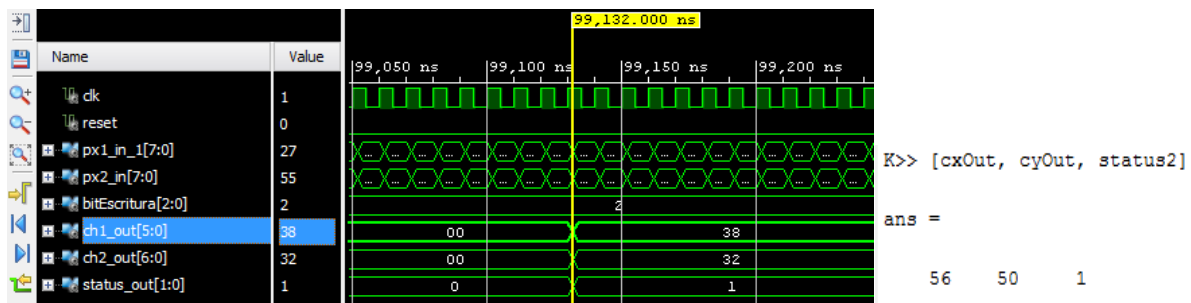


Figura 50. Término del cálculo. Observamos que los valores en hexadecimal de la simulación a la izquierda corresponden a los valores que nos arroja la consola de MATLAB a la derecha.

Bit de Escritura 3: Cálculo aproximado del centro.

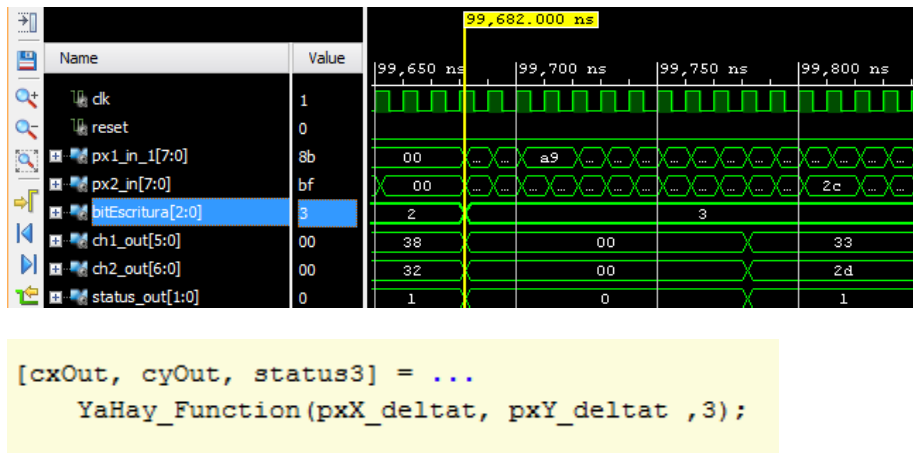


Figura 51. Observamos que los valores de los puertos de salida se mantienen en nivel bajo mientras se hace el procesamiento de cálculo de centros llamado por la función en MATLAB mostrada.

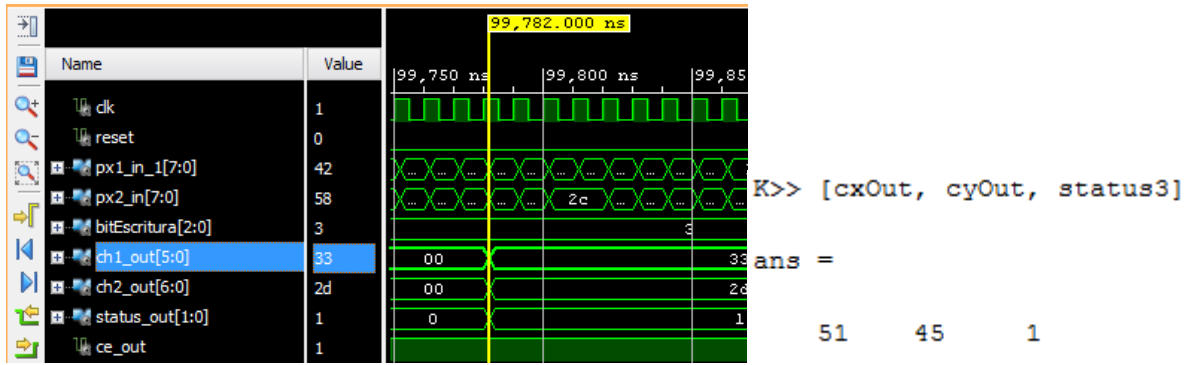


Figura 52. Término del cálculo de la aproximación de centros, vemos cómo el bit de *status_out* cambia a 1 y en los canales *ch1_out* y *ch2_out* se representan en hexadecimal los valores que nos arroja la depuración en MATLAB mostrada a la derecha.

Bit de Escritura 5: Comparación de la imagen contra la huella estadística.

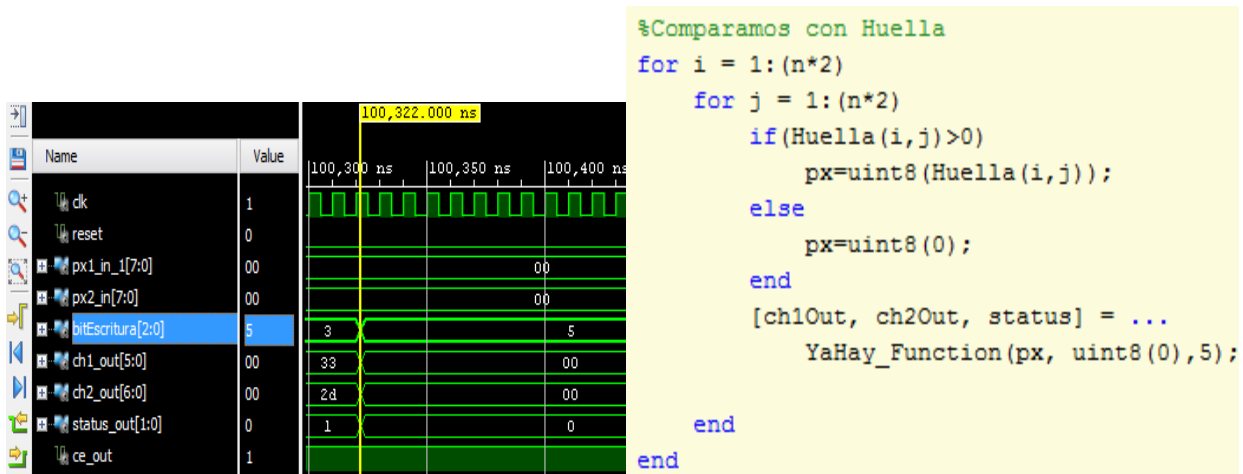


Figura 53. Se observa que durante el procesamiento los canales de salida se mantienen en bajo nivel y por los puertos de entrada *px1_in* y *px2_in* se introducen los valores generados por la función a la derecha.

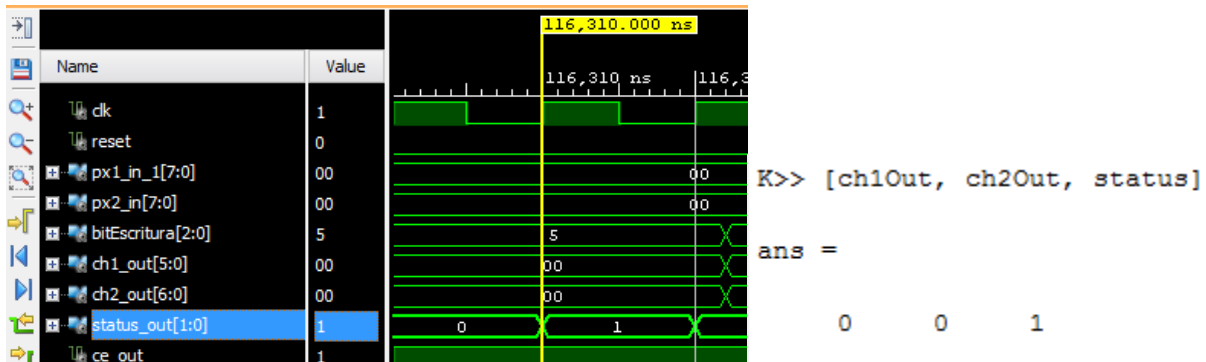
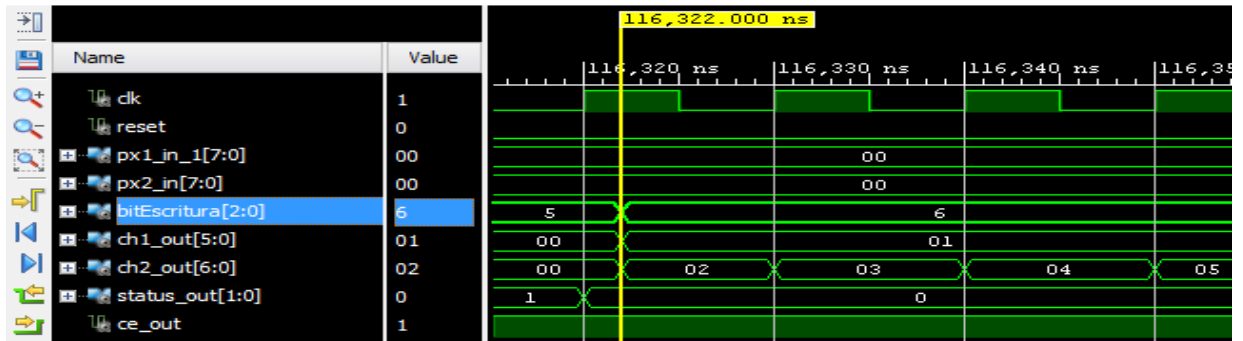


Figura 54 Término de la comparación. El bit de salida *status_out* se pone en alto para indicar que el procesamiento ha terminado y los canales *ch1_out* y *ch2_out* se ponen en cero como se corrobora en la consola de MATLAB mostrada a la derecha.

Bit de Escritura 6: Envío de la imagen de coincidencia



```

%Pedimos coincidencia
for y = 0:(n*2)
    for x = 0:(n*2)

        [ch1Out, ch2Out, status6] = ...
            YaHay_Function(uint8(0), uint8(0), 6);

        if (ch2Out >= 0 && ch2Out < (n*2+1)) && (ch1Out >= 0 && ch1Out < (n*2+1))
            coincidencia(ch2Out+1, ch1Out+1) = status6;
        end
    end
end
end
    
```

Figura 55. En la imagen superior observamos el envío de la imagen de coincidencia a través del puerto de salida *status_out*, y el manejo de los índices de la imagen (coordenadas x e y del pixel enviado) en los puertos de salida *ch1_out* y *ch2_out*, lo cual se recibe y se guarda en la variable *coincidencia*, como se muestra en la función que observamos en la imagen inferior.

Termina el procesamiento

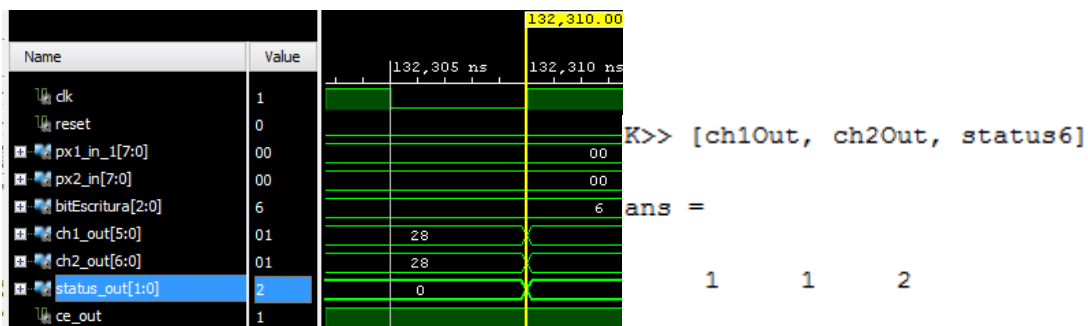
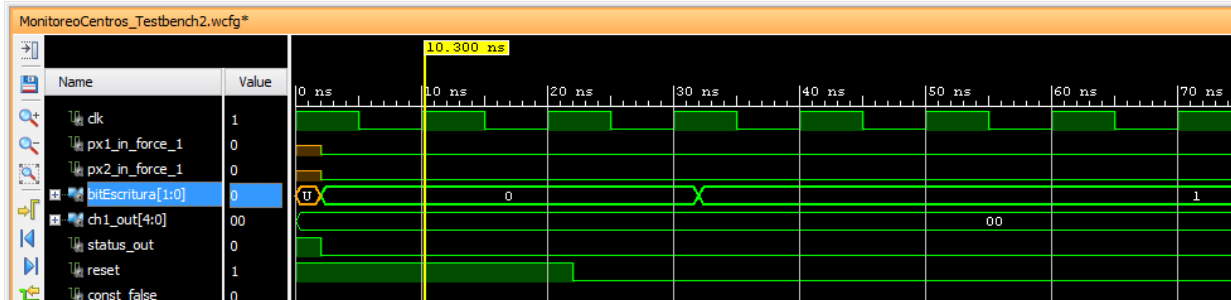


Figura 56. Fin del procesamiento. Se muestran la coincidencia de resultados de los puertos de salida de la simulación y de la depuración en MATLAB.

V.3.b. Método “Monitoreo de Centros”:

Para corroborar que los resultados de la arquitectura son los esperados, se procedió a simular el testbench del caso 1, trampa 94, las capturas 2 y 4. Para validar los resultados fue necesario hacer un *debug* del código en MATLAB para ir corroborando que las salidas por proceso coincidieran con lo que arroja la simulación en el software de Xilinx. Para una mejor visualización de resultados, las simulaciones se hicieron en el software VIVADO 2016.4.

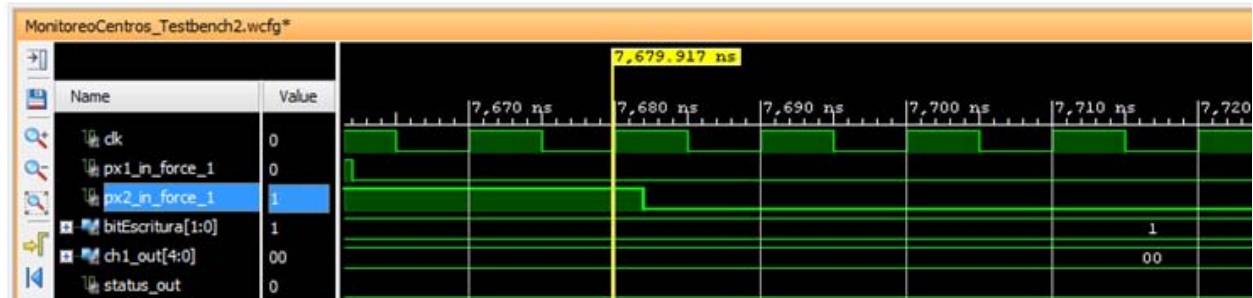
Reset de las variables internas.



```
K>> [ch1Out status]
%% Reset de las variables globales
[ch1Out, status] = ...
    MonitoreoCentros_Function(false, false, 0);
ans =
    0    0
```

Figura 57. En la imagen superior observamos que los valores de los puertos de salida *status_out* y *chanell_out*, cuando el índice del proceso de entrada es 0, corresponden con los valores obtenidos en la depuración del código en MATLAB mostrada en la imagen inferior.

Envío de las imágenes:



```
[cambioOut, status] = ...
    MonitoreoCentros_Function(px1,px2,1);
```

```
K>> [cambioOut status]

ans =

     0     0
```

Figura 58. En la imagen superior observamos que los valores de los puertos de salida *status_out* y *chanell_out*, cuando el índice del proceso de entrada (*bitEscritura*) es 1, corresponden con los valores obtenidos en la depuración del código en MATLAB mostrada en la imagen inferior.

Termina de enviar las imágenes y se despliega el estado:

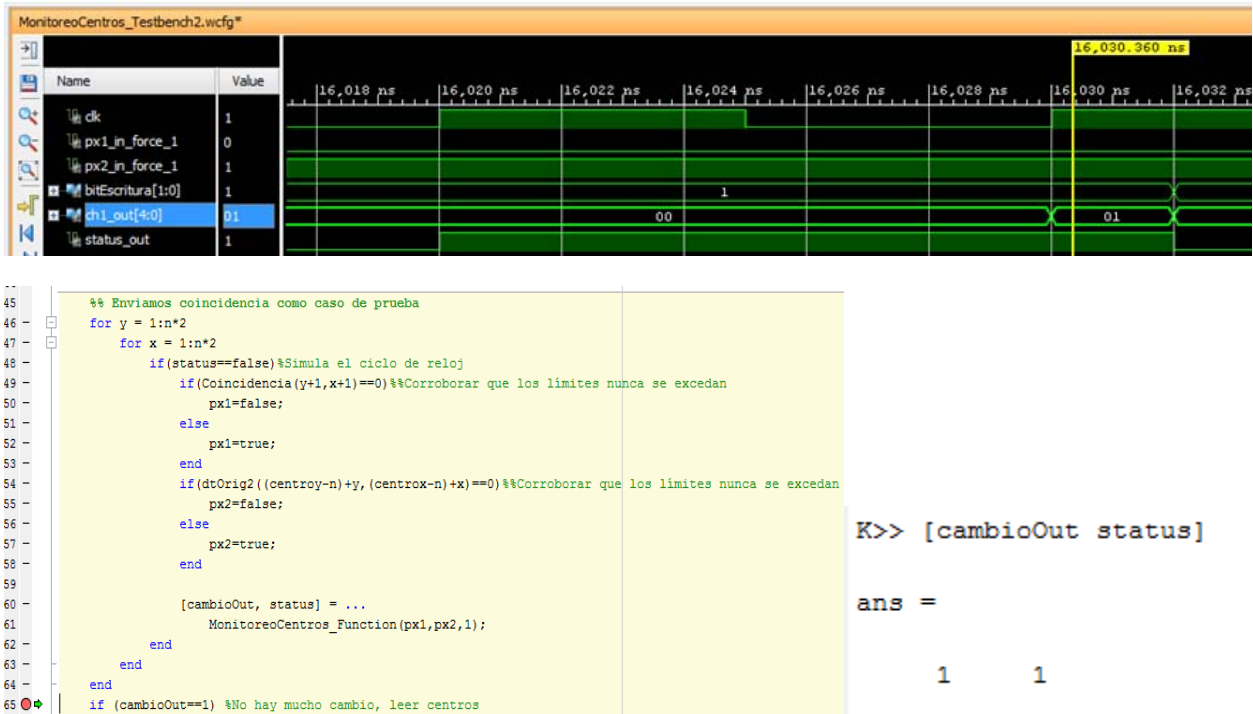
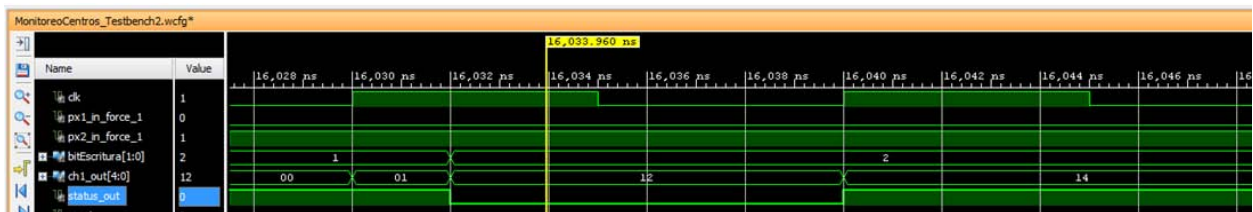


Figura 59. En la imagen superior observamos que los valores de los puertos de salida *status_out* y *chanell_out*, cuando termina el proceso de lectura de imagen, corresponden con los valores obtenidos en la depuración del código en MATLAB mostrada en la imagen inferior.

Envío de centros calculados:



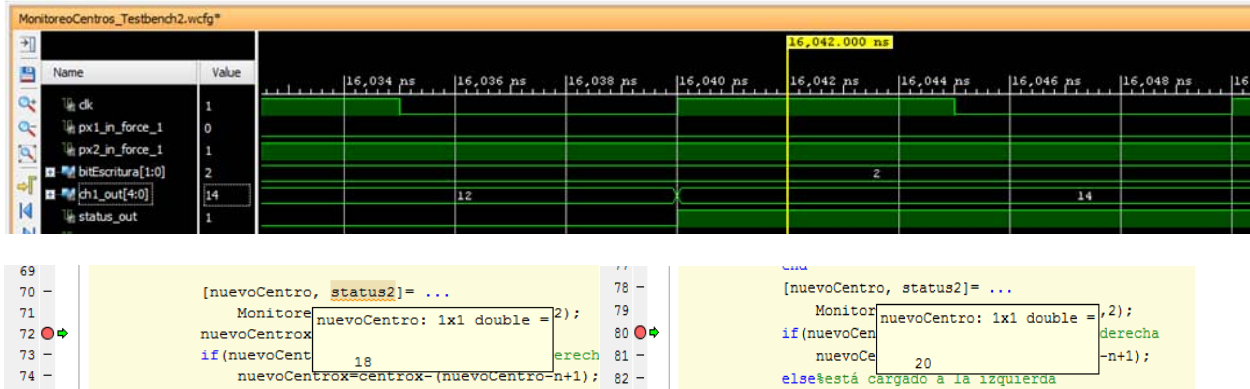


Figura 60. Observamos en las imágenes superiores de la simulación de Xilinx, el valor en hexadecimal que arroja la depuración de MATLAB mostrada en la parte inferior.

V.3.c. Método “¿Se Reproduce?”:

Para corroborar que los resultados de la arquitectura son los esperados, se procedió a simular el testbench de la trampa 34, captura 5, para lo cual también se hizo el *debug* del código en MATLAB para verificar que las salidas correspondieran a los resultados esperados dentro del análisis.

Reset de variables del sistema:

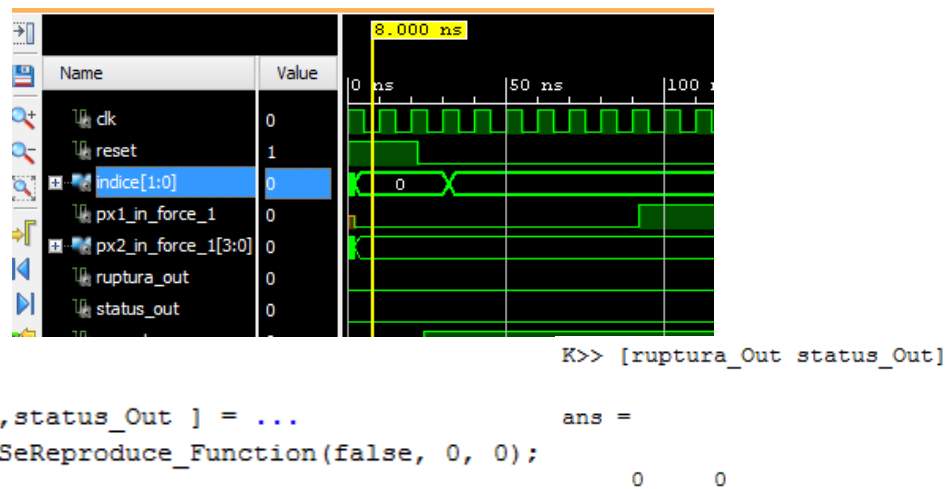
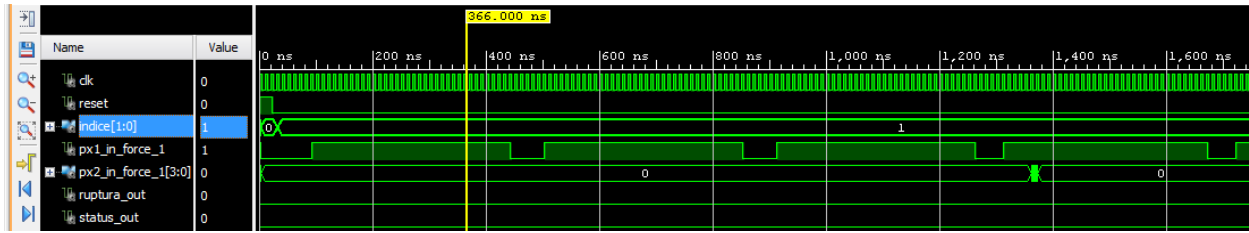


Figura 61. En la imagen superior observamos que los valores de los puertos de salida *status_out* y *ruptura_out*, cuando el índice del proceso de entrada (*indice*) es 0, corresponden con los valores obtenidos en la depuración del código en MATLAB mostrada en la imagen inferior.

Lectura de imágenes

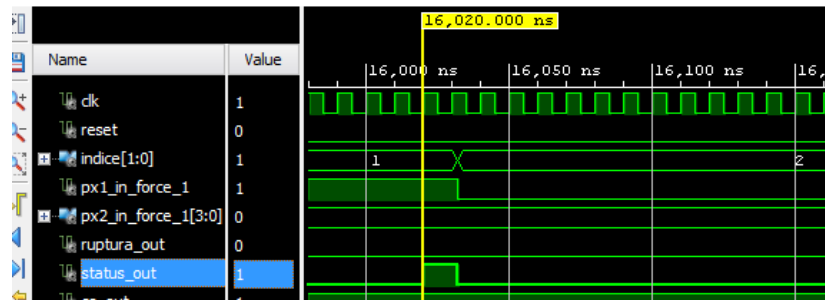


```
[ruptura_Out,status_Out ] = ...
  SeReproduce_Function(px, huella, 1);
```

<pre>58 59 60 61 62 63</pre>	<pre>[ruptura_Out,status_Out] = ... SeReproduce_Function(px, huella, 1); end</pre>	<pre>58 59 60 61 62 63</pre>	<pre>[ruptura_Out,status_Out] = ... SeReproduce_Function(px, huella, 1); end</pre>
------------------------------	---------------------------------------------------------------------------------------	------------------------------	---------------------------------------------------------------------------------------

Figura 62. En la imagen superior observamos que los valores en la simulación de Xilinx de los puertos de salida *status_out* y *ruptura_out*, cuando el selector del proceso de entrada (*indice*) es 1, corresponden con los valores obtenidos en la depuración del código en MATLAB mostrada en la imagen inferior.

Cuando termina de leer las imágenes:



```
[ruptura_Out,status_Out ] = ...
  SeReproduce_Function(px, huella, 1);
end
end
```

Figura 63. En la imagen superior observamos que el valor de *status_out*, cuando termina de leer las imágenes (índice es igual a 1), corresponden con los valores obtenidos en la depuración del código en MATLAB mostrada en la imagen inferior.

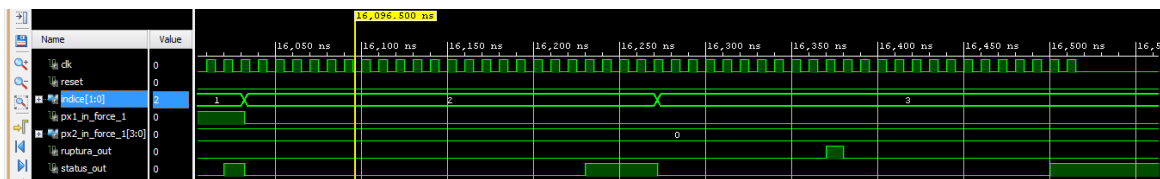


Figura 64. Bit selector (*indice*) es igual a 2 y su transición al método 3.

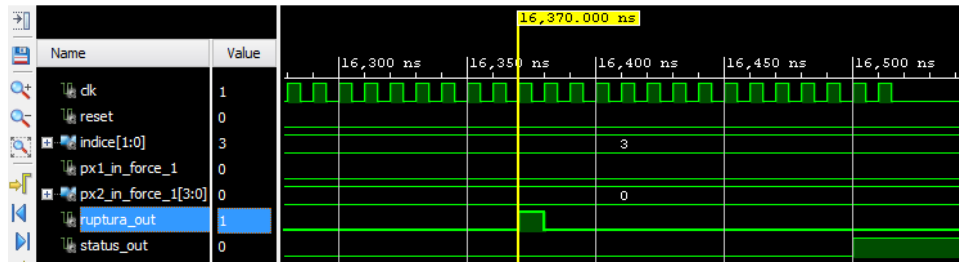


Figura 65. Bit de salida *ruptura_out* igual a 1 y término del procesamiento.

V.4. Definición de requerimientos de procesamiento y memoria

En esta sección se discutirán los aspectos más importantes para determinar las características del microprocesador a utilizar en la implementación de los algoritmos de control de procesos del sistema de acuerdo a la arquitectura del sistema propuesta en el punto V.1 así como a la capacidad de memoria necesaria para alojar las imágenes y los datos generados de su análisis.

Como primer punto se propone tener un microprocesador embebido con su respectiva unidad de gestión de memoria, memoria local de acceso rápido para las variables de procesamiento, memoria externa para el almacenamiento de las imágenes, la conexión con un bus de periféricos necesario para el control de los módulos de procesamiento de imagen y un bus SPI (Serial Peripheral Interface) que es el estándar de comunicaciones que actualmente se tiene como base de la interacción con la computadora de a bordo, la cual, como podemos recordar, es la responsable del control de todos los procesos de la plataforma satelital incluido el Sistema de Visión Artificial para el análisis de la carga útil.

Los requerimientos de esta arquitectura que dependen directamente de las características del sistema de visión artificial se pueden reducir en dos: las terminales necesarias en el bus de periféricos y la capacidad de memoria interna y externa demandada, para lo que se tomó en cuenta lo siguiente:

- El número de terminales de los puertos de entrada y salida definidos por los módulos de procesamiento de imagen.
- Una memoria principal con capacidad para alojar a las variables de procesamiento y a los archivos del código.
- Una memoria secundaria para almacenar las imágenes y datos que resulten del análisis.

A partir de esto se definen los puertos necesarios en el microprocesador como se muestran en la Tabla 14.

I/O	Puerto	Tamaño (bits)			
O	clk	1			
O	reset	1			
O	indice	2			
O	px2_in	4			Puertos SeReproduce
O	clk_enable	1			Puertos YaHay
O	px1_in	1			Puertos MonitoreoCentros
I	ruptura_out	1			
I	status_out	1			
O	bitEscritura	3	Puertos de entrada = 23		
O	px1_in	8	Puertos de salida = 45		
O	px2_in	8	TOTAL = 68		
O	clk_enable	1			
I	ch1_out	6			
I	ch2_out	7			
I	status_out	2			
O	bitEscritura	2			
O	clk_enable	1			
O	px1_in	1			
O	px2_in	1			
I	ch1_out	5			
I	status_out	1			

Tabla 14. Relación de puertos de entrada y salida requeridos para la comunicación con los módulos de procesamiento de imagen, donde la primera columna nos indica si es un puerto de entrada o salida (Input/Output), en la segunda columna a qué señal corresponde el puerto y en la tercera su tamaño en bits.

La definición del tamaño del vector de datos por puerto como se observa en la tercera columna de la Tabla 14 está expresada en bits, lo que puede traducirse como el número de terminales necesarias en el bus de periféricos del microprocesador. El dejarlo expresado en bits tiene la finalidad de dar cabida al diseño de otros posibles módulos de comunicación si en un trabajo futuro se quisiera reducir este requerimiento de puertos para utilizar un microcontrolador específico.

Para hacer el cálculo de la memoria interna necesaria en el microprocesador, se midió el espacio de almacenamiento que ocupa el programa de control del algoritmo y se determinaron las variables de procesamiento que resultaran factibles de ser almacenadas internamente la cual se muestra en la Tabla 15.

#trampas = 124						
Tamaño del vector	ancho	alto	#bits		Total (bits)	MB
Variable						
CentroX	1	124	7		868	0.000109
CentroY	1	124	6		744	0.000093
Estado YH	1	124	2		248	0.000031
EstadoMC	1	124	2		248	0.000031
EstadoSR	1	124	2		248	0.000031
Programas					341600	0.0427
				TOTAL =	343956	0.042995 MB

Tabla 15. Cálculo de requerimientos de memoria interna en Megabytes.

Los requerimientos de capacidad de la memoria externa fueron calculados de acuerdo al espacio que ocupan las imágenes y variables mayores a 8 bits que no están comprometidas con el procesamiento, lo cual puede verse desglosado en la Tabla 16. La velocidad a la que necesitamos leer y escribir en ella está dada por la velocidad de reloj que requiere el microprocesador para realizar la tarea, lo que analizaremos más adelante.

#trampas = 124						
Tamaño del vector	ancho	alto	#bits			
Variable					Total (bits)	MB
Imagen Trampa	2752	2208	8		48611328	6.076416
Imagen dt	2752	2208	8		48611328	6.076416
Coincidencia	40	40	1		1600	0.0002
Rupturas	24	124	1		2976	0.000372
Huella	40	40	6		9600	0.0012
Contador Hijas	1	124	6		744	0.000093
Coordenadas	2	124	7		1736	0.000217
trampa (recorte)	110	90	8		79200	0.0099
dt (recorte)	110	90	8		79200	0.0099
				TOTAL =	97397712	12.17471 MB

Tabla 16. Cálculo de requerimientos de memoria externa en Megabytes.

La arquitectura con la propuesta de los módulos para su integración y con la definición de los periféricos de entrada y salida desglosada se muestra en la Figura 66:

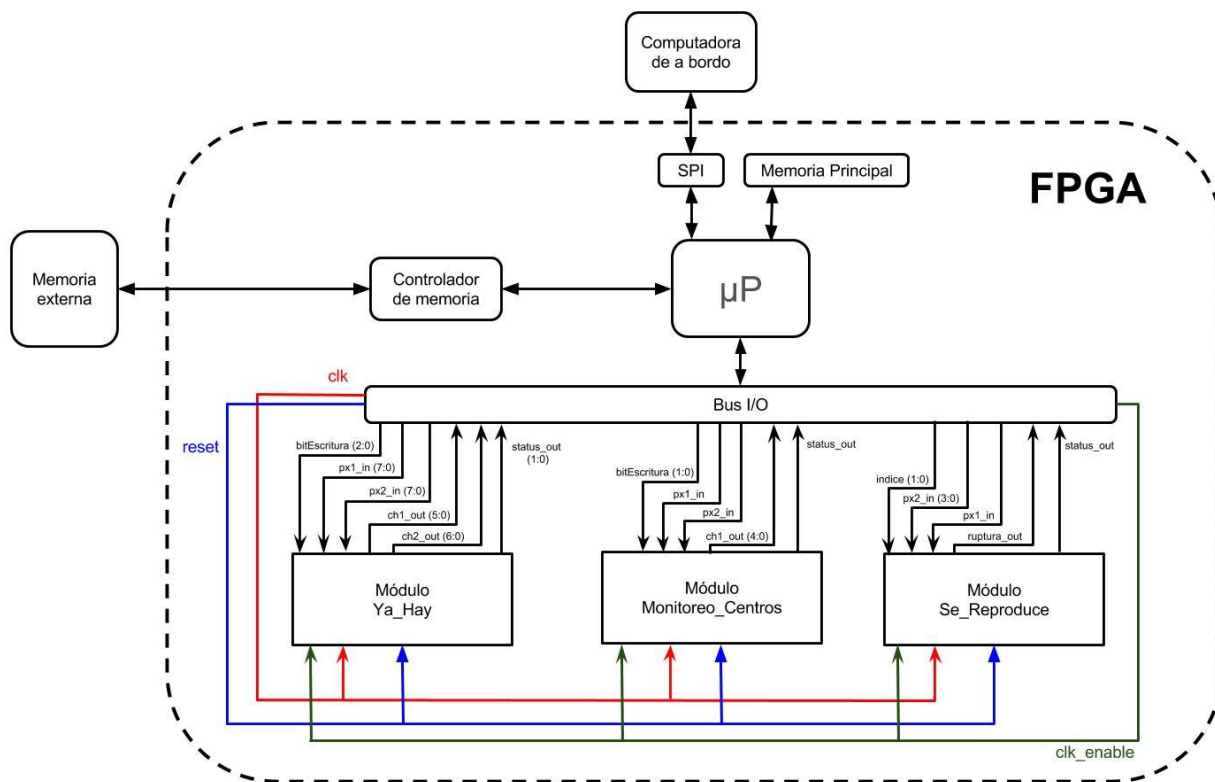


Figura 66. Arquitectura del sistema, donde se propone la implementación de un microprocesador embebido tomando en cuenta la integración que se requiere con una memoria externa y con la computadora de a bordo. Se muestran de forma desglosada los puertos necesarios para la comunicación con los módulos de procesamiento de imagen, donde los índices entre paréntesis indican el tamaño de vector (2:0 indicaría por ejemplo un vector de 3 bits); cuando se omite dicho índice significa que ese puerto consta de sólo un bit. Las flechas indican la dirección del flujo de las señales.

Para determinar la frecuencia de la señal de reloj para el procesamiento de datos del Sistema de visión tomaremos como referencia los resultados arrojados por la simulación de cada módulo, en donde se utilizó una señal de 10 ns. En la Tabla 17 se muestra el tiempo en nanosegundos calculado por proceso, y el equivalente en ciclos de reloj.

Proceso:	Ya Hay	Monitoreo	Se reproduce	Total
Tiempo [ns]:	132310	16040	16370	164720
	10 ns = 1 ciclo de clk			
Total de ciclos de reloj =	16472			

Tabla 17. Tiempo por proceso calculado en la simulación del sistema con el software ISE de Xilinx.

Tomando en cuenta que cada imagen debe ser analizada cada 10 minutos, para un escenario de 125 trampas se hace el cálculo siguiente:

$$\frac{16472 \frac{\text{ciclos}}{\text{trampa}} * 125 \text{ trampas}}{10 \text{ min} * \frac{60 \text{ s}}{\text{min}}} = 3431.67 \frac{\text{ciclos}}{\text{s}}$$

Proponemos un factor de seguridad de 1.5, considerando que el análisis dependiendo de las características de la imagen o del caso en el que se encuentre la célula puede variar un poco en el procesamiento requerido, con lo que obtenemos un requerimiento de frecuencia mínima de 6 kHz.

$$3431.67 \frac{\text{ciclos}}{\text{s}} * 1.5 = 5147.5 \text{ Hz} \sim 6 \text{ kHz}$$

Los algoritmos de control de los módulos de procesamiento de imagen se encuentran en lenguaje M (propio del software MATLAB), por lo que se adjuntan los códigos y sus diagramas de flujo por si se necesitara su migración hacia cualquier otro lenguaje de programación.

Por último, para hacer la implementación del sistema es necesario conocer los recursos en hardware que demandan los módulos de procesamiento de imagen, a continuación en la Tabla 18 se muestra la cantidad de LUTs (Lookup tables) y registros necesarios, lo cuales fueron estimados por el software ISE versión 14.7 de Xilinx al sintetizar el código de cada módulo.

Módulo	Registros	LUTs
Ya Hay	11725	22501
Monitoreo de Centros	61	2439
Se Reproduce	1731	25092
Total	13517	50032

Tabla 18. Estimación de los requerimientos totales de hardware demandados por los módulos de procesamiento de imagen.

Después de definir estos requerimientos se prosiguió a hacer un análisis de las tarjetas FPGA de la familia Xilinx que cumplen los requisitos mínimos para albergar los módulos de procesamiento de imagen, lo cual se muestra en la Tabla 19.

Familia de FPGA	Modelo	Slices	LUTs	%Usado por AVA	Registros	%Usado por AVA
Spartan 7	XC7S100	16000	64000	78.18%	128000	10.56%
Artix 7	XC7A100T	15850	63400	78.91%	126800	10.66%
Artix 7	XC7A200T	33650	134600	37.17%	269200	5.02%
Kintex 7	XC7K160T	25350	101400	49.34%	202800	6.67%
Kintex 7	XC7K325T	50950	203800	24.55%	407600	3.32%
Kintex 7	XC7K355T	55650	222600	22.48%	445200	3.04%
Kintex 7	XC7K410T	63550	254200	19.68%	508400	2.66%
Kintex 7	XC7K420T	65150	260600	19.20%	521200	2.59%
Kintex 7	XC7K480T	74650	298600	16.76%	597200	2.26%
Virtex 7	XC7VX330	51000	204000	24.53%	408000	3.31%

Tabla 19. FPGAs de la familia Xilinx que cumplen los requisitos para albergar los módulos de procesamiento de imagen. En la primera y segunda columna se muestran la familia y modelos de FPGAs que cumplen los requerimientos mínimos para contener los tres módulos del sistema de visión. En la tercera columna se indica el número de slices especificados por el fabricante, cada slice contenida en un FPGA de la serie 7 de Xilinx contiene 4 LUTs y 8 flip-flops [25](registros), de ahí se tomó el cálculo del porcentaje mostrado en las columnas 5 y 7 de la estimación de uso por parte de los módulos del algoritmo de visión artificial (AVA).

La lista mostrada en la Tabla 19 es sólo un referente para la elección de tarjeta. Se han incluido únicamente las tarjetas de la serie más nueva de Xilinx ya que seg su diseño está orientado a optimizar el consumo energético y sólo se dejó indicado también el FPGA más reducido de la familia Virtex 7 puesto que a pesar de ser el de recursos más limitados dentro de su clasificación, sólo se aprovecharía un 25% de los recursos disponibles en este FPGA. No se han incluido otras marcas de FPGAs aun cuando pudieran cumplir las necesidades del sistema, pues de acuerdo con el artículo “Mitigation of Radiation Effects in SRAM-based FPGAs for Space Applications”[16], científicos de la NASA han encontrado que los FPGAs de Xilinx son los más adecuados para hacer tareas de procesamiento de alto desempeño a bordo debido a su flexibilidad comparado con otras tarjetas o procesadores.

En el Laboratorio de Instrumentación de Sistemas Espaciales de la Facultad de Ingeniería de la UNAM hay antecedentes del uso de microprocesadores embebidos dentro de FPGAs para uso espacial. Tomando como referencia el trabajo “Diseño de un sistema de comando, manejo de información y telecomunicaciones para un microsatélite de percepción remota”[24] donde el sistema que contiene al microprocesador, módulos de control de puertos de entrada y salida, memoria y controlador de memoria ocupa menos del 25% de los recursos del FPGA Artix 7 modelo XC7A200T, si a esto le sumamos el 37.17% que demandan los módulos de procesamiento de imagen en esa misma tarjeta (ver Tabla 19), estaríamos hablando de un aprovechamiento aproximado del 62.17% de los recursos lógicos, cuyo restante representa un buen factor de seguridad para atrevernos a proponer esta tarjeta para la implementación del Sistema de Visión Artificial, ya que en caso de que las características del microprocesador específico para esta aplicación sobrepasen las del diseño descrito en el trabajo citado, se tiene casi un 38% de los recursos de la tarjeta para compensar estos cambios, y en el caso contrario,

donde las posibles diferencias disminuyesen el uso de recursos estimados, éstos podrían ser aprovechados en la aplicación de técnicas de tolerancia a fallas.

Con lo anterior se tienen definidos todos los elementos necesarios para continuar con la implementación del sistema en hardware.

VI. CONCLUSIONES Y TRABAJO A FUTURO

Conclusiones:

Durante el desarrollo del proyecto se analizaron diferentes técnicas para hacer el análisis de imagen y se determinó que la más adecuada a las necesidades de la misión fue el uso de sustracción de fondo sobre imágenes binarias de acuerdo a los resultados de desempeño obtenidos (por arriba del 85% de aciertos en la detección correcta de células en las trampas) y su viabilidad de implementación en hardware reconfigurable al utilizar imágenes binarias y tener requerimientos de procesamiento viables para los recursos disponibles a bordo. Se diseñó una arquitectura a la medida y se describieron los elementos que la componen con el fin de generar un sistema capaz de hacer un análisis autónomo de la carga útil de la misión MICROLAB utilizando técnicas de visión artificial. A continuación se presentan a modo de lista los objetivos particulares del proyecto y una breve descripción de las conclusiones para cada uno:

- Proponer una solución para el análisis y monitoreo del experimento que resulte más adecuada en dimensiones y consumo de energía que las reportadas en el estado de arte.

Este objetivo se cumplió en cuanto a que las soluciones existentes para experimentos similares en el estado del arte no podrían hacer un análisis cuantitativo de las células que en este caso es requerido por la naturaleza del análisis planteado. Se logró tener un requerimiento mínimo de un reloj de 6kHz para el sistema de procesamiento de imagen, lo que representaría una demanda muy baja de energía y el uso de un solo sensor de imagen (comparado con el estado de arte donde se usa un sensor por pozo a analizar) para hacer el análisis de todo el dispositivo microfluídico, por lo que concluimos que la solución propuesta en esta tesis resulta adecuada para su aplicación espacial.

- Diseñar una arquitectura de procesamiento de alto desempeño en FPGA, la cual cumpla los requerimientos del sistema de visión.

Se logró hacer el diseño de la arquitectura del sistema de visión para su implementación en FPGA, generando los requerimientos y especificaciones detalladas de cada elemento que lo compone, todo esto tomando en cuenta las limitaciones que representa generar un sistema embebido y las características de la plataforma satelital.

- Diseñar e implementar los algoritmos de reconocimiento, detección, evaluación, seguimiento y estimación del sistema de visión artificial.

Se realizó el diseño para las necesidades específicas de solución del sistema, por lo que se generaron tres módulos de control del algoritmo y tres más para el análisis de la imagen diseñados para ser implementados en hardware reconfigurable y para disminuir la demanda de procesamiento en el microcontrolador. Cada módulo de análisis de imagen representa una arquitectura a la medida. Con esto

se logró desempeñar las tareas descritas en este objetivo y se validó su funcionamiento comparando los resultados arrojados por la simulación del software ISE de Xilinx contra los de su implementación en MATLAB.

- Lograr un análisis cuantitativo de los organismos en estudio.

Para lograr este objetivo se propusieron diferentes soluciones resultando la más adecuada el análisis por técnicas de substracción de fondo, con lo que se obtuvieron resultados correctos del análisis de visión artificial por arriba del 85% de los casos comparado con los resultados analizados por un humano. Este proceso requirió un trabajo de investigación, un análisis meticuloso del problema a resolver y el planteamiento de diferentes propuestas de solución, con lo que hubo un proceso creativo importante que considero un conocimiento clave en mi formación y en mi futura actividad profesional como Ingeniera. Se logró generar un contador de células hijas por cada trampa a analizar, lo que constituye una innovación en el área.

- Diseñar el algoritmo de control de procesos a partir de la interpretación de las imágenes tomadas a las levaduras.

Este objetivo se logró a partir de la definición de cómo debían interactuar los diferentes procesos que intervienen en el análisis de cada trampa, con el que se obtuvieron los resultados esperados.

Con lo anterior se logró comprobar la hipótesis, pues fue posible hacer el diseño de un sistema autónomo para el estudio de la longevidad de células eucariotas utilizando técnicas de visión artificial. El resultado fue un sistema a la medida que a través del análisis efectuado por tres bloques de procesamiento de imagen es capaz de interpretar y monitorear el estado de las células bajo estudio, lo cual conformará un sistema para experimentación biológica a bordo de un nanosatélite único en su tipo.

Trabajo a futuro

El presente proyecto representa una base de la que aún se pueden generar diferentes proyectos, siendo las principales tareas:

- Implementación en el FPGA del microprocesador requerido para alojar al algoritmo de control de los módulos de procesamiento de imagen.
- Diseño e implementación de los módulos necesarios para la comunicación del microprocesador con los módulos de procesamiento de imagen y con una memoria externa que cumplan con los requerimientos planteados en la presente tesis.
- Diseño y construcción del sistema óptico necesario para la toma de imagen con los requerimientos planteados en la presente tesis.
- Diseño e implementación de técnicas de tolerancia a fallas a todo el sistema.

Así mismo uno de los principales problemas que se tienen al analizar una imagen por técnicas de sustracción de fondo, es que son muy susceptibles a los movimientos, y tomando en cuenta que el sistema va a estar sujeto a vibraciones que podrían alterar la posición del sensor con respecto al objeto a analizar, se recomienda la generación de una marca de referencia en el dispositivo microflúidico (objeto a analizar) para facilitar la localización y eliminación de los píxeles pertenecientes a las trampas que requiere el análisis.

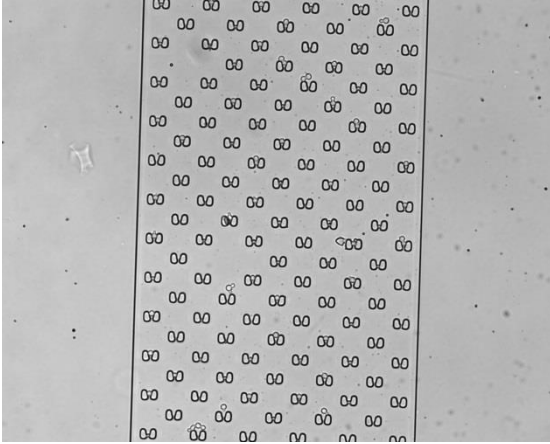
VII. REFERENCIAS

- [1] Medina Francisco Javier, “Alteraciones en funciones celulares esenciales observadas en semillas germinadas en la Estación Espacial Internacional”, fragmento de Plantas en el espacio, pág.79, España 2010.
- [2] Can Jo Myeong, Liu Wei, Gu Liang, “High-throughput analysis of yeast replicative aging using a microfluidic system”.
- [3] Ricco Antonio, Hine John, Piccini M, “Autonomous Genetic Analysis System to study Space Effects on Microorganisms: Results from Orbit”, The 14th International Conference on Solid-State Sensors, Actuators and Microsystems, Lyon France, Junio 2007.
- [4] García Cordero José Luis, González Alan, “Imágenes del experimento de la carga útil de la misión MICROLAB”, Marzo 2016.
- [5] Hines John, Piccini Mathew, D Squires David, Ricco Antonio, “Extended Life Flight Results from the GeneSat-1 Biological Microsatellite Mission”, 22nd Annual AIAA/USU Conference on Small Satellites 2015.
- [6] Kitts Christopher, Hines John, Ricco Antonio, “The GeneSat-1 Microsatellite Mission: A Challenge in Small Satellite Design”, 20th Annual AIAA/USU Conference on Small Satellites.
- [7] NASA Ames Research Center, “PharmaSat Nanosatellite”, NASAfacts, 1 de abril de 2009.
- [8] Kitts Christopher, Agasid Elwood, Fredericks Charlie, Piccini Mathew, “Initial Flight results from the PharmaSat Biological Microsatellite Mission”, 23th Annual AIAA/USU Conference on Small Satellites.
- [9] Kitts Christopher, Niesel D, McGinnis M, “Initial Flight Results from the PharmaSat Biological Microsatellite Mission”, presentación en la Universidad de Santa Clara 2010.
- [10] Ricco Tony, “Integrated Microfluidics Systems in Challenging Environments: Biological Studies in Earth Orbit”.
- [11] Kitts Christopher, Ricco Antonio, Stackpole Eric, “Initial On-Orbit Results from the O/OREOS Nanosatellite”, 25th Annual AIAA/USU Conference on Small Satellites.
- [12] Minelli Giovanni, Ricco Antonio, Beasley Christopher, “O/OREOS Nanosatellite: A multi-payload Technology Demonstration”, 24th Annual AIAA/USU Conference on Small Satellites.
- [13] Horst Frank, “Espectro electromagnético”, recurso digital consultado el 27 de abril del 2016, <http://www.zetmaschinisten.com/svg/Sprektrum.svg>.

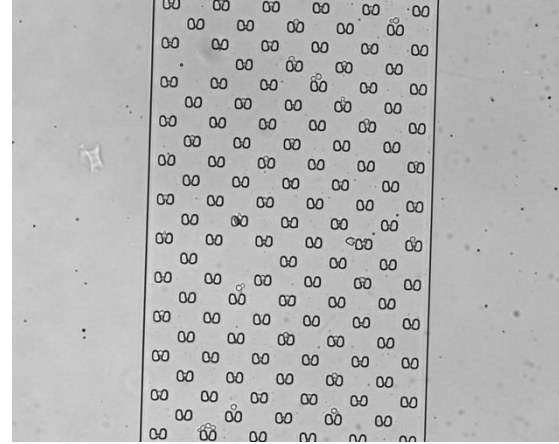
- [14] “Espectro de luz emitido por proteínas fluorescentes”, obtenido de: <https://www.semrock.com/SetDetails.aspx?id=2716> recurso digital consultado el 30 de junio de 2015.
- [15] Escalante Boris, “Apuntes de Procesamiento Digital de Imágenes”, Facultad de Ingeniería UNAM 2009.
- [16] Siegle, F., Vladimirova, T., Ilstad, J., & Emam, O. (2015). “Mitigation of radiation effects in SRAMbased FPGAs for space applications”, *ACM Computing Surveys (CSUR)*, 47(2), 37.
- [17] Gray Stephen, “Local Properties of Binary Images in two dimensions”, *IEEE Transactions on computers*, vol.C-20, Mayo de 1971.
- [18] Garay Erika, Campos Sergio, González De La Cruz Jorge, “High-Resolution Profiling of Stationary-Phase Survival Reveals Yeast Longevity Factors and Their Genetic Interactions”, Febrero 2014.
- [19] “Apuntes de fundamentos de percepción remota”, recurso digital obtenido de http://geoservice.igac.gov.co/contenidos_telecentro/fundamentos_pr-semana2/index.php?id=2 consultado el 24 de junio de 2016.
- [20] Vélez Serrano José Francisco, Moreno Díaz Ana, Sánchez Calle Ángel, “Visión por computador”, 2da edición, 2003.
- [21] Martínez Elena, “Apuntes de Procesamiento Digital de Imágenes”, IIMAS, UNAM 2012.
- [22] Carcedo Y Franco Ana, “Programa de segmentación de regiones en imágenes médicas en MATLAB”, Puebla 2004.
- [23] “Clustering and Clasification methods for Biologists”, apuntes de la Universidad Metropolitana de Mánchester, recurso digital consultado en <http://www.alanfielding.co.uk/multivar/accuracy.htm> el 26 de junio de 2017.
- [24] Alvarado Miguel, “Diseño de un Sistema de Comando, Manejo de Información y Telecomunicaciones (SCMIyT) para un Microsatélite de Percepción Remota”, UNAM 2017.
- [25] 7 Series FPGAs Datasheet: overview, recurso digital consultado en: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf el 23 de junio de 2017.

ANEXO I

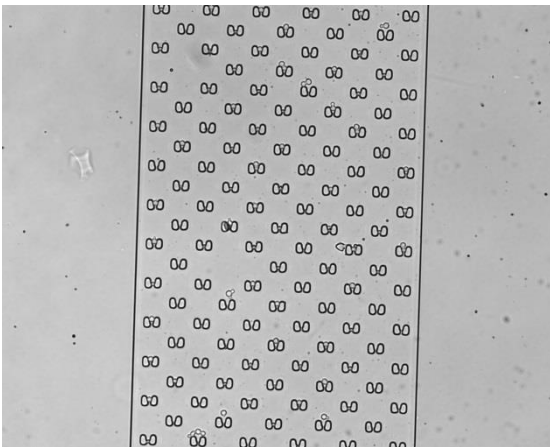
Imágenes del experimento



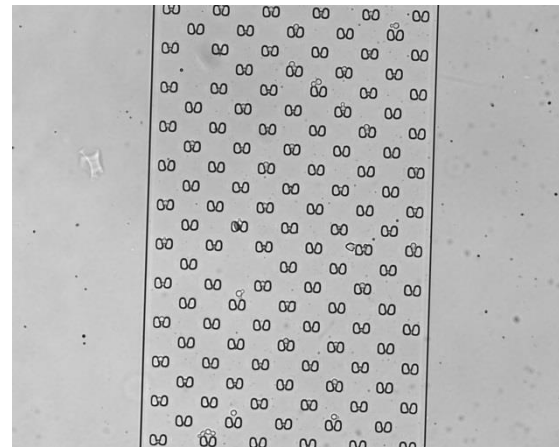
Toma en Delta_t58



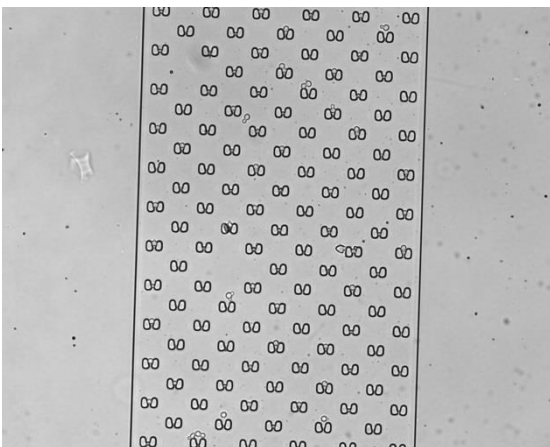
Toma en Delta_t57



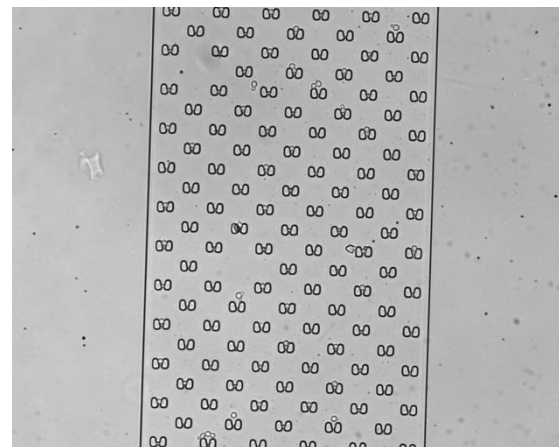
Toma en Delta_t56



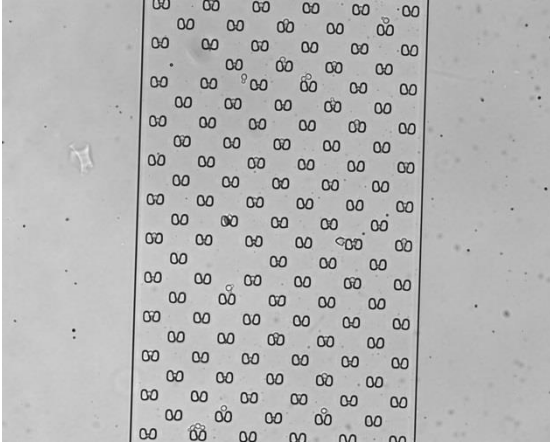
Toma en Delta_t55



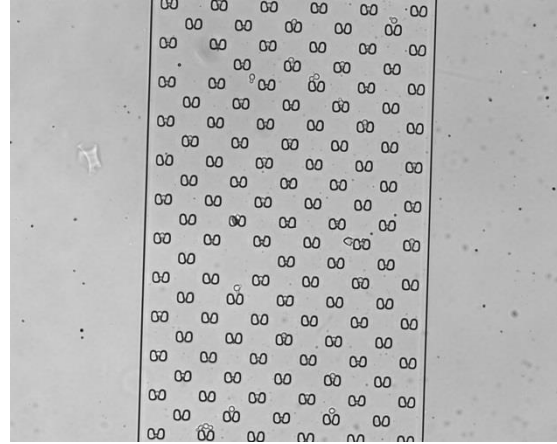
Toma en Delta_t54



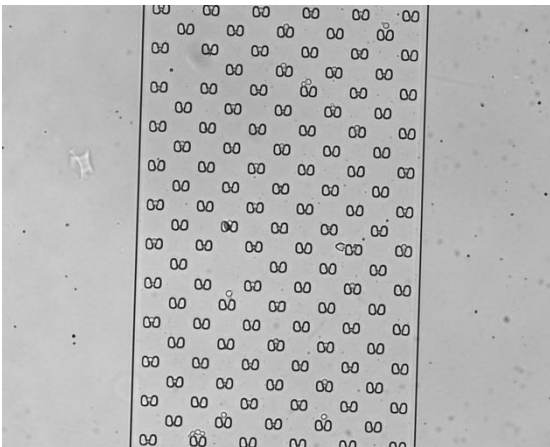
Toma en Delta_t53



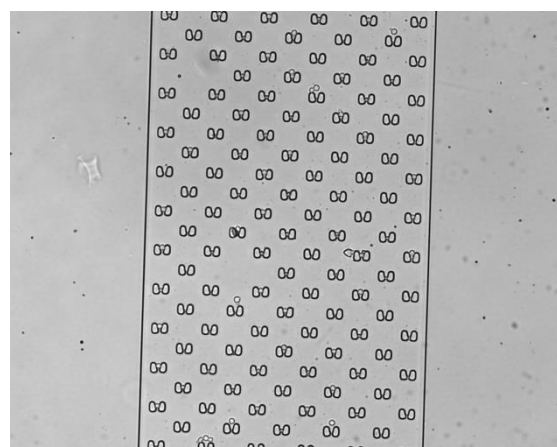
Toma en Delta_t52



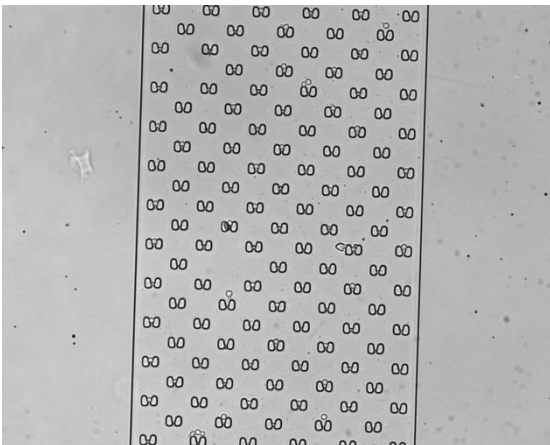
Toma en Delta_t51



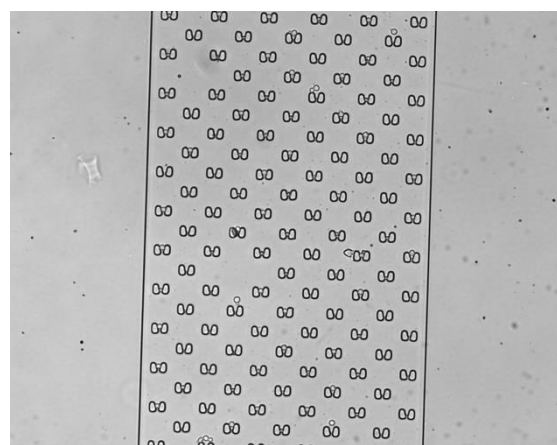
Toma en Delta_t50



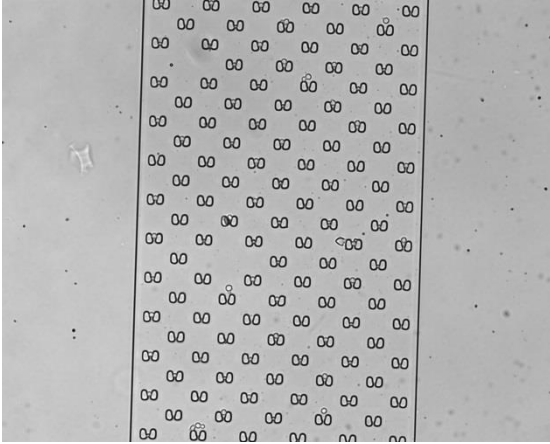
Toma en Delta_t49



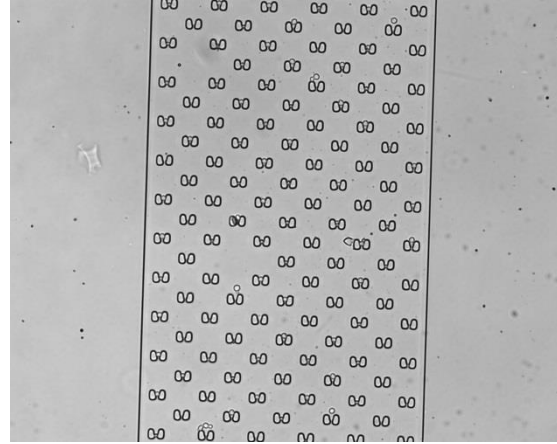
Toma en Delta_t48



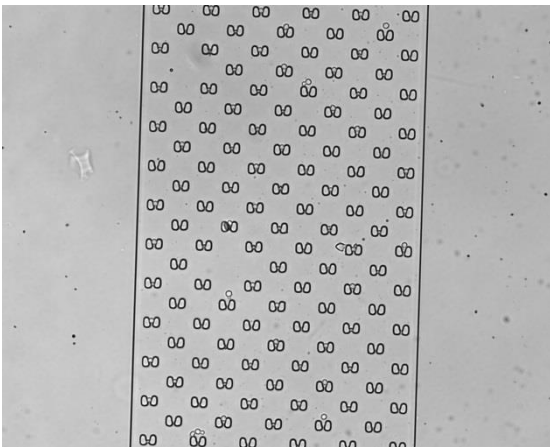
Toma en Delta_t47



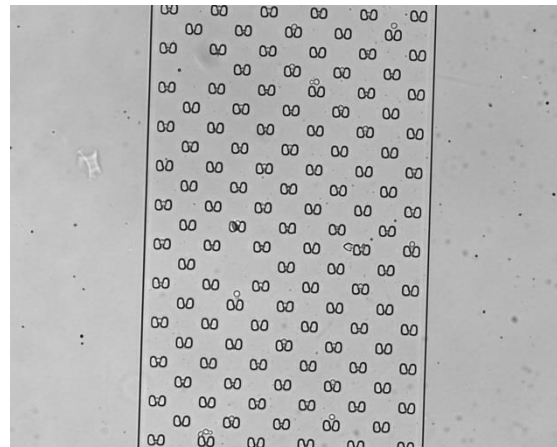
Toma en Delta_t46



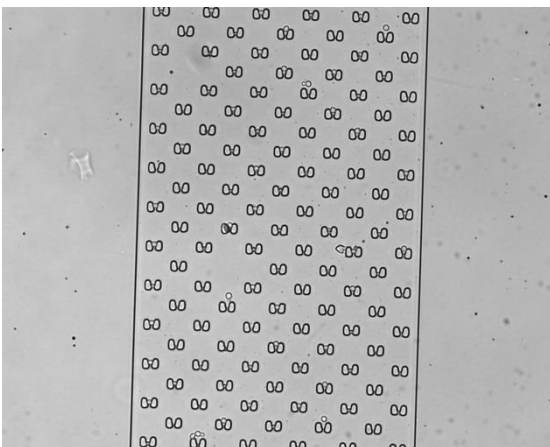
Toma en Delta_t45



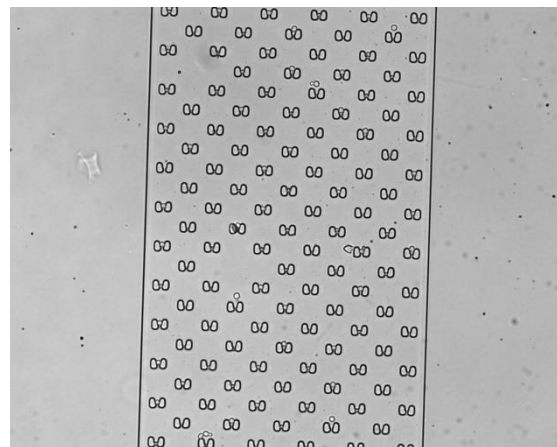
Toma en Delta_t44



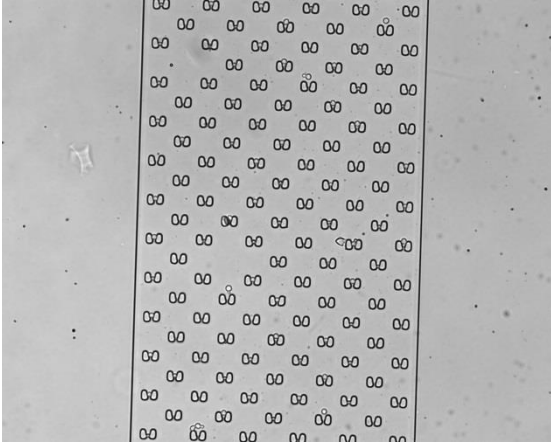
Toma en Delta_t43



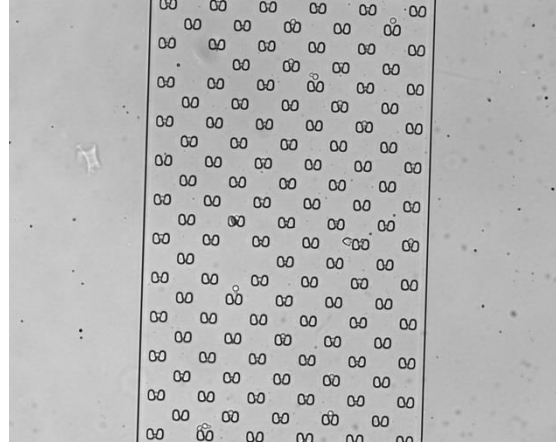
Toma en Delta_t42



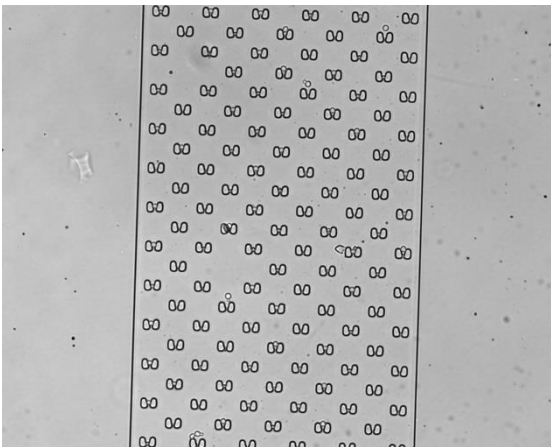
Toma en Delta_t41



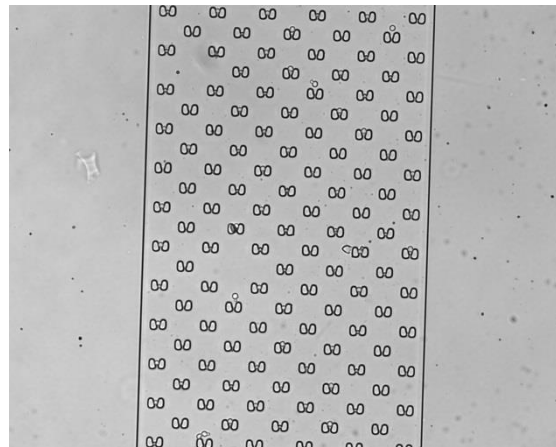
Toma en Delta_t40



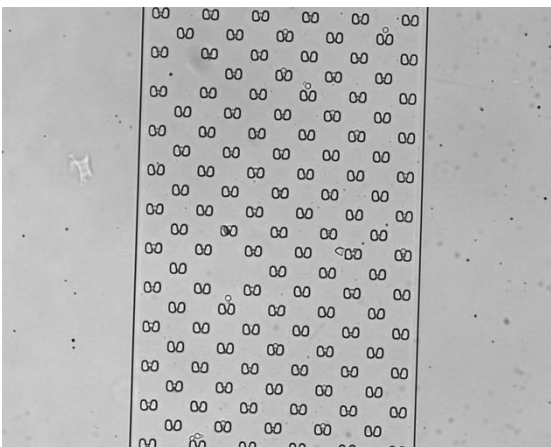
Toma en Delta_t39



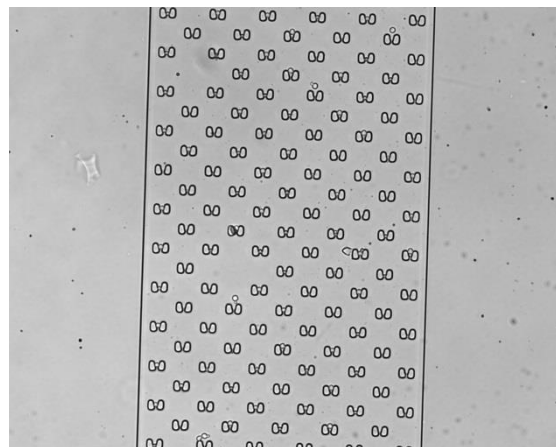
Toma en Delta_t38



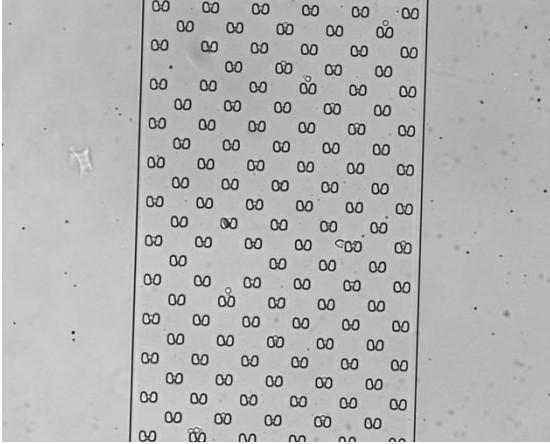
Toma en Delta_t37



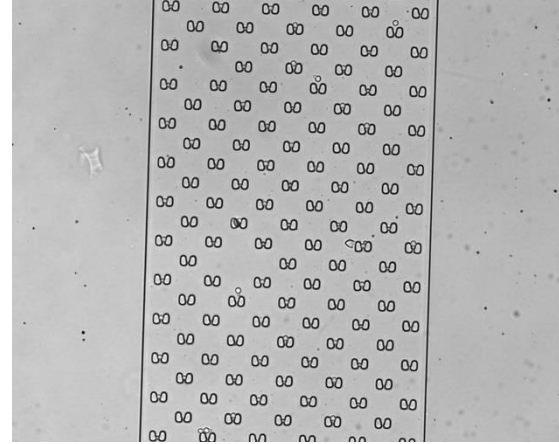
Toma en Delta_t36



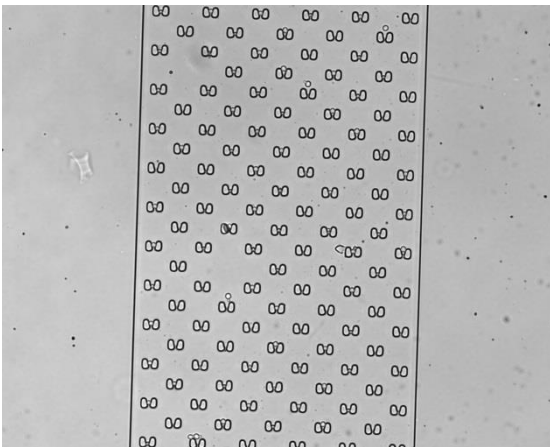
Toma en Delta_t35



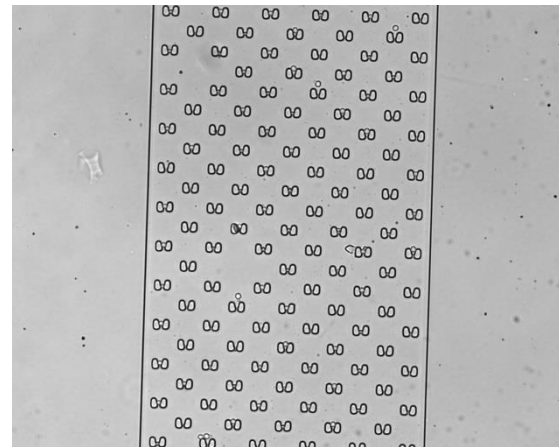
Toma en Delta_t34



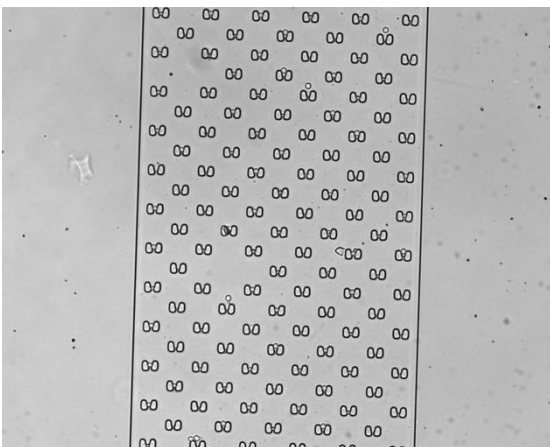
Toma en Delta_t33



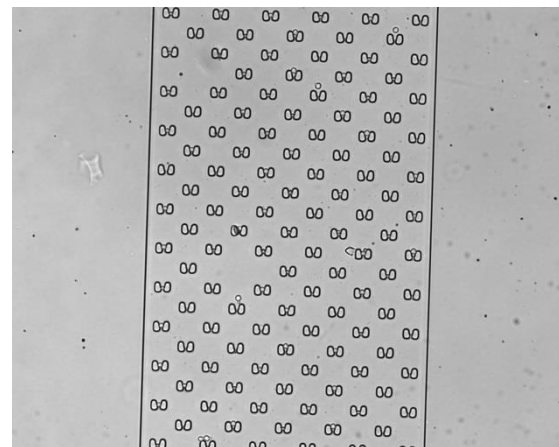
Toma en Delta_t32



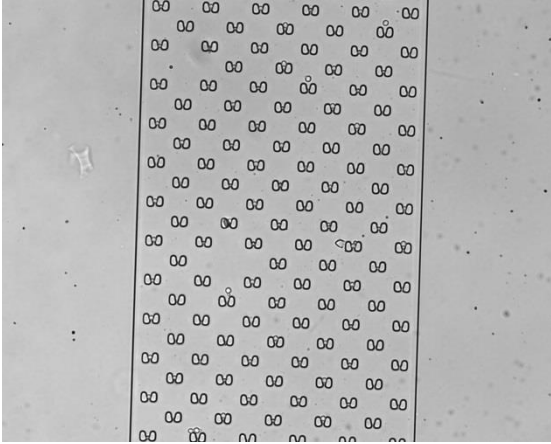
Toma en Delta_t31



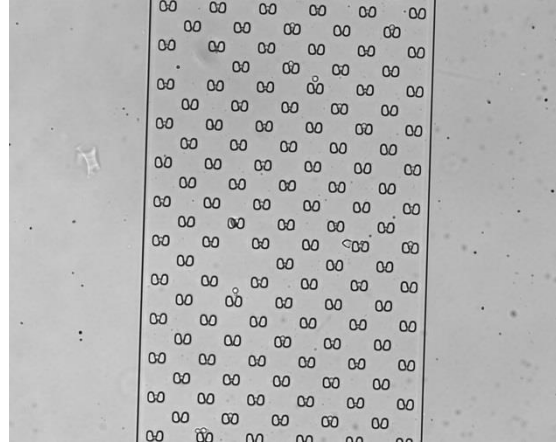
Toma en Delta_t30



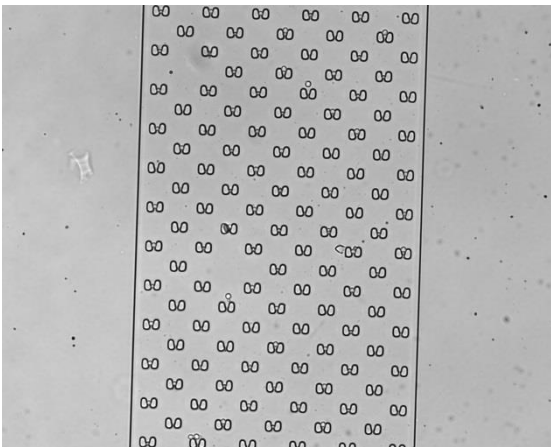
Toma en Delta_t29



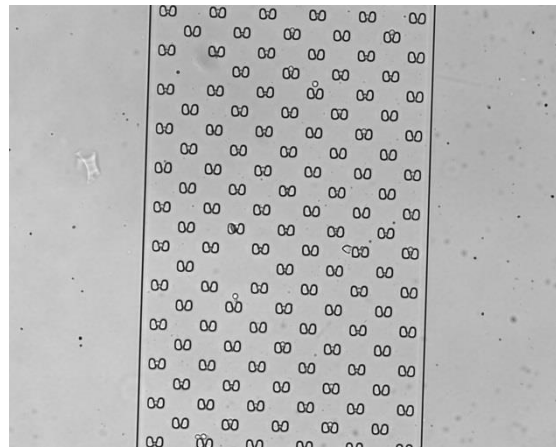
Toma en Delta_t28



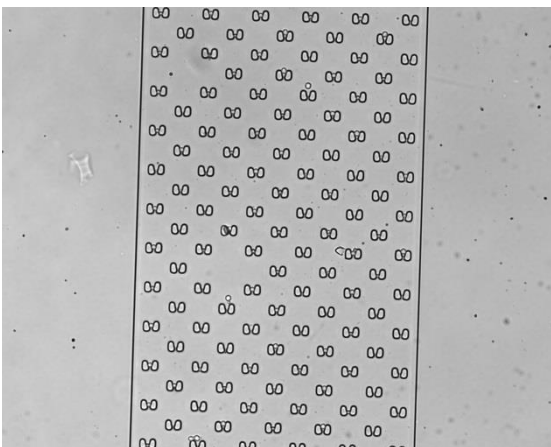
Toma en Delta_t27



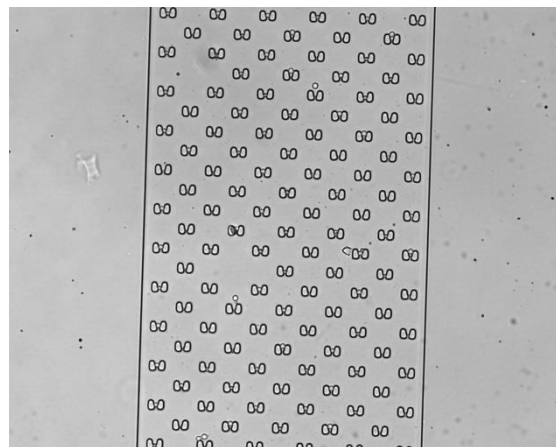
Toma en Delta_t26



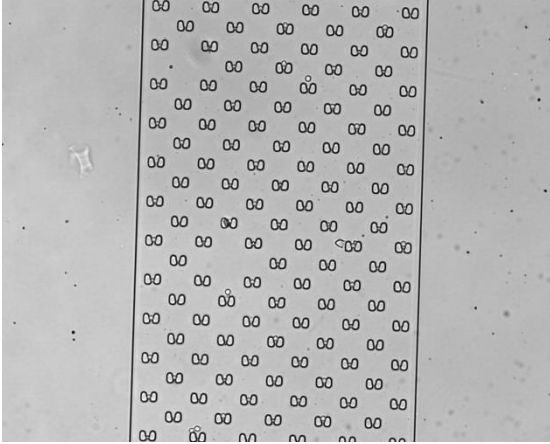
Toma en Delta_t25



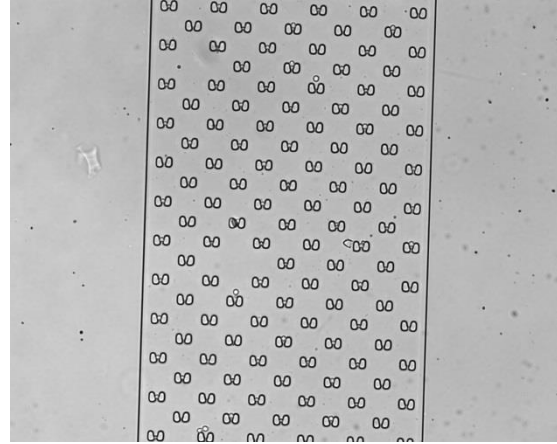
Toma en Delta_t24



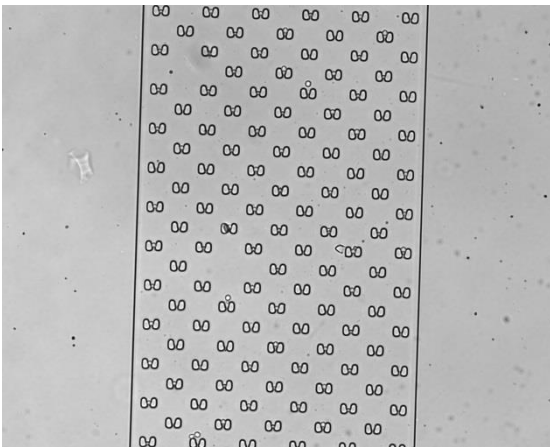
Toma en Delta_t23



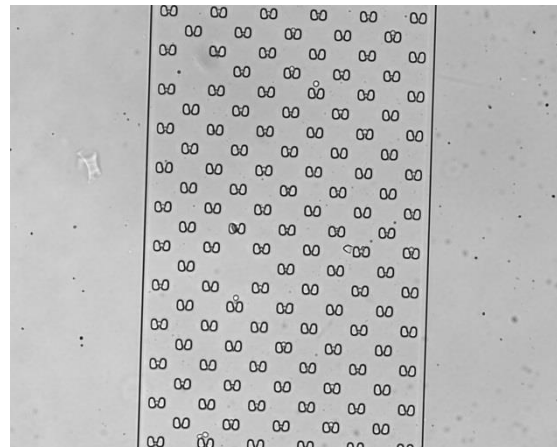
Toma en Delta_t22



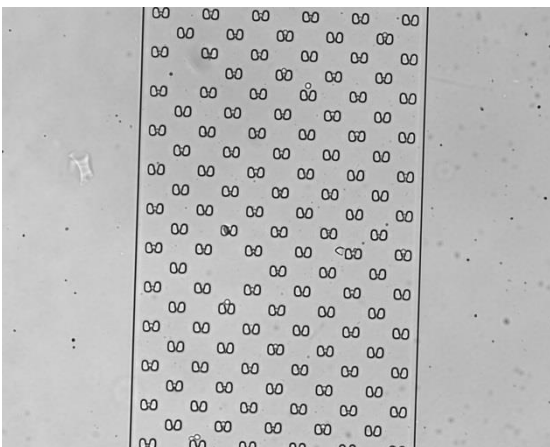
Toma en Delta_t21



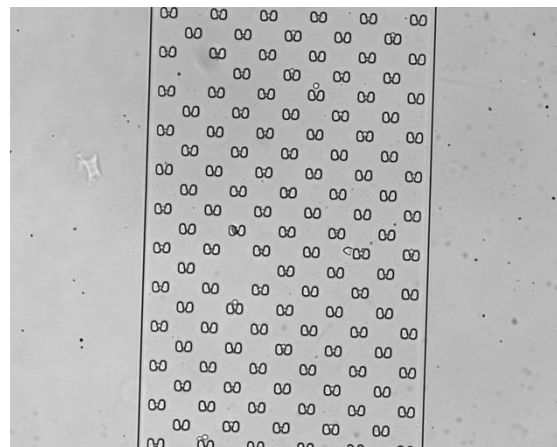
Toma en Delta_t20



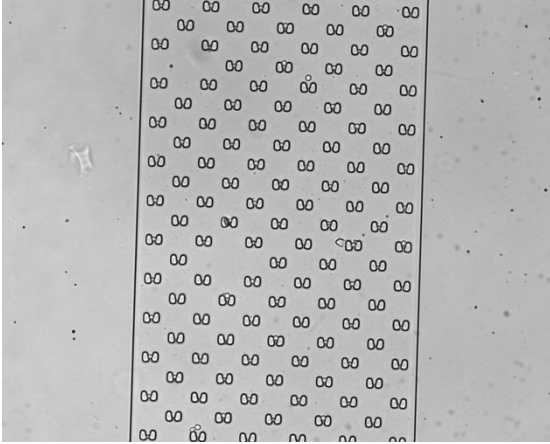
Toma en Delta_t19



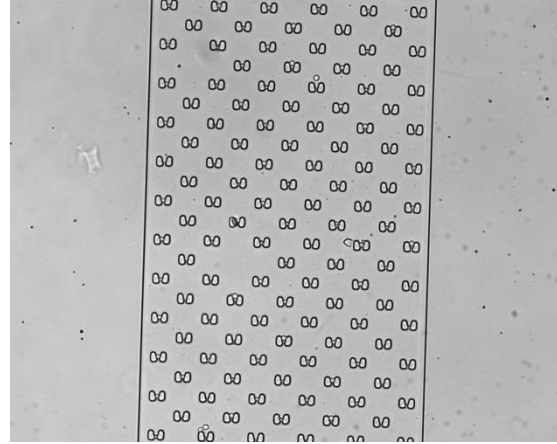
Toma en Delta_t18



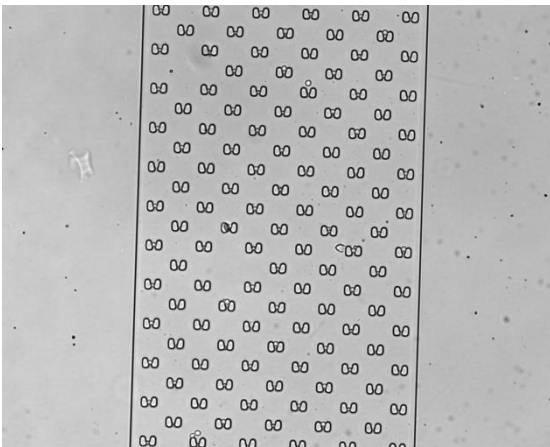
Toma en Delta_t17



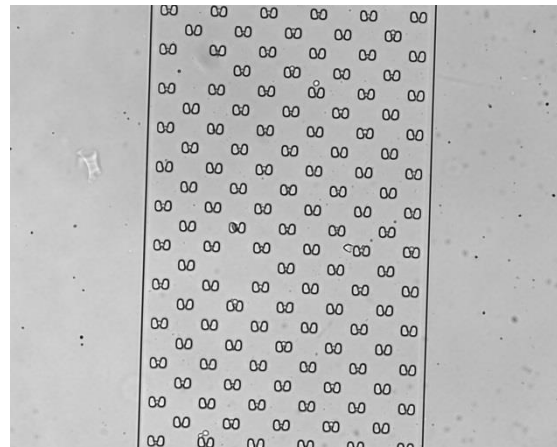
Toma en Delta_t16



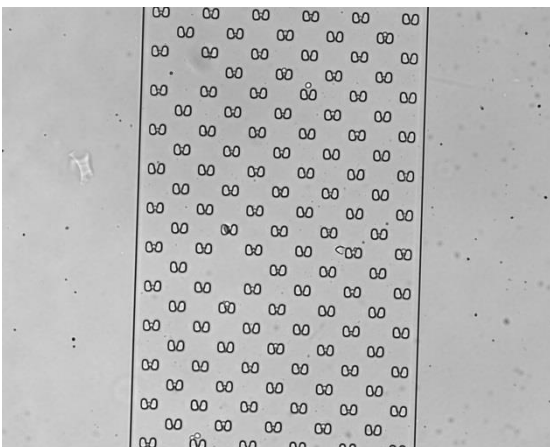
Toma en Delta_t15



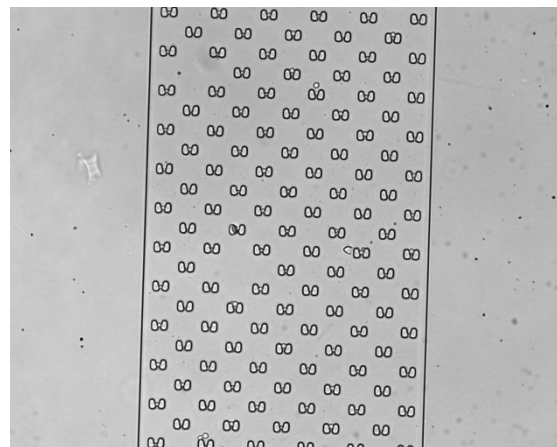
Toma en Delta_t13



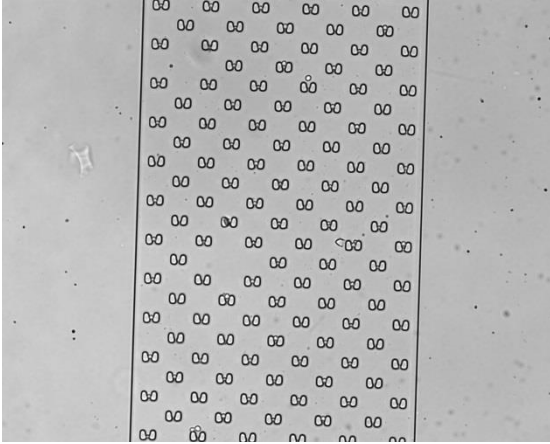
Toma en Delta_t12



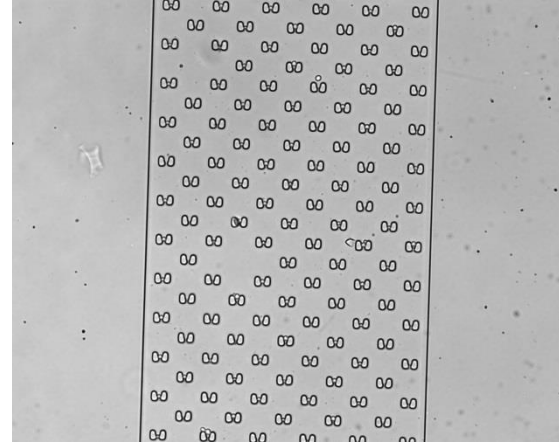
Toma en Delta_t11



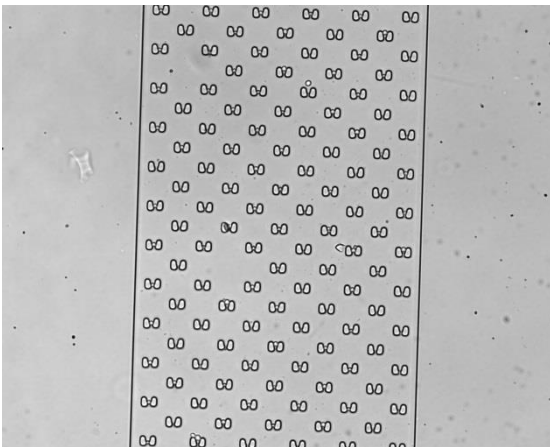
Toma en Delta_t10



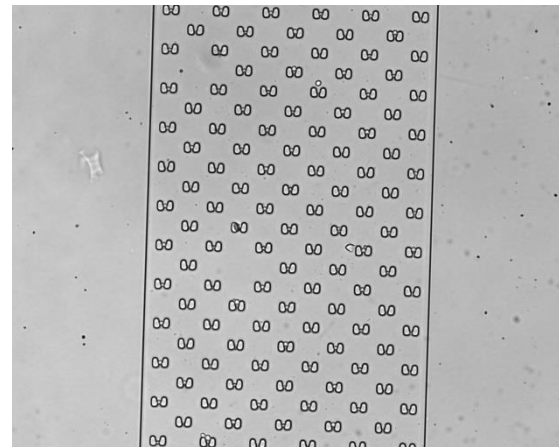
Toma en Delta_t9



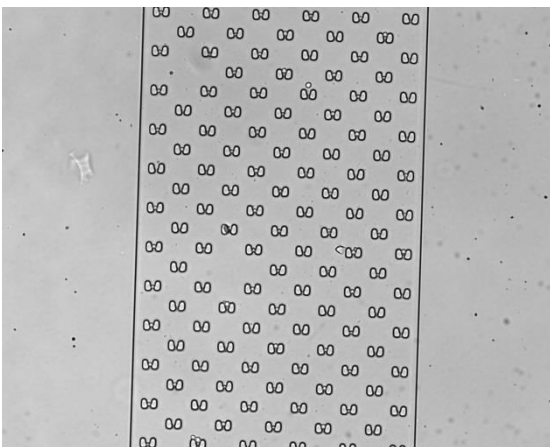
Toma en Delta_t8



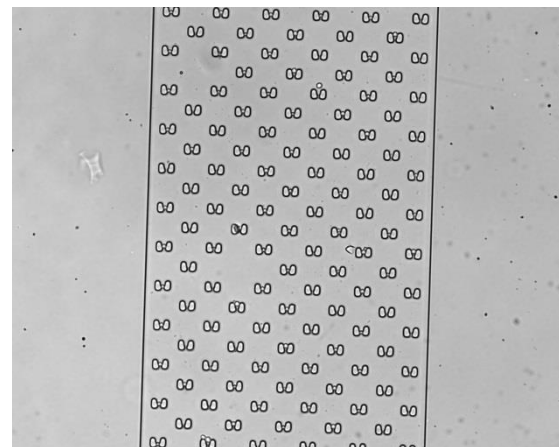
Toma en Delta_t7



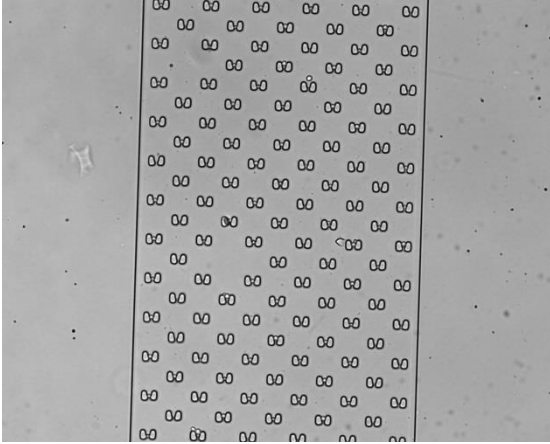
Toma en Delta_t6



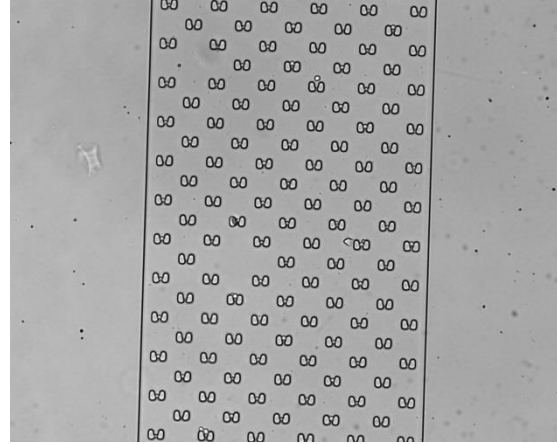
Toma en Delta_t5



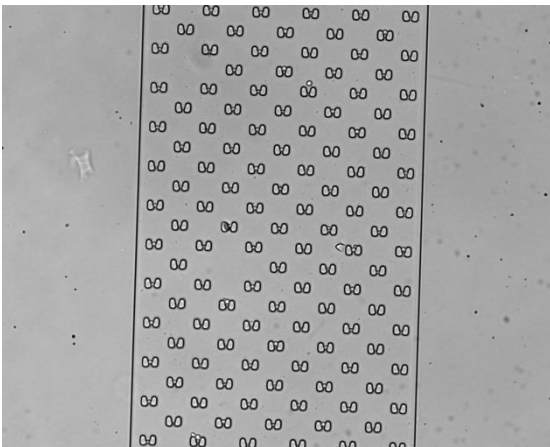
Toma en Delta_t4



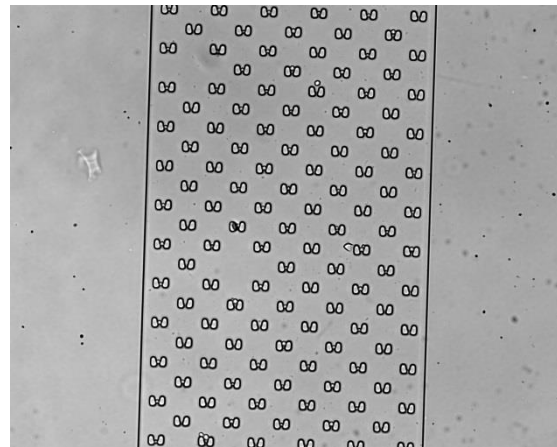
Toma en Delta_t3



Toma en Delta_t2



Toma en Delta_t1



Toma en Delta_t0

ANEXO II

Algoritmos

Control_Function:

```
%Este código controla todos los procesos del Sistema de Visión artificial,
%lo que representa el nivel máximo de abstracción del algoritmo.

clear all; close all; clc; % Limpiamos el ambiente de trabajo

CAPTURAS = 58; %Número de frames o imágenes en el tiempo a analizar
No_TRAMPAS = 124; %Cantidad de trampas a analizar

%Banco de imágenes e índices de trampa
captura = matfile('captura.mat');
coordenadas = matfile('coord.mat');
im_trampa = uint8(imread('C:\Users\Rubi\Documents\AA Tesis\Vision
artificial\trampas\0.jpg','jpeg'));%Como esta imagen es constante la hacemos
global
im_trampa = im_trampa(:,:,1);%Convertimos a escala de grises

anchoI = 2208;%Ancho de la imagen completa
altoI = 2752; %Alto de la imagen completa
ancho = 110; alto = 90; n = 20; %Índices del tamaño del recorte de la iagen
que representa el área de una trampa

%Banco de datos del análisis
CentroX = ones(No_TRAMPAS,1);
CentroY = ones(No_TRAMPAS,1);
Estado_YH = zeros(CAPTURAS+1,No_TRAMPAS);
Estado_MC = zeros(CAPTURAS+1,No_TRAMPAS);
Estado_SR = zeros(CAPTURAS+1,No_TRAMPAS);

%% Inicialización de las memorias externas
%%Rupturas, coincidencia
Generador_Rupturas();
Generador_Coincidencia();

%Cargamos las imágenes en el ambiente de trabajo

for dt = 1 : 10%CAPTURAS-1
    im_delta_t = captura.Captura((dt*anchoI)-(anchoI-1):(anchoI*dt),:);
    %Leemos la captura de la imagen en el delta de tiempo i

    fprintf('CAPTURA: %d ',dt);

    for id = 1 : 124%No_TRAMPAS
        fprintf('TRAMPA: %d \n\n',id);
        H = coordenadas.coord(id,1); V = coordenadas.coord(id,2); H1 = H+109;
V1 = V+89;% pixeles de localización tomando en cuenta trampas de 90V x 110H

        %Recortamos el área de la trampa
        %if id == 56

    %end
    if(H1<altoI && V1<anchoI && H1>0 && V1>0)
        trampa = im_trampa(V:V1,H:H1);
```

```

delta_t = im_delta_t(V:V1,H:H1);

save('imagenes.mat', 'trampa', 'delta_t');

if(Estado_YH(dt,id) ~= 1)

    %% Llamamos a YaHay
    [EstadoYH_Out, centroX_Out, centroY_Out] = ...
        YaHay_Function_Control(id);

    Estado_YH(dt+1,id) = EstadoYH_Out;

    if(EstadoYH_Out == 1) %% Si hay célula
        %Asignamos los valores al arreglo global
        CentroX (id)= centroX_Out;% Lo que salga del Ya Hay
        CentroY (id)= centroY_Out;% Lo que salga del Ya Hay
    end
else

    %% Llamamos a MonitoreoCentros
    [EstadoMC_Out, centroX_Out, centroY_Out] = ...
        MonitoreoCentros_Function_Control(id, CentroX (id),
CentroY (id));

    Estado_MC(dt,id)= EstadoMC_Out;
    %De lo que salgo del análisis de monitoreo:
    if(EstadoMC_Out == 1) %Cálculo correcto del centro

        CentroX (id)= centroX_Out;% Lo que salga del Monitoreo
        CentroY (id)= centroY_Out;% Lo que salga del Monitoreo
        Estado_YH(dt+1,id) = 1; %Para que en la siguiente
iteración se salte el análisis YaHay

        %% Función Se Reproduce
        [EstadoSR_Out] = ...
            SeReproduce_Function_Control(id, CentroX (id),
CentroY (id));

        Estado_SR(dt,id) = EstadoSR_Out;
        if (Estado_SR(dt,id) == 3)
            Estado_YH(dt+1,id) = 0;
            %Si hay error, pedir de nuevo hacer análisis de
            %YaHay
        end
    else

        Estado_YH(dt+1,id) = 0; %Se le asigna el valor de
referencia para la siguiente iteraci[on
        end
    end
    Matriz_Rupturas = matfile('rupturas.mat'); hijas =
Matriz_Rupturas.hijas;
    fprintf('Conteo Hijas TOTAL en el dt%d de la trampa %d = %d
\n',dt,id, hijas(id));

```

```

        fileID = fopen('resultados.txt','wt');
        fprintf(fileID, 'Conteo Hijas TOTAL en el dt%d de la trampa %d =
%d \n', dt, id, hijas(id));
        fclose(fileID);
    end

end

end
end

```

YaHay_Function_Control:

%Este código manda dos imágenes pixel por pixel para ser procesadas en el método YaHay_Function y nos devuelve los datos de interés para saber si ya hay célula en la trampa a analizar.

```

function [Estado_YH_out, centrox_out, centroy_out] =
YaHay_Function_Control(id_in)

```

ancho = 110; alto = 90; n=20; umbral=100; %Esto cambiara con la referencia de las trampas, cambiar tambien en la funcion de procesamiento

%%Guardar la matriz de coincidencia en la memoria externa

```

Matriz_Coincidencia = matfile('coincidencia.mat');
coincidencia = Matriz_Coincidencia.coincidencia; %Leemos la captura de la
imagen en el delta de tiempo i

```

%Traemos las imágenes

```

imagen = matfile('imagenes.mat');
trampa = imagen.trampa;
delta_t= imagen.delta_t;
Huella = matfile('huella.mat'); Huella = Huella.im;

```

%Variables auxiliares para la visualización de resultados

```

imByN = im2bw(delta_t,0.5);

```

%Inicio del programa

%% Reset de las variables globales

```

[ch1Out, ch2Out, status] = ...
    YaHay_Function(uint8(0), uint8(0),0);

```

%% Pasamos las imágenes

```

for y = 1:alto
    for x = 1:ancho
        px_Trampa = trampa(y,x);
        px_deltat = delta_t(y,x);
    end
end

```



```

        [ch1Out, ch2Out, status1] = ...
            YaHay_Function(px_Trampa,px_deltat,1);
    end
end
if status1== 1
    fprintf('La imagen se envi6 correctamente. \n');
else
    fprintf('Error en el envio de la imagen. \n');
end
if status1 ==1 %Indica que ya termin6 de pasar las im6genes
    centrox= ch1Out;
    centroy= ch2Out;

    if(delta_t(centroy, centrox)> umbral) %status diferente de 3

        %% Proceso YaHay

        indiceMin=centrox;
        if(centroy<centrox)
            indiceMin=centroy;
        end

        for i=indiceMin : ancho

            if((centrox+(i-indiceMin+1)) < ancho+1)
                pxX_deltat= delta_t(centroy,centrox+(i-indiceMin+1));
            else
                pxX_deltat=uint8(0);
            end

            if((centroy+(i-indiceMin+1)) < alto+1)
                pxY_deltat= delta_t(centroy+(i-indiceMin+1),centrox);
            else
                pxY_deltat=uint8(0);
            end

            [cxOut, cyOut, status2] = ...
                YaHay_Function(pxY_deltat, pxX_deltat, 2); %% Aproximamos
indices
        end

        if status2==1 %Termino de ver los centros bien
            x1=cxOut;
            y1=cyOut;

            if(ancho-x1 > alto-y1)
                dif=ancho-x1;
            else
                dif=alto-y1;
            end

            for i=1 : dif+10

```

```

    if(y1-i>0)
        pxX_deltat= delta_t(y1-i,centrox);
    else
        pxX_deltat=uint8(0);
    end

    if(x1-i>0)
        pxY_deltat= delta_t(centroy,x1-i);
    else
        pxY_deltat=uint8(0);
    end

    [cxOut, cyOut, status3] = ...
        YaHay_Function(pxX_deltat, pxY_deltat ,3);

end

if(status3==1)
    if(cxOut ~= 0 && cyOut ~= 0) %Posible abertura corregirr
        centro

        %Pedimos los centros
        centrox=cxOut;
        centroy=cyOut;

        %Comparamos con Huella
        for i = 1:(n*2)
            for j = 1:(n*2)
                if(Huella(i,j)>0)
                    px=uint8(Huella(i,j));
                else
                    px=uint8(0);
                end
            end
            [ch1Out, ch2Out, status] = ...
                YaHay_Function(px, uint8(0),5);

        end
    end
    fprintf('Estatus de la trampa: %d \n ', status);
    fprintf('(1: sí hay, 2: no hay) \n');
    if status ==1 %Sí hay célula

        status_celula=1;

        %Pedimos coincidencia
        for y = 1:(n*2)
            for x = 1:(n*2)

                [ch1Out, ch2Out, status6] = ...
                    YaHay_Function(uint8(0), uint8(0),6);
            end
        end
    end
end

```

```

                                if (ch2Out >= 0 && ch2Out < (n*2+1)) &&
(ch1Out >= 0 && ch1Out < (n*2+1))
                                coincidencia((id_in*(2*n+1))-
((2*n+1)-1)+ch2Out,ch1Out+1,:) = status6;
                                end
                                end
                                end
                                save('coincidencia.mat', 'coincidencia');

                                else %No hay célula
                                status_celula=2;
                                fprintf('No hay célula. \n');
                                end
                                else
                                status_celula=2;
                                fprintf('abertura en los ejes donde vemos centro.
\n');

                                end

                                else
                                status_celula=2;
                                fprintf('ERROR posible movimiento en los límites de la
trampa. \n');
                                end
                                else
                                status_celula=2; %Hay un error
                                fprintf('ERROR status2 \n');
                                end

                                else %%Mucho ruido en la imagen STATUS 3
                                fprintf('Estatus 3: hay ruido, posible célula, volviendo a hacer
análisis...) \n');
                                status_celula=3;

                                %Pasamos el centro
                                [cenx, ceny, status4] = ...
                                YaHay_Function(uint8(centrox), uint8(centroy),4);

                                %Comparamos con Huella
                                for i = 1:(n*2)
                                for j = 1:(n*2)
                                if(Huella(i,j)>0)
                                px=uint8(Huella(i,j));
                                else
                                px=uint8(0);
                                end
                                [ch1Out, ch2Out, status5] = ...
                                YaHay_Function(px, uint8(0),5);

                                end
                                end
                                if(status5 ~= 2) %Si el acumulador fue grande
                                %Vemos aproximación
                                for i=1:n*2 %primero hacemos el análisis en dirección x

```

```

        for j=1:n*2
            [cxOut, cyOut, status8] = ...
                YaHay_Function(uint8(0), uint8(0),7);
        end
    end
    centrox=cxOut;
    centroy=cyOut;

    %Pedimos Coincidencia

    for y = 0:(n*2)
        for x = 0:(n*2)
            [ch1Out, ch2Out, status6] = ...
                YaHay_Function(uint8(0), uint8(0),6);

            if (ch2Out >= 0 && ch2Out < (n*2+1)) && (ch1Out >= 0
&& ch1Out < (n*2+1))

                coincidencia((id_in*(2*n+1))-((2*n+1)-
1)+ch2Out,ch1Out+1,:) = status6;
            end
        end
    end

    save ('coincidencia.mat', 'coincidencia');
    else %No hay celula
        status_celula=2;
    end
end

Estado_YH_out = status_celula;
centrox_out = centrox;
centroy_out = centroy;

% Graficas ilustrativas

if(status_celula == 1) %Si sí hay célula
    fprintf('Sí hay célula con centro aproximado: cx: %d, cy: %d. \n',
centrox, centroy);
    imByN(centroy,centrox)=0;

end
figure(1);
colormap(gray(64));
subplot(2,2,1);imagesc(trampa);title('Imagen t');
subplot(2,2,2);imagesc(imByN);title('Imagen en un delta de tiempo');
subplot(2,2,3);imagesc(coincidencia((id_in*(2*n+1))-((2*n+1)-
1):(id_in*(2*n+1)),:,:));title('Imagen coincidencia');
end
end

```

MonitoreoCentros_Function_Control:

%Este codigo es el controlador para el caso 1 del Monitoreo de centros.

```
function [Estado_MC_out, centrox_out, centroy_out] =
MonitoreoCentros_Function_Control(id_in,centrox, centroy)

n=20;

%Traemos las imágenes
imagen = matfile('imagenes.mat');
delta_t= imagen.delta_t;
delta_t=im2bw(delta_t,0.5);

%%Guardar la matriz de coincidencia en la memoria externa
Matriz_Coincidencia = matfile('coincidencia.mat');
coincidencia = Matriz_Coincidencia.coincidencia; %Leemos la captura de la
imagen en el delta de tiempo i

status2=false;
nuevoCentroy=0;nuevoCentrox=0;

%% Reset de las variables globales
[ch1Out, status] = ...
    MonitoreoCentros_Function(false, false, 0);

%% Enviamos coincidencia como caso de prueba
for y = 1:n*2
    for x = 1:n*2
        if(status==false)%Simula el ciclo de reloj
            if(coincidencia((id_in*(2*n+1))-
((2*n+1))+y+1,x+1,:)==0)%Corroborar que los límites nunca se excedan
                %coincidencia(y+1,x+1)
                px1=false;
            else
                px1=true;
            end
            if(delta_t((centroy-n)+y,(centrox-n)+x)==0)%Corroborar que los
límites nunca se excedan
                px2=false;
            else
                px2=true;
            end

            [cambioOut, status] = ...
                MonitoreoCentros_Function(px1,px2,1);
        end
    end
end
if (cambioOut==1) %No hay mucho cambio, leer centros
    %Pedimos los centros
    if(status2==false)
```

```

    [nuevoCentro, status2]= ...
        MonitoreoCentros_Function(px1,px2,2);
nuevoCentrox=(centrox-n)+nuevoCentro;
    [nuevoCentro, status2]= ...
        MonitoreoCentros_Function(px1,px2,2);
nuevoCentroy=(centroy-n)+nuevoCentro;
end

elseif(cambioOut==2) %Centros aproximados Posible crecimiento

    %Imprimimos la distancia cartesiana entre centros
    % distancia=round(square((nuevoCentroy-centroy)*(nuevoCentroy-
centroy)+(nuevoCentrox-centrox)*(nuevoCentroy-centroy)));
    % fprintf('Distancia entre centros> %d \n', distancia);

    %Pedimos los centros
    if(status2==false)

        [nuevoCentro, status2]= ...
            MonitoreoCentros_Function(px1,px2,2);
nuevoCentrox=(centrox-n)+nuevoCentro;
        [nuevoCentro, status2]= ...
            MonitoreoCentros_Function(px1,px2,2);
nuevoCentroy=(centroy-n)+nuevoCentro;
    end

    %Prueba
    %% Reset de las variables globales
    [chlOut, status] = ...
        MonitoreoCentros_Function(false, false, 0);

    for y = 1:n*2
        for x = 1:n*2
            if(status==false)%Simula el ciclo de reloj
                if(coincidencia((id_in*(2*n+1))-
((2*n+1))+y+1,x+1,:)==0)%%Corroborar que los límites nunca se excedan
                    px1=false;
                else
                    px1=true;
                end
                if(delta_t((nuevoCentroy-n)+y,(nuevoCentrox-
n)+x)==0)%%Corroborar que los límites nunca se excedan
                    px2=false;
                else
                    px2=true;
                end

                [cambioOut, status] = ...
                    MonitoreoCentros_Function(px1,px2,1);
            end
        end
    end

    %%Si el cambioOut=1 estaba movida, pedimos los centros

```

```

elseif(cambioOut==3) %Volver a hacer YaHay, mucho cambio.
    nuevoCentrox=centrox;
    nuevoCentroy=centroy;
else %Error en los índices
    nuevoCentrox=centrox;
    nuevoCentroy=centroy;
end

if(cambioOut==2 || cambioOut==3)
    fprintf('Cambio -> %d, Hacer de nuevo YaHay. \n', cambioOut);
end

Estado_MC_out = cambioOut;
centrox_out = nuevoCentrox;
centroy_out = nuevoCentroy;

%% Visualización de resultados
fprintf('Cambio= %d, cenx= %d, ceny= %d. \n',
cambioOut,nuevoCentrox,nuevoCentroy);
fprintf('Centros anteriores cenx= %d, ceny= %d. \n', centrox,centroy);
Aux= delta_t(centroy-19:centroy+19,centrox-19:centrox+19);
Aux(nuevoCentroy+n-1-centroy,nuevoCentrox+n-1-centrox)=0;
xAux(1)=centrox+n-1-centrox;
xAux(2)=nuevoCentrox+n-1-centrox;
yAux(1)=centroy+n-1-centroy;
yAux(2)=nuevoCentroy+n-1-centroy;

figure(2);
colormap(gray(64));
subplot(1,2,1);imagesc(delta_t);title('Imagen en un delta de tiempo');
subplot(1,2,2);imagesc(Aux);title('Corrección de Centro');hold on; plot(xAux,
yAux, 'Color', 'r'); grid on;grid minor;
end

```

SeReproduce_Function_Control:

%Este código es el controlador de la función que determina las rupturas de la célula madre en la trampa, para el análisis de imagen de la misión MICROLAB

```
function [Estado_SR_out] = MonitoreoCentros_Function_Control(id_in,cenx,
ceny)
```

```

ancho = 110; %Esto cambiara con la referencia de las trampas, cambiar tambien
en la funcion de procesamiento
alto = 90; n=20; umbral=100;

```

```

Matriz_Rupturas = matfile('rupturas.mat');
rupturas =Matriz_Rupturas.rupturas;
hijas = Matriz_Rupturas.hijas;

```

```

Estado=0;

%Traemos las imágenes
imagen = matfile('imagenes.mat');
delta_t= imagen.delta_t;
Huella = matfile('huella.mat'); Huella = Huella.im;

%% Hacemos reset a las variables

[ruptura_Out,status_Out ] = ...
    SeReproduce_Function(false, 0, 0);

%% Pasamos deltatByN y Huella
for y = 1:n*2
    for x = 1:n*2
        if y >= 0 && y < alto && x >= 0 && x < ancho
            if(ceny-21+x>0 && cenx-21+y >0)
                if (delta_t(ceny-n+y,cenx-n+x) > umbral) %Binarización
directa
                    px = 1;
                else
                    px = 0;
                end
            else
                px=0;%REVISAR
            end
            if(Huella(y,x)>0)
                huella = Huella(y,x);
            else
                huella =0;
            end

            %aux_borrar(y+1,x+1)= deltatByN(ceny-21+x,cenx-21+y);
        else
            px = 0;
        end

        [ruptura_Out,status_Out ] = ...
            SeReproduce_Function(px, huella, 1);
    end
end

fprintf('Status %d, ruptura %d. \n',status_Out, ruptura_Out);

%% Vemos ruptura

for i=1:24
    [ruptura_Out,status_Out] = ...
        SeReproduce_Function(false, 0, 2);
end

```



```

%%Casos del status
if status_Out==1 %Todo bien
    aux=false;
    angles=zeros(4,1); cont=1;
    for x = 1:24

        [ruptura_Out,status_Out] = ...
            SeReproduce_Function(false, 0 ,3);

        if(status_Out==0)

            if(ruptura_Out ==true)%Si hay ruptura
                if(cont<=4)%Si no sobrepasa el límite de rupturas
                    if(aux==false)
                        angles(cont)=x;
                        cont=cont+1;
                        aux=true;
                        if(rupturas(x, id_in)==false)
                            if (x>1 && x<24)
                                if(rupturas(x-1, id_in)==false &&
rupturas(x+1, id_in)==false)
                                    hijas(id_in)=hijas(id_in)+1;
                                end
                            end
                        end
                    else
                        angles(cont-1)=(angles(cont-1)+x)/2;
                    end

                    else %Si sobrepasa el límite de rupturas ERROR
                        fprintf('ERROR, demasiadas aberturas');
                        Estado=3;
                    end
                    rupturas(x, id_in)= true;
                else %Si no hay ruptura
                    aux=false;
                    rupturas(x, id_in)= false;
                end

            end

        end

        if(cont>1)
            Estado=2;
            for i=1:cont-1
                angles(i)=(angles(i))*15-15;
                fprintf('Ángulo %d: %d grados. \n',i, round(angles(i)));
            end
        else %No hay aberturas
            fprintf('No hay aberturas \n');
            Estado=1;
        end
    end
end

```

```

else %No se mandaron bien los datos

    fprintf('Error');
    Estado=3;
end
save('rupturas.mat', 'rupturas','hijas');
Estado_SR_out = Estado;

%Graficamos para corroborar comportamiento

fprintf('Conteo Hijas = %d \n', hijas(id_in));

ImAux= delta_t(ceny-20:ceny+20,cenx-21:cenx+20);
ImAux=im2bw(ImAux,0.5);
delta_t(ceny,cenx)=0;
if(Estado==2)
    lineLength = 20;
    x1(1) = 21;
    y1(1) = 21;

    x1(2) = x1(1) + lineLength * cosd(angles(1));
    y1(2) = 42 - (y1(1) + lineLength * sind(angles(1)));
else
    x1(1) = 21;
    y1(1) = 21;
    x1(2) = 21;
    y1(2) = 21;
end

figure(3);
colormap(gray(64));
subplot(1,2,1);imagesc(delta_t);title('Imagen Delta_t');
subplot(1,2,2);imagesc(ImAux);title('Dirección de abertura'); hold on;
plot(x1, y1, 'r'); grid on;grid minor;

end

```

YaHay_Function:

```

%%Este algoritmo analiza dos imagenes del proyecto MICROLAB de acuerdo a
%%diferentes procesos, desde la lectura de la imagen, hasta la descripción
%%de los cambios en un delta de tiempo.

```

```

function [ch1_out, ch2_out,status_out] = YaHay_Function(px1_in, px2_in,
bitEscritura)

```

```

% Declaración de las variables a usar en todo el algoritmo

```

```

%persistent px1_binary px2_binary

```

```

persistent Resta coincidencia marcador marcador2 marcador3 marcador4 y_aux
x_aux bandera index i j

```

```

persistent x1 y1 cont cont3 contador contador2 x y centroy centrox status
acumulador

ancho=110; alto=90; umbral=100;%se deben modificar si cambia el tamaño de
imagen
n=20; %Valor medio del radio de una célula

%% Rellenamos las variables si están vacías

if isempty(Resta)
    x1 = 1; y1 = 1; x = 0; y = 0;
    marcador=false;
marcador2=false;marcador3=false;marcador4=false;y_aux=0;x_aux=0;
bandera=false;
    Resta= false(alto,ancho); coincidencia=false(n*2+1,n*2+1); index=1;i=1;
j=1;
    cont=0; cont3=0; centroy =0; centrox=0; contador=0; contador2=0;
status=0;acumulador=double(0);
end

%% Reset de las salidas

if bitEscritura==0
    x1 = 1; y1 = 1; x = 0; y = 0; bandera=false; i=1; j=1;
    marcador=false;
marcador2=false;marcador3=false;marcador4=false;y_aux=0;x_aux=0;
    Resta= false(alto,ancho); coincidencia=false(n*2+1,n*2+1);
    cont=0; cont3=0; centroy =0; centrox=0; contador=0; contador2=0;
status=0;acumulador=double(0); index=1;
    %Aseguramos que siempre los niveles de salida están en bajo
    ch1_out=0;
    ch2_out=0;
    status_out=2;

    %% Lectura y binarización de imagen original o trampa y de deltat
elseif bitEscritura==1

    %% Binarización
    if px1_in > umbral %Pixel de la trampa
        px_trampa=true;
    else
        px_trampa=false;
    end

    if px2_in > umbral %Pixel de deltat
        px_deltat=true;
    else
        px_deltat=false;
    end

    if(px_deltat==false && px_trampa==true) %Equivalente a -1 (píxeles que
aparecieron)
        Resta(y1,x1)=true; %Hacemos Resta de las imágenes para eliminar la
trampa, matrizByN debe ser la trampa sin célula
        cont=cont+1;

```

```

        x=x+x1;
        y=y+y1;
    end

    if(x1 == ancho && y1==alto) %Indica que terminó de leer las imágenes
        x=round(x/cont); %Coordenada del centro en direccion x
        y=round(y/cont); %Coordenada del centro en direccion x
        index=x;
        if(y<x)
            index=y;
        end
        %% BORRAR
        figure(6);
        colormap(gray(64));
        Resta2=double(Resta);
        Resta2(y,x)=3;
        subplot(1,2,1);imagesc(Resta2);title('Resta2');
        %%
        ch1_out=x; ch2_out=y;
        status_out=1;
    else %Aún está procesando
        if(x1==ancho)
            x1=1;
            y1=y1+1;
        else
            x1=x1+1;
        end
        status_out=0;
        ch1_out=0; ch2_out=0;
    end

    %% Proceso ¿Ya Hay célula?

elseif bitEscritura==2 %Lo que hacía selector 1

    %% Binarización
    if px1_in > umbral %Pixel de la trampa
        pxX_deltat=true;
    else
        pxX_deltat=false;
    end

    if px2_in > umbral %Pixel de deltat
        pxY_deltat=true;
    else
        pxY_deltat=false;
    end

    %%Volvemos a sacar el centro por el método 2 para depurar pixeles basura

    %Hacia abajo-arriba
    if (index < alto) %ver limites no sobrepasen

```

```

        if(pxX_deltat==false && marcador == false)
            y1=index;
            marcador=true;
        end
    end
    if (index < ancho) %ver limites no sobrepasen
        if(pxY_deltat==false && marcador2 == false)
            x1=index;
            marcador2=true;
        end
    end
    if((marcador == true && marcador2 ==true) ||(index==ancho) )
        status_out=1; %terminó
        ch1_out=x1; ch2_out=y1;
        index=1;
    else
        index=index+1;
        status_out=0; %sigue procesando
        ch1_out=0; ch2_out=0;
    end
elseif bitEscritura==3 %selector==2

%% Binarización
if px1_in > umbral %Pixel de la trampa  deltatByN (y1-index,x)
    pxX_deltat=true;
else
    pxX_deltat=false;
end

if px2_in > umbral %Pixel de deltat
    pxY_deltat=true;
else
    pxY_deltat=false;
end

if(y1-index>0 && y1-index <alto) %Cambie esto

    if (pxX_deltat==true && marcador3 == false && index+y1 <= alto)
        centroy=centroy+y1-index;
        contador=contador+1;
    elseif pxX_deltat==false
        marcador3=true;
    end
end

if(x1-index>0 && x1-index <ancho)%Cambie esto
    if (pxY_deltat==true && marcador4 == false && index+x1 <= ancho)
        centrox=centrox+(x1-index);
        contador2=contador2+1;
    elseif pxY_deltat==false
        marcador4=true;
    end
end
end

```

```

    if((marcador3 == true && marcador4 ==true) ||(index+x1 >= ancho &&
index+y1 >= alto) )

        if(bandera==false)
            if(contador2>0)
                centrox=round(centrox/contador2);
            end
            if(contador>0)
                centroy=round(centroy/contador);
            end
            bandera=true;
        end

        %fprintf('Contadores 1: %d 2: %d \n', contador, contador2);
        if(contador<2*n-(n/2) && contador2< 2*n-(n/2))%Asegura que no esté
contando las aberturas en esas direcciones
            ch1_out=centrox; ch2_out=centroy;
        else
            ch1_out=0; ch2_out=0;
        end

        status_out=1; %terminó
    else
        index=index+1;
        status_out=0; %sigue procesando
        ch1_out=0; ch2_out=0;
    end

elseif bitEscritura==4 %% Si el status es 3 conviene pasar el centro

    centrox=double(px1_in);
    centroy=double(px2_in);
    ch1_out=0; ch2_out=0; status_out=1;

elseif bitEscritura==5

    %%Procesamos con Huella

    px_Huella=double(px1_in);

    if(centroy-n+i<=alto && centrox-n+j<=ancho && centroy-n+i >0 && centrox-
n+j>0)
        if Resta(centroy-n+i,centrox-n+j) == true && px_Huella >=1% Si es
blanco le damos el valor de peso que tiene en la huella
            coincidencia(i,j)= true;
            acumulador=px_Huella+acumulador;
        end
    end
end

```

```

    if(i == n*2) && (j == n*2)%Acabó
        i=1;j=1;
        if acumulador > 500 %%Este valor (500) es estadístico, se cambia si
se genera una nueva huella o se trabajan imágenes con diferente brillo
            fprintf('Acumulador: %d \n', acumulador);
            status = 1; %Si hay
        else
            fprintf('Acumulador2: %d \n', acumulador);
            status = 2; %No hay
        end
    else
        if(j==n*2)
            j=1;
            i=i+1;
        else
            j=j+1;
        end
        status = 0; %Procesando
    end

    status_out=status; ch1_out=0; ch2_out=0;

    %% Proceso enviar coincidencia
elseif bitEscritura==6

    if(i==n*2 && j==n*2) %Terminó
        i=1;j=1;
        ch1_out = i;
        ch2_out = j;
        status_out=2;
    else %Procesando

        if(j==n*2)
            j=1;
            i=i+1;
        else
            j=j+1;
        end

        if(coincidencia(j,i)==true)
            status_out=1;
        else
            status_out=0;
        end
        ch1_out = i;
        ch2_out = j;
    end

    %% Calculamos centro aproximado
elseif bitEscritura==7 %El centro no cayó en blanco, mucho ruido
%%Indica que el centro no cayó en blanco y
%%puede que sólo sea el centro aproximado
%%NUEVO Hacemos analisis del centro con coincidencia
valor=1; valor2=1;

```

```

if(j<=n*2 && i<=n*2) %Verificamos que entra en el rango
    if coincidencia(i,j) == 1 %Si el pixel CAMBIAR por la nueva matriz
        cont3=cont3+1;
        y_aux=y_aux+i;
        x_aux=x_aux+j;
    end
end
if(j==n*2 && i==n*2) %Terminó
    j=1; i=1;
    if(cont3~=0)
        valor=x/cont3;
        valor2=y/cont3;
    end
    centrox=centrox-n+round(valor); %Coordenada del centro en direccion x
    centroy=centroy-n+round(valor2); %Coordenada del centro en direccion
y

    ch1_out=centrox; ch2_out=centroy; status_out=1;

else %Procesando
    if(j==n*2)
        j=1;
        i=i+1;
    else
        j=j+1;
    end
    ch1_out=0; ch2_out=0; status_out=0;
end

%% Error de entrada (caso default)
else
    %% Enviamos código de error

    ch1_out=0; ch2_out=0; status_out=0;
end

```

MonitoreoCentros_Function:

```

%%Este algoritmo monitorea el centro de los pixeles pertenecientes a una
%%célula madre del proyecto MICROLAB dado un centro aproximado calculado
%%por el método "YaHay_Function" que forma parte del sistema de visión a
%%bordo.

```

```

function [ch1_out, status_out] = MonitoreoCentros_Function(px1_in, px2_in,
bitEscritura)

```

```

%% Declaración de las variables a usar en todo el algoritmo

```

```

%persistent cenx cenx

```



```

persistent x1 y1 Resta ceny cenx coincidentes faltantes aux xmin xmax ymin
ymax
n=20; %Valor medio del radio de una célula
umbral =0.8; %%Cambiar de acuerdo a analisis estadistico
umbralMin=0.5;

%% Rellenamos las variables si están vacías

if isempty(Resta)
    x1 = 1;y1 = 1; aux=false; ymin=40; ymax=1; xmin=40; xmax=1;
    coincidentes=1;faltantes=1;
    Resta=false(n*2,n*2);
    ceny=0; cenx=0;
end

%% Inicialización de todo o Reset y lectura de centro

if bitEscritura==0
    x1 = 1;y1 = 1; aux=false; ymin=40; ymax=1; xmin=40; xmax=1;
    coincidentes=1;faltantes=1;
    Resta=false(n*2,n*2);
    ceny=0; cenx=0;

    %Aseguramos que siempre los niveles de salida están en bajo
    chl_out=0;
    status_out=false;

    %% Proceso leer coincidencia y recorte
elseif bitEscritura==1

    coincidencia=px1_in;
    recorte=px2_in;

    if(recorte==false && coincidencia==true) %Significa que coinciden
        Resta(y1,x1)=true;
        coincidentes=coincidentes+1;

        if(x1<xmin && x1>1 && y1>1)
            if(Resta(y1,x1-1)==true && Resta(y1-1,x1)==true)%Conectividad 2
                xmin=x1;
            end
        end
        if(y1<ymin && x1>1 && y1>1);
            if(Resta(y1,x1-1)==true && Resta(y1-1,x1)==true)%Conectividad 2
                ymin=y1;
            end
        end
        if(x1>xmax && x1>1 && y1>1)
            if(Resta(y1,x1-1)==true && Resta(y1-1,x1)==true)%Conectividad 2
                xmax=x1;
            end
        end
        if(y1>ymax && x1>1 && y1>1);
            if(Resta(y1,x1-1)==true && Resta(y1-1,x1)==true)%Conectividad 2
                ymax=y1;
            end
        end
    end
end

```

```

        end
    end

    else
        Resta(y1,x1)= false;

    end

    if(recorte==true && coincidencia==true)%Significa pixel faltante
        faltantes=faltantes+1;
    end

    if(y1==n*2 && x1==n*2)
        status_out=true; %Terminó de leer bien

        ceny=round(( (ymax-ymin)/2)+ymin);
        cenx=round(( (xmax-xmin)/2)+xmin);

        %fprintf('Porcentaje %d \n',coincidentes/(faltantes+coincidentes));

        if coincidentes/(faltantes+coincidentes) > umbral %Cambio pequeño
            fprintf('Coincidentes: %d \n',
coincidentes/(faltantes+coincidentes));
            cambio=1;
            elseif((ceny>n-3 && ceny<n+3 && cenx>n-3 &&
cenx<n+3)&&(coincidentes/(faltantes+coincidentes) > umbralMin))%Mucho cambio,
pero centros aproximados +-2px
                cambio=2;
            else%Cambio grande en la estructura, volver a hacer análisis de YaHay
                cambio=3;
            end

    else
        cambio=0;
        if(x1==n*2)
            x1=1;
            y1=y1+1;
        else
            x1=x1+1;
        end

        status_out=false; %Procesando
    end

    ch1_out=cambio;

elseif bitEscritura==2 %Pasamos los centros
    if(aux==false)
        ch1_out=cenx;
        status_out=false;
        aux=true;
    else
        ch1_out=ceny;
        status_out=true;
    end
end

```

```

else %%ERROR de entrada
    chl_out=0;
    status_out=true;
end

```

SeReproduce_Function:

%%Este algoritmo ayuda a determinar las rupturas de la célula para detectar su reproducción.

```

function [ruptura_out,status_out] = SeReproduce_Function(px1_in,px2_in,
indice) %px1_in=deltatByN px2_in=matrizByN

```

```

persistent aux x1 y1 ruptura i j

```

%%Rectas para determinar ruptura con respecto al centro

```

n=20; nAux=false;

```

% Coordenadas en x

```

R1xa=[22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40];
R2xa=[29,30,31,32,33,34,35,36,37,38,39,40];
R3xa=[24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40];
R4xa=[22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40];
R5xa=[23,23,24,24,25,26,27,27,27,28,28,28,29,29,30,30,31,32];
R6xa=[21,22,22,23,23,23,23,24,24,24,25,25,26,26,26,27,27,28];
R7xa=[21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21];

```

```

R1xb=[20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2];
R2xb=[13,12,11,10,9,8,7,6,5,4,3,2];
R3xb=[18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2];
R4xb=[20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2];
R5xb=[19,19,18,18,17,16,15,15,15,14,14,14,13,13,12,12,11,10];
R6xb=[21,20,20,19,19,19,19,18,18,18,17,17,16,16,16,15,15,14];

```

% Coordenadas en y

```

R1ya=[21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21];
R2ya=[18,18,18,18,17,17,16,15,15,15,14,14];
R3ya=[19,19,19,18,18,17,16,15,14,14,13,12,12,11,10,9,9];
R4ya=[19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1];
R5ya=[17,16,15,14,13,13,12,11,10,9,8,7,6,5,4,3,2,1];
R6ya=[17,16,15,14,13,12,12,11,10,9,8,7,6,5,4,3,2,1];
R7ya=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21];

```

```

R2yb=[24,24,24,24,25,25,26,27,27,27,28,28];
R3yb=[23,23,23,24,24,25,26,27,28,28,29,30,30,31,32,33,33];
R4yb=[23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41];
R5yb=[25,26,27,28,29,29,30,31,32,33,34,35,36,37,38,39,40,41];
R6yb=[25,26,27,28,29,30,30,31,32,33,34,35,36,37,38,39,40,41];
R7yb=[41,40,39,38,37,36,35,34,33,32,31,30,29,28,27,26,25,24,23,22,21];

```

%%% si el status es 2 tiene que leer la matriz de aberturas

```

if isempty(aux)

```

```

x1=1; y1=1; i=1; j=1;
aux=true(n*2+1,n*2+1);
ruptura=true(24,1); %Tomamos la cantidad de angulos a revisar
end

if(indice==0) %Reset
x1=1; y1=1; i=1;j=1;
ruptura=true(24,1);
aux=true(n*2+1,n*2+1);

%Ponemos las salidas a nivel bajo
status_out=0;
ruptura_out=false;

elseif (indice==1)
%%Huella sobre imagen binaria sobre centro ya calculado
deltatByN = px1_in; % deltatByN(ceny-21+x1,cenx-21+y1)
Huella = px2_in; %En el testbench validar que es >0

if (Huella > 0 && deltatByN==0)% Si es blanco le damos el valor de peso
que tiene en la huella
%Verificamos que no exceda nunca los límites de la matriz
if((x1 <= n*2) && (y1 <= n*2))
aux(y1,x1)=false;
end
end

if(x1>=n*2 && y1>=n*2) %Terminó
status_out=1;
ruptura_out=false;
else %Procesando

if(x1==n*2)
x1=1;
y1=y1+1;
else
x1=x1+1;
end
status_out=0;
ruptura_out=false;
end

elseif(indice==2)

%Preguntamos el status del pixel para determinar si hay ruptura de borde

if i<=19 % HORIZONTALES
if(aux(R1ya(i),R1xa(i))== nAux)
ruptura(1)=nAux;
end
if(aux(R1ya(i),R1xb(i))== nAux)
ruptura(13)=nAux;
end
end
end

```

```

if i<=12 % 15 GRADOS
    if (aux(R2ya(i),R2xa(i))== nAux)
        ruptura(2)=nAux;

    end

    if (aux(R2ya(i),R2xb(i))== nAux)
        ruptura(12)=nAux;

    end

    if (aux(R2yb(i),R2xb(i))== nAux)
        ruptura(14)=nAux;

    end

    if (aux(R2yb(i),R2xa(i))== nAux)
        ruptura(24)=nAux;

    end

end

end

if i<=17 % 30 GRADOS
    if (aux(R3ya(i),R3xa(i))== nAux)
        ruptura(3)=nAux;

    end

    if (aux(R3ya(i),R3xb(i))== nAux)
        ruptura(11)=nAux;

    end

    if (aux(R3yb(i),R3xb(i))== nAux)
        ruptura(15)=nAux;

    end

    if (aux(R3yb(i),R3xa(i))== nAux)
        ruptura(23)=nAux;

    end

end

end

if i<=19 % 45 GRADOS
    if (aux(R4ya(i),R4xa(i))== nAux)
        ruptura(4)=nAux;

    end

    if (aux(R4ya(i),R4xb(i))== nAux)
        ruptura(10)=nAux;

    end

    if (aux(R4yb(i),R4xb(i))== nAux)
        ruptura(16)=nAux;

    end

    if (aux(R4yb(i),R4xa(i))== nAux)
        ruptura(22)=nAux;

    end

```

```

    end
end

if i<=18 % 60 GRADOS
    if (aux(R5ya(i),R5xa(i))== nAux)
        ruptura(5)=nAux;

    end
    if (aux(R5ya(i),R5xb(i))== nAux)
        ruptura(9)=nAux;

    end
    if (aux(R5yb(i),R5xb(i))== nAux)
        ruptura(17)=nAux;

    end
    if (aux(R5yb(i),R5xa(i))== nAux)
        ruptura(21)=nAux;

    end
end

if i<=18 % 75 GRADOS
    if (aux(R6ya(i),R6xa(i))== nAux)
        ruptura(6)=nAux;

    end
    if (aux(R6ya(i),R6xb(i))== nAux)
        ruptura(8)=nAux;

    end
    if (aux(R6yb(i),R6xb(i))== nAux)
        ruptura(18)=nAux;

    end
    if (aux(R6yb(i),R6xa(i))== nAux)
        ruptura(20)=nAux;

    end
end

if i<=21 % VERTICALES
    if (aux(R7ya(i),R7xa(i))== nAux)
        ruptura(7)=nAux;

    end
    if (aux(R7yb(i),R7xa(i))== nAux)
        ruptura(19)=nAux;

    end
end

if(i>=21) %Terminó
    status_out=1; %Indica que termino el procesamiento

```

```

else
    i=i+1;
    status_out=0; %Procesando
end
ruptura_out=false;

elseif(indice==3) %Sacamos la matriz de ruptura

if(j<=24) %Nos aseguramos que nunca exceda los límites
    ruptura_out= ruptura(j);
else
    ruptura_out=false;
end

if (j>=24)%Terminó
    status_out=1;
else %Procesando
    j=j+1;
    status_out=0;
end
else %ERROR
    status_out=0;
    ruptura_out=false;
end

```