Universidad de las Ciencias Informáticas

Facultad 6

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

Título: Desarrollo de un video sensor para el control de perimetrado virtual.

Autor(a): Yaimit Creagh Venero.

Tutor: Ing. Fernando Echemendía Tourt.

Cotutor: Reinier Pupo Ruiz.

Ciudad de la Habana, junio del 2011

AGRADECIMIENTOS

A mi mamá por ser la mejor madre del mundo, por estar a mi lado en todos los momentos, por ser madre, padre y amiga, por dedicar su existencia, su esfuerzo y empeño a que yo salga adelante. Por su paciencia, amor, dedicación, comprensión; por aguantar mis malcriadeces, por haberme formado, guiado y hacer de mí la persona que soy. Te lo debo todo mamita, te amo, eres lo más grande que tengo en esta vida.

A mis abuelas Lucía y María por sus consejos, su paciencia, su dedicación y por estar siempre a mi lado, las amo.

A mi padrastro por contribuir a mi formación, por darme un lugar en su vida, por haber sido para mí como un padre, por aguantar mis malcriadeces.

A mis primas, por estar a mi lado y por ser como las hermanas que no tuve.

A toda mi familia por su apoyo y comprensión, los quiero mucho.

A Felix Noel por su empeño, paciencia, por soportarme y guiarme estos cinco años, creo que sin él nada de esto habría sido posible, te quiero mucho nenito.

A todas las amistades que he cultivado durante estos 5 años, especialmente a mis amigas de toda la vida Elisa, Misbel, Arianne, Susana, Sonia, Yordialis y a mi amigo Raydel.

A mi tutor por guiarme y orientarme durante todo el desarrollo de mi tesis y por ser más que un tutor, un amigo para mí.

A Joel y a Pupo, creo que sin ellos yo no estaría hoy aquí, muchas gracias de todo corazón.

En fin gracias a todos los que de una forma u otra contribuyeron a mi formación profesional así como al desarrollo de este trabajo.

Yaimit Creagh Venero.

DEDICATORIA

A mi papá que aunque no está físicamente conmigo, lo llevo en mi corazón y en mi mente cada día de mi vida.

A mi mamita linda, por ser padre y madre para mí, por dedicarme su vida entera.

A mi primi Dairenys, que Dios la tenga en el cielo.

A mis abuelas, las amo mucho.

Yaimit Creagh Venero.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al Centro de Geoinformática y Señales Digitales de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los 14 días del mes de junio del año 2010

Yaimit Creagh Venero.	Ing. Fernando Echemendía Tourt.

DATOS DE CONTACTO

Ing. Fernando Echemendía Tourt.

Graduado de Ingeniero en Ciencias Informáticas en el año 2008. Se desempeña como desarrollador e investigador en el proyecto Video Vigilancia (SURIA) del departamento de Señales Digitales del Centro Geoinformática y Señales Digitales (GEySED). Imparte las asignaturas optativas Procesamiento de Imágenes Digitales I y II.

Correo: fechemendia@uci.cu

Ing. Reinier Pupo Ruiz.

Graduado de la Universidad de las Ciencias Informáticas (UCI) en el año 2009. Es Instructor Recién Graduado y ha impartido clases de Programación 2 en la Facultad 7 de la UCI. Pertenece al Departamento Señales Digitales del Centro de Geoinformática y Señales Digitales (GEySED) de la Facultad 6. Pertenece al proyecto Video Vigilancia (SURIA), donde es el responsable del módulo Video Sensores.

Correo: rpupo@uci.cu

RESUMEN

Con el progresivo aumento de la tecnología digital en cuanto a almacenamiento y transmisión por red de la información visual, la presencia de numerosas cámaras de seguridad en cualquier entorno urbano es un hecho, los sistemas de vigilancia han evolucionado atravesando diferentes etapas en las últimas décadas, las cuales son divididas en tres generaciones. La primera generación de sistemas de vigilancia emplea señales y transmisión analógicas, la segunda se basa, en métodos de procesamiento y comunicación híbridos analógico-digitales, o completamente digitales y la tercera generación ofrece detección de intrusos adaptando y utilizando recursos existentes para transformarlos en un sistema inteligente.

En el Centro de Geoinformática y Señales Digitales (GEySED) se está desarrollando un sistema de Video Vigilancia que actualmente es llevado a cabo, en su totalidad, por un operario el cual debe estar atendiendo varias cámaras y procesando una gran cantidad de información de video. Por lo que la presente investigación tiene como objetivo general desarrollar un video sensor que procese los flujos de video obtenidos de cámaras IP, permitiendo crear y controlar perímetros virtuales.

Este documento recoge un estudio sobre sistemas de video sensores que emplean el control de perimetrado virtual. En el mismo quedan plasmadas las características de las herramientas usadas, como la metodología de desarrollo, la herramienta CASE, el lenguaje de programación, la librería y el IDE (Entorno de Desarrollo Integrado) utilizado.

Asimismo se realizó una investigación que arrojó como resultado los algoritmos más factibles para la implementación del video sensor. Además se recoge la validación de la solución mediante los casos de prueba. Se obtuvo como resultado un video sensor para el control de perimetrado virtual, logrando crear y controlar un entorno de monitoreo, perfeccionando así el sistema de Video Vigilancia.

PALABRAS CLAVES

Cámaras IP, monitoreo, video sensor, Video Vigilancia.

ABSTRACT

With the progressive increase of digital technology in terms of visual information storage and transmission through a network, the presence of numerous security cameras in any urban environment has been noticed. Surveillance systems have evolved through different stages in the last decades which are divided into three generations. The one that uses analog signals and transmission, the system based either in hybrid analog-digital processing and communication, or completely digital methods, and the one that offers intrusion detection by adapting and using existing resources to transform them into an intelligent system.

A video-surveillance system is being developed by the Geo-informatics and Digital Signals Center (GEySED); such system is currently being carried out, as a whole, by an operator which must be attending several cameras and processing a large amount of video information.

This fact addressed us to define as a general objective of this work to develop a video sensor that processes video flows coming from cameras IP, allowing the creation and monitoring of virtual perimeters. This document contains a study on systems of video sensors employing virtual perimeter control at the national and international levels.

This document contains a study on systems of video sensors employing virtual perimeter control. Furthermore, the characteristics of the used tools, such as, the methodology of development, CASE tool, programming language, the library and the IDE (Integrated Development Environment) are also embodied.

An investigation that found the most feasible algorithms for the implementation of the video sensor was also conducted. Also includes the validation of the solution through the test cases. As a result a video sensor to control of the virtual perimeter was obtained, managing to create and control a monitoring environment, perfecting in this way the Video Surveillance System.

KEY WORDS: Cameras IP, monitoring, video sensor, Video-Surveillance.

Índice

Índice de figuras

Índice de tablas

Introducción

Con el avance de la tecnología digital, en cuanto a captura, codificación, almacenamiento y transmisión por red de la información visual, se ha disparado exponencialmente la implementación de sistemas de seguridad basados en cámaras digitales, que junto al desarrollo de potentes herramientas de software logran un avance extraordinario en las prestaciones y capacidades de los sistemas de video vigilancia.

A la par que los riesgos de seguridad aumentan, la necesidad de controlar visualmente y registrar los acontecimientos se ha vuelto aún más importante. Por otra parte, el valor de la actividad de video vigilancia ha crecido de manera significativa con la introducción de los videos sensores.

Desde el punto de vista tecnológico, los sistemas de vigilancia han evolucionado atravesando diferentes etapas en las últimas décadas.

La primera generación de sistemas de vigilancia basada en video emplea señales y transmisión analógicas. En la toma de decisiones, siempre es el operador humano el encargado de realizar todas las tareas de análisis de secuencias de video presentadas en varios monitores situados en una sala de control remoto, donde las escenas monitoreados por las distintas cámaras se multiplexan y se presentan en un orden periódico y predefinido. Adicionalmente, la vigilancia por video tradicional precisa gran cantidad de espacio de almacenamiento. Todo lo que captura una cámara de seguridad, o se guarda en un archivo de video o se sobrescribe periódicamente. Este procedimiento limita la duración del video que puede guardarse y hace que el tiempo necesario para su revisión sea elevado. (Colomer, 2003)

La segunda generación de sistemas de vigilancia se basa, principalmente, en métodos de procesamiento y comunicación híbridos analógico-digitales, o completamente digitales. Aprovechan la flexibilidad ofrecida por los primeros algoritmos de procesado de video que permiten centrar la atención del operador humano en un grupo de situaciones de interés y además, las facilidades proporcionadas por los primeros métodos de compresión digital para aprovechar el ancho de banda de transmisión. (Colomer, 2003)

Actualmente, se está produciendo una migración de los sistemas de video clásicos a los sistemas de tercera generación, estos aprovechan el progreso de las redes de ordenadores de bajo coste y alto rendimiento, y las comunicaciones multimedia fijas y móviles. La

investigación en este campo trabaja en técnicas distribuidas de procesamiento de video. Esta *"tercera generación"* de soluciones de video ofrece detección de intrusos adaptando y utilizando recursos existentes para transformarlos en un sistema inteligente, permitiendo que cada cámara tenga su propio procesamiento de la información, asegurando de esta manera la calidad y confiabilidad de la información brindada.

Con esta tecnología, el análisis de la escena se basa en exploración en 3D, diferenciándose así de las 2 dimensiones con las que trabajan otros tipos de sistemas y aprendiendo desde la perspectiva del video, transformando cualquier cámara en un detector inteligente y proactivo. Los denominados sistemas de procesamiento inteligente de video surgen como respuesta a la necesidad de proveer mecanismos de análisis para la detección automática y en tiempo real de situaciones concretas que afectan a la seguridad como, por ejemplo, intrusos, vehículos mal estacionados, bultos abandonados (explosivos), sustracción de objetos (robos), reconocimiento de patentes y protección perimetral. (Colomer, 2003)

En las empresas e instituciones cubanas existen gran cantidad de recursos que deben ser protegidos por la importancia que los mismos confieren para el desarrollo del país. La Universidad de las Ciencias Informática (UCI) cuenta con una infraestructura tecnológica amplia, por lo que debe disponer de un sistema de seguridad que garantice su protección.

En el Centro de Geoinformática y Señales Digitales (GEySED) se está desarrollando un sistema de Video Vigilancia que cuenta con los siguientes módulos: Visor, Grabador, Recuperador, Gestor, Módulos de Diseño Web y se le está incorporando un nuevo módulo de video sensores que permitirá disponer de funcionalidades como detección de objetos abandonados, seguimiento y estimación de velocidad de vehículos, conteo de personas y protección perimetral o alambrado virtual.

Precisamente en este último aspecto está centrada la presente investigación debido a que la Video Vigilancia actual es llevada a cabo, en su totalidad, por un operario, el cual debe estar atendiendo varias cámaras, lo que se hace engorroso ya que tiene que procesar una gran cantidad de información de video que se le muestra en uno o varios monitores. Además es bien conocido que el operario al vigilar varios escenarios simultáneamente y por varias horas seguidas puede que su atención disminuya y por tanto, aumente la probabilidad de pasar por alto situaciones peligrosas.

Teniendo en cuenta lo mencionado anteriormente se define el siguiente **problema a resolver**: Incapacidad del operario del sistema de Video Vigilancia para controlar eficientemente los entornos en los que se realiza el monitoreo.

Teniendo como **objetivo general** de la investigación: Desarrollar un video sensor que procese los flujos de video obtenidos de cámaras IP, permitiendo crear y controlar perímetros virtuales.

Para darle solución a este problema se define como **objeto de estudio** las técnicas de procesamiento de flujos de videos digital.

El resultado de la investigación tendrá como **campo de acción** los videos sensores para el control de perimetrado virtual en flujos de videos obtenidos de cámaras IP.

Dando lugar a definir como **idea a defender** que si se desarrolla un video sensor para el control de perimetrado virtual, se podrá controlar los entornos de monitoreo y perfeccionar el sistema de Video Vigilancia.

Durante el desarrollo de esta investigación, y para dar respuesta al problema científico planteado, se aplicarán los siguientes métodos científicos:

Métodos teóricos:

Histórico-Lógico: Mediante el cual se analizará la trayectoria y desarrollo de los videos sensores de control de perimetrado virtual, así como las etapas principales por las que han trascurrido dichos sistemas, para de esta forma saber la tendencia que tendrán los mismos en un futuro.

Analítico-Sintético: Con su utilización se podrá estudiar el problema con mayor profundidad para dar la solución adecuada al problema científico. De esta forma se facilitará el entendimiento de los videos sensores para el control de perimetrado virtual y se descubrirá sus principales características así como las generales.

Métodos empíricos:

Observación: Permitirá realizar valoraciones y obtener información a partir de la observación. Este método es de suma importancia puesto que permite observar mediante el registro visual lo que ocurre en la situación real que se analiza, permitiendo así un control adecuado, realizándose de forma reiterada y en distintos momentos.

Para el cumplimiento del objetivo planteado se proponen las siguientes tareas de investigación:

- Caracterizar los sistemas existentes a nivel nacional e internacional.

- Definir las herramientas y tecnologías que se utilizarán para el desarrollo del video sensor de control de perimetrado virtual.
- Elaborar la documentación que contiene las especificaciones de cada una de las fases del desarrollo del software.
- Implementar el video sensor para el control de perimetrado virtual.
- Validar el video sensor para el control de perimetrado virtual.

El trabajo consta de tres capítulos donde se realizará un estudio exhaustivo de las tecnologías que se escogerán para hacer el sistema. Se plantearán las características del sistema comenzando por una vista de los procesos del negocio existentes; se diseñará el sistema propuesto y luego se implementará cumpliendo así con los principales flujos del ciclo de desarrollo.

- En el Capítulo 1, se realizará la fundamentación teórica que justifica la investigación, se analizará el estado actual del tema a tratar a nivel nacional e internacional así como las nuevas tendencias, las tecnologías y metodologías que se usarán en la solución.
- En el capítulo 2, se definen las clases de análisis y diseño de los casos de usos; así como los diagramas de secuencias cumpliendo con las descripciones de los casos de uso. Se presenta la arquitectura del sistema, describiendo además las clases entidades y las controladoras del flujo de trabajo análisis y diseño.
- En el capítulo 3, se explicará en detalles los diferentes algoritmos que se emplearán en la realización del video sensor para el control de perimetrado virtual.
- En el capítulo 4, se presentan los modelos definidos en RUP como diagrama de componentes, objetivo primordial de la fase implementación, así como la validación de la solución propuesta a través de las pruebas que se le realizarán al software.

Capítulo1: Fundamentación teórica.

4.4 Introducción

En este capítulo, se exponen conceptos relacionados con el dominio del problema, se realiza un análisis detallado sobre los sistemas de video sensores existentes a nivel nacional e internacional que utilizan específicamente control de perimetrado virtual, así como las herramientas, tecnologías y metodologías que se usarán en la solución del sistema.

1.2 Conceptos asociados al dominio del problema

Control de perimetrado virtual.

La protección perimetral o alambrado virtual es una variante de la detección de intrusos que activa una alarma cuando una persona o un vehículo traspasa una línea o zona demarcada previamente. La detección se puede especificar para prohibir cualquier cruce o para permitir el movimiento en una sola dirección.

El mismo no hace caso del movimiento en paralelo a las líneas especificadas y detecta solamente si se cruzan las líneas. Es ideal para detectar las personas que pasan a través de una cerca, un punto de control de acceso, o el perímetro de un edificio. Para los vehículos se puede detectar cuando paran en una zona de estacionamiento temporal o cuando cruzan un límite físico de seguridad.

Un **video sensor** no es más que una herramienta de análisis de video digital que ofrece información significativa proveniente de una secuencia de video. El desarrollo de video sensores tiene su base en el procesamiento digital de la imagen. (Colomer, 2003)

Una **cámara IP** (que también recibe el nombre de cámara de red) puede describirse como una cámara y un ordenador combinados para formar una única unidad inteligente. Captura y envía video en directo directamente a través de una red IP, como una LAN, Intranet o Internet, permite a los usuarios ver o gestionar la cámara con un navegador Web estándar o con software de gestión de video en cualquier equipo local o remoto conectado a una red. Permite a usuarios autorizados de distintas ubicaciones acceder simultáneamente a las imágenes captadas por la misma cámara IP de red. (Electronic Dreams)

1.3 Sistemas existentes a nivel internacional

1.3.1 ObjectVideo

ObjectVideo es el principal proveedor de software de procesamiento inteligente de video para la seguridad, control de la inteligencia empresarial, mejora de procesos y otras aplicaciones. Basándose en la tecnología patentada de visión por ordenador, los productos de ObjectVideo convierten el video en datos para la detección y clasificación de objetos que resulten de interés, según las normas definidas por los usuarios.

Los productos de ObjectVideo se comercializan y venden a través de una red mundial de socios, integradores y principales compañías tecnológicas de información en mercados que incluyen la seguridad nacional, el transporte, la banca y la educación.

1.3.2 Axis Cross Line Detection de Axis Communications

AXIS Cross Line Detection es una aplicación del sistema de detección de intrusiones que se instala en cámaras de red de Axis. La aplicación detecta los objetos en movimiento que cruzan una línea virtual, lo que hace posible que se notifiquen automáticamente los incidentes. Cross Line Detection aumentará la eficacia del sistema, el ancho de banda, el almacenamiento y facilitará la búsqueda de eventos grabados. (Axis Communications)

1.5.1 Vaelsys

Vaelsys es una empresa especializada en soluciones de visión artificial. Su tecnología abarca las distintas ramas de conocimiento relacionadas con el análisis de imagen: tratamiento, reconocimiento y clasificación. La experiencia y especialización de la empresa la convierten en un socio idóneo para proyectos relacionados con detección y reconocimiento de imagen o video (intrusión, objetos abandonados y conteos).

1.3.4 Video Analytics de la empresa Intekio

Intekio desarrolla software inteligente de video para la movilidad, la inteligencia de negocios, la mejora de procesos, la seguridad y otras aplicaciones especiales. La innovación de esta empresa consiste en convertir las cámaras de fotos y video ordinarias en sistemas de información inteligente que permiten, en tiempo real, facilitar la toma de decisiones. La misma es pionera en el desarrollo de soluciones de reconocimiento móvil de imágenes. (Intekio)

Video Analytics es una solución de dicha empresa, la misma cuenta con las siguientes funcionalidades:

- Detección de Intrusos.
- Protección perimetral o alambrado virtual (tripwire).
- Detección de robo y remoción de objetos.
- Objetos abandonados y vehículos detenidos.
- Seguimiento de objetos y flujo contrario.

1.4 Sistemas existentes a nivel nacional

Cuba está empezando adentrarse en todo lo relacionado con video vigilancia y específicamente los videos sensores. Existe una solución de video vigilancia desarrollada por DATYS que cuenta con diversas funcionalidades pero todavía no se cuenta con un video sensor que realice funciones de control de perimetrado virtual.

1.4.1 XYMA SAFE VISION

XYMA SAFE VISION es un producto desarrollado por DATYS, empresa que nace desde ACITED informática en el año 2005 y adquiere personalidad jurídica propia el 16 de abril del 2007. El producto XYMA SAFE VISION es un software de video vigilancia profesional basado en tecnología IP, con un alto grado de modularidad, adaptable a una gran cantidad de entornos, flexible, escalable y que admite también el uso de tecnologías analógicas. Permite mantener la seguridad monitoreando y controlando en tiempo real y de forma histórica cada uno de los movimientos que ocurren en las áreas sensibles que se identifiquen.

Principales ventajas de XYMA SAFE VISION.

- Permite ver y grabar múltiples cámaras continuamente.
- Implementa la posibilidad de trabajo con bajo ancho de banda mediante el uso de diversos formatos de video y el ajuste del tamaño de la imagen, de la cantidad de cuadros por segundo y la compresión.
- Arquitectura modular, que le permite adaptar su configuración, según las exigencias de los escenarios de despliegue.
- Utiliza componentes estándares (servidores, switches y cámaras).
- Independencia del hardware dando la oportunidad de elegir el más adecuado a sus propósitos.
- Opera una amplia gama de cámaras de diversas prestaciones y precios.
- Permite la interacción con otros sistemas incluso de tipo analógico.
- Control de acceso y asignación de permisos a los usuarios.
- Realiza trazas de las incidencias del sistema y de su uso.

- Sistema de licencia única que incluye a todos los dispositivos.
- Módulo de Alertas que actúa como un vigilante, chequeando el funcionamiento de los módulos del sistema, y generando avisos a los usuarios y administradores de los eventos detectados. (Datys, 2010)

Los sistemas que se mencionaron anteriormente cuentan con disímiles funcionalidades para la video vigilancia utilizando videos sensores. En todo el mundo se está abogando por el software libre y es esta precisamente la estrategia que está siguiendo la UCI y el país en general. Estos sistemas están desarrollados en plataformas propietarias, lo que implica que queden descartados para su utilización. Además se está tratando de desarrollar un sistema de video vigilancia de tercera generación para comercializarlo y que aporte ingresos al país lo que implica procesamiento inteligente de video.

1.6 Principales herramientas y tecnologías a utilizar

1.5.1 Lenguajes de Programación

Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen un lenguaje informático. El lenguaje de programación permite a un programador especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje léxico. En la actualidad existen disímiles lenguajes de programación cada uno de ellos con características que lo distinguen, a continuación se exponen algunos de ellos como posible lenguajes a utilizar.

- **C++**

C++ es un lenguaje imperativo orientado a objetos derivado del C. Es una mejoría sobre muchas de las características de C, y proporciona capacidades de P.O.O. (Programación Orientada a Objeto) que promete mucho para incrementar la productividad, calidad y reutilización del software.

Este lenguaje está muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción.

C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de

aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original.

La incorporación de la librería STL años más tarde, obra de Alexander Stepanov y Adrew Koening proporciona a C++ una potencia única entre los lenguajes de alto nivel.

No es un lenguaje totalmente orientado a objetos, esto es positivo desde el punto de vista que deja al programador la posibilidad de tomar lo mejor de los dos mundos. Este lenguaje es muy eficaz en cuanto a rapidez y uso de memoria en las aplicaciones que se obtienen. Las bibliotecas estándar de C++ proporcionan un conjunto extenso de capacidades de entrada/salida. En el diseño del C++ primó sobre todo la velocidad de ejecución del código.

- **C#**

C# es un lenguaje moderno y altamente expresivo que se ajusta al paradigma de programación orientada a objetos. Su sintaxis es similar a C++ y Java. El lenguaje fue desarrollado en gran parte por Anders Hejlsberg (creador del mítico compilador Turbo Pascal1 y uno de los diseñadores líder del lenguaje de programación Delphi). En C# no existe el concepto de función global o variable fuera de una clase u objeto.

Por su buen apego a la Programación Orientada a Objetos (POO), es posible sobrecargar métodos y operadores. Soporta definición de interfaces. Permite además la declaración de propiedades, eventos y atributos (que son construcciones declarativas). La principal desventaja de C# es que se desarrolla bajo la plataforma .Net la cual sólo está disponible para la familia Windows. Dicha plataforma es de código cerrado, no hay licencias libres y la infraestructura para desarrollar en .NET representa un alto costo para las empresas. Aunque también se puede desarrollar utilizando Mono en ambientes Windows, GNU/Linux y Mac Os X.

- **Java**

Java es un lenguaje potente orientado a objetos y multiplataforma desarrollado por Sun Microsystems a principios de los años 90. El mismo proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Consta con una librería de clases base, pero para aplicaciones de envergadura se requiere de Frameworks externos, los cuales son en su mayoría propietarios o libres que están en desarrollo por una comunidad de programadores y cuentan con escasa documentación. Este lenguaje es soportado por IDEs de programación libres pero tiene la desventaja que para poder ejecutar las sentencias se tienen que importar máquinas virtuales lo que conlleva a una compilación muy lenta. Las

aplicaciones resultantes requieren de gran cantidad de memoria en las computadoras clientes.

- **C++ como solución propuesta para la implementación del sistema.**

A través de un análisis minucioso de los antes expuestos lenguajes de programación se tomó como lenguaje de programación para la implementación del sistema el lenguaje C++ debido a las características siguientes:

- Difusión: al ser uno de los lenguajes más empleados en la actualidad, posee un gran número de usuarios y existe una cantidad inmensa de libros, cursos y páginas web dedicadas a él.
- Portabilidad: el lenguaje está estandarizado y un mismo código fuente se puede compilar en diversas plataformas.
- Eficiencia: C++ es uno de los lenguajes más rápidos en cuanto a ejecución lo cual es de vital importancia a la hora de trabajar con flujos de video.
- Herramientas: existe una gran cantidad de compiladores, depuradores, librerías y se integra muy bien con Qt.

1.5.2 Metodología de Desarrollo

- **Proceso Unificado de Desarrollo (RUP)**

El proceso unificado del software es la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos. El mismo es un proceso que define quien está haciendo qué, cuándo y cómo alcanzar un determinado objetivo. Es muy efectivo y proporciona normas para el desarrollo eficiente de software de calidad. Está basado en componentes, lo que significa que el sistema de software en construcción está formado por componentes de software, interconectados a través de interfaces bien definidas. Sus principales características son:

Dirigido por casos de uso: Los casos de uso guían el proceso de desarrollo, no se desarrollan aisladamente. Además guían el diseño, implementación y prueba son los que indican cómo debe actuar el sistema con el usuario final o con otro sistema para conseguir su objetivo.

Centrado en la arquitectura: Los modelos son proyecciones del análisis y el diseño lo cual constituye la arquitectura del producto a desarrollar. La arquitectura es una vista del diseño completo con las características más importantes resaltadas.

Iterativo e incremental: A medida que avanza el proceso de desarrollo se producen versiones incrementales, las cuales se acercan cada vez más al producto terminado. En cada iteración los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componente y verifican que los componentes satisfacen los casos de uso.

- **XP**

XP (Extreme Programing) nace como nueva disciplina de desarrollo de software hace aproximadamente unos seis años, y ha causado un gran revuelo entre el colectivo de programadores del mundo. XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo.

XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Las características esenciales de XP son las siguientes: historias de usuario, roles, proceso y prácticas.

- **RUP como metodología de desarrollo propuesta para la implementación del sistema.**

Después de la realización de un análisis detallado de las metodologías RUP y XP se determinó que la más apropiada para modelar el sistema es RUP, por las siguientes razones:

- RUP es una metodología que tiene la particularidad de que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.
- RUP no necesita que el cliente forme parte del equipo de desarrollo al contrario de XP y en este caso es muy beneficioso ya que no hay un cliente específico para el sistema.
- RUP realiza las pruebas al final del producto al contrario de XP que está constantemente realizando pruebas.
- La gran cantidad de artefactos que se generan en RUP contribuyen a un mejor entendimiento del problema.

- Es una metodología con alta adaptabilidad a las condiciones reales del desarrollo del sistema. Es decir que se puede hacer más ágil según se necesite.

1.5.3 Herramienta CASE

Las herramientas CASE proporcionan un conjunto de instrumentos semiautomatizados y automatizados que brindan ayuda y dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software.

- **Rational Rose**

Rational Rose es una herramienta de diseño orientada a objetos, que da soporte al modelado visual, es decir, que permite representar gráficamente el sistema, permitiendo hacer énfasis en los detalles más importantes, centrándose en los casos de uso y enfocándose hacia un software de mayor calidad, empleando un lenguaje estándar común que facilita la comunicación.

Proporciona mecanismos para realizar la ingeniería inversa, es decir, que a partir del código se pueda obtener información sobre su diseño; adicionalmente permite generar código en diferentes lenguajes a partir de un diseño en UML. Brinda la posibilidad de que varias personas trabajen a la vez, permitiendo que cada desarrollador opere en un espacio de trabajo privado que contiene el modelo completo y permite que tenga un control exclusivo sobre la propagación de los cambios en ese espacio de trabajo. Rational además, soporta los diagramas de UML, excepto los Diagramas de Implementación.

- **Visual Paradigm**

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Presenta la desventaja de que requiere bastante RAM debido a estar desarrollado en JAVA y tiene problemas de integración con otras herramientas de desarrollo.

- **Enterprise Architect**

Enterprise Architect es una herramienta de diseño y análisis UML, que cubre el desarrollo de software desde la captura de requerimientos a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento. Esta herramienta es multi-usuarios, diseñada para

ayudar a construir software robusto y fácil de mantener. Además, ofrece salida de documentación flexible y de alta calidad, velocidad, estabilidad y rendimiento. También provee trazabilidad completa desde el análisis de requerimientos y los artefactos de diseño, a través de la implementación y el despliegue. Usa perfiles UML para extender el dominio de modelado, mientras que la validación del modelo asegura integridad. Entre las principales características que este brinda se pueden mencionar:

- ✓ Crea elementos del modelo UML para un amplio alcance de objetivos.
- ✓ Ubica esos elementos en diagramas y paquetes.
- ✓ Documenta los elementos que ha creado.
- ✓ Genera código para el software que está construyendo.
- ✓ Realiza ingeniería directa e inversa de código en lenguajes como ActionScript, C++, C#, Java y PHP.

- ➢ **Enterprise Architect 7.0 como herramienta propuesta para la modelación del sistema.**

Luego de una investigación sobre algunas de las herramientas case se decidió que la más factible para realizar el sistema es el Enterprise Architect (EA) ya que es una herramienta completa de análisis y diseño UML, que cubre el desarrollo de software desde la concepción de las exigencias, a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento. Permite gestionar todos los aspectos importantes del diagrama, sus clases, métodos, atributos, estereotipos y modo de acceso. EA está diseñada para construir software robusto y conservable. Además despliega documentación de salida flexible y de alta calidad. Puede modelar procesos de negocio, sitios web, interfaces de usuario, redes, configuraciones de hardware y mensajes. Estima el tamaño del proyecto en esfuerzo de trabajo en horas. Captura y traza requisitos, recursos, planes de prueba, solicitudes de cambio y defectos.

1.5.4 El Lenguaje Unificado de Modelado (UML) como soporte de la modelación de la solución propuesta

UML es ante todo un lenguaje que proporciona un vocabulario y unas reglas para permitir una comunicación. Es un conjunto de herramientas, que permite modelar (analizar y diseñar) sistemas orientados a objetos. El mismo se centra en la representación gráfica de un sistema y nos indica cómo crear y leer los modelos.

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

1.5.5 IDE (Entorno de Desarrollo Integrado) de Desarrollo

- **Qt Creator**

Qt Creator es un IDE (Entorno de desarrollo integrado) creado por Trolltech para el desarrollo de aplicaciones con las bibliotecas Qt. Soporta sistemas operativos como GNU/Linux, Mac OS X, Windows XP y Vista, por lo que es multiplataforma y software libre. Permite construir interfaces de usuario complejas de una forma visual y rápida ya que incluye un editor de texto con autocompletado, diseñador de interfaces gráficas, gestión de proyectos, sistema de depuración e integración con sistemas de control de versiones.

Qt Creator se integra bien con el C++ usando las librerías de Qt, abastece no sólo a los desarrolladores que están acostumbrados a utilizar el ratón, sino también a los que se sienten más cómodos con el teclado, con una amplia gama de métodos abreviados de teclado y navegación, que están disponibles para ayudar a acelerar el proceso de desarrollo de una aplicación. Es distribuido bajo tres tipos de licencias: Qt Commercial Developer License, Qt GNU LGPL v. 2.1, Qt GNU GPL v. 3.0.

- **KDevelop**

El Proyecto KDevelop surgió en 1998 con el fin de desarrollar un IDE fácil de usar para KDE. Desde entonces, el IDE KDevelop está públicamente disponible bajo la licencia GPL y soporta varios lenguajes de programación. A diferencia de muchas otras interfaces de desarrollo, KDevelop no cuenta con un compilador propio. Cuenta con editor de código fuente con indentado automático (Kate) y la gestión de diferentes tipos de proyectos, como CMake, Automake, qmake (para proyectos basados en la biblioteca Qt y Ant, para proyectos basados en Java). También proporciona completado automático del código en C y C++ y asistentes para generar y actualizar las definiciones de las clases y el Framework de la aplicación.

- **Qt Creator 2.0.1 como IDE propuesto para la solución del sistema.**

Para la implementación del sistema se decidió usar como IDE el Qt Creator debido a las múltiples ventajas que ofrece el mismo para la programación con C++, las cuales se exponen a continuación:

- Cuenta con una interfaz muy cómoda, que es funcional y estéticamente agradable.
- Tiene plugins para varios sistemas de control de versiones como Perforce y Subversion.
- Tiene una herramienta de búsqueda eficaz donde se puede buscar fácilmente las clases, métodos y archivos.
- El Qt Creator cuenta con el QtDesigner que ayuda a diseñar formas de interfaz de usuario e incluye también la gestión y finalización del proyecto de código.

1.5.6 Librería a utilizar para el desarrollo del sistema, Opencv 2.1.0

OpenCV (Open Source Computer Vision Library) es una librería de funciones en C y C++ desarrollado por Intel que proporciona un alto nivel de funciones para el procesado de imágenes, visión artificial, captura de video y visualización de imágenes. Es de código abierto, gratuita, multiplataforma (disponible para entornos MS Windows, Mac OS y Linux), está desarrollada bajo la licencia BSD (Distribución de Software Berkeley), rápida, de fácil uso y en continuo desarrollo. Estas librerías permiten a los programadores crear aplicaciones poderosas en el dominio de la visión digital.

OpenCV implementa una gran variedad de herramientas para la interpretación de imagen. Es compatible con Intel Image Processing Library (IPL) que implementa algunas operaciones en imágenes digitales. La misma implementa algoritmos para las técnicas de la calibración (Calibración de la Cámara), detección de rasgos, para rastrear (Flujo Óptico), análisis de la forma (Geometría, Contorno que Procesa), análisis del movimiento (Plantillas del Movimiento, Estimadores), reconstrucción 3D (Transformación de vistas), segmentación de objetos y reconocimiento (Histograma).

Los algoritmos están basados en estructuras de datos muy flexibles; más de la mitad de las funciones han sido optimizadas aprovechándose de la arquitectura de Intel. En cuanto a análisis de movimiento y seguimiento de objetos, ofrece una funcionalidad interesante, incorpora funciones básicas para modelar el fondo para su posterior sustracción, generar imágenes de movimiento MHI (Motion History Images) para determinar dónde hubo movimiento y en qué dirección, lo cual es de vital importancia para la realización de un video sensor para el control de perimetrado virtual.

1.6 Conclusiones Parciales

En este capítulo a través de un estudio de la evolución de los sistemas de video sensores se decidió implementar un sistema de video sensores para el control de perimetrado virtual. Para la realización del sistema se usará como metodología de desarrollo RUP, apoyándose en la herramienta case Enterprise Arquitect. También se decidió que el lenguaje más factible para la programación del sistema es el C++ con la ayuda de las librerías de OpenCv, y el IDE QT Creator.

Capítulo 2: Dominio, Requisitos, Sistema, Análisis y Diseño.

2.1 Introducción

En este capítulo, se realiza la propuesta inicial del sistema, estableciendo y describiendo el negocio en el que se enmarca el mismo, así como los requisitos funcionales y no funcionales. Además se puntualizan los casos de uso del sistema identificados, dando un breve resumen de cada uno. También se definen las clases de análisis y diseño de los casos de usos; así como los diagramas de secuencias cumpliendo con las descripciones de los casos de uso. Se presenta la arquitectura del sistema, describiendo además las clases entidades y las controladoras del flujo de trabajo análisis y diseño.

2.2 Modelo de Dominio

Como no está bien determinado el proceso del negocio con fronteras bien establecidas donde se logre ver claramente, quienes son las personas que lo inician, quienes son los beneficiados con cada uno de estos procesos, pero además quienes son las personas que desarrollan las actividades en cada uno de estos procesos, entonces se realizará la modelación del dominio.

El Modelo de Dominio es una representación visual estática del entorno real objeto del proyecto. Es decir, un diagrama con los objetos que existen (reales) relacionados con el proyecto que se va a acometer y las relaciones que hay entre ellos. Pero no son clases de software (aunque algunos objetos del Modelo de Dominio pueden terminar siéndolo). Este se centra en una parte del negocio, la relacionada con el ámbito del proyecto. En este contexto el término "dominio" representa una parte del "negocio".

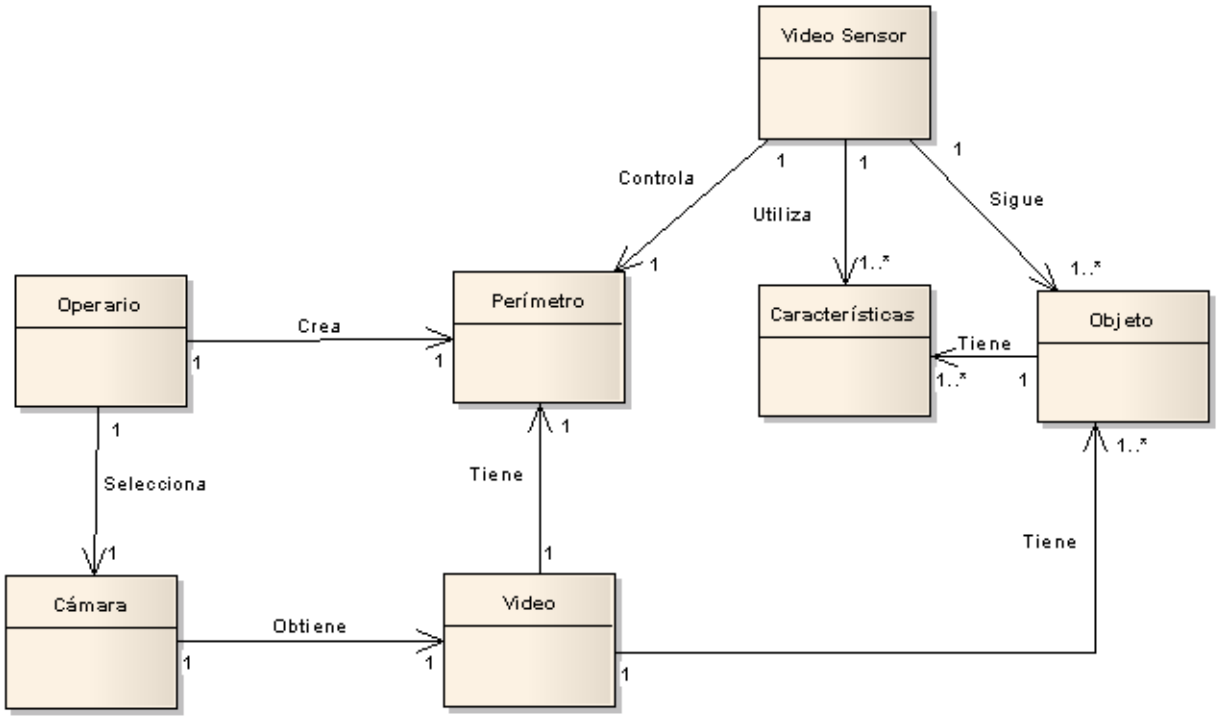

Figura 1: Modelo de Dominio.

2.2.1 Descripción del Modelo de Dominio

En la figura anterior el operario selecciona una cámara, de la cual se obtiene un flujo de video, luego el operario crea un perímetro en el flujo de video. Este perímetro es controlado por el video sensor para cuando los objetos lo traspasen alertar al operario o notificar al sistema para que realice una acción determinada. El video sensor utiliza las características que presenta el objeto y le realiza un seguimiento, para determinar en cualquier instante de tiempo dónde se encuentra ubicado.

2.3 Especificación de los requisitos de software

Lograr una comunicación efectiva entre los usuarios y el equipo de proyecto con el objetivo de llegar a un entendimiento de lo que hay que hacer, es la forma más efectiva de lograr un producto de software con calidad. Es por esto que se le da gran importancia a la identificación de los requisitos como parte del proceso de desarrollo del software. Los requisitos se pueden clasificar en: funcionales y no funcionales.

Requisitos funcionales: Son capacidades o condiciones que el sistema debe cumplir. Los mismos se mantienen invariables sin importar con que propiedades o cualidades se relacionen.

Requisitos no funcionales: Los requisitos no funcionales son propiedades o cualidades que el producto debe tener.

Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. En muchos casos los requisitos no funcionales son fundamentales en el éxito del producto. Normalmente están vinculados a requisitos funcionales, es decir una vez se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser.

2.3.1 Requisitos funcionales

RF1 Crear perímetro en un flujo de video obtenido de una cámara IP.

RF2 Controlar perímetro en un flujo de video obtenido de una cámara IP.

2.3.2 Requisitos no funcionales

- **Portabilidad**
  - ✓ **RNF1**: El video sensor debe funcionar sobre Sistema Operativo Linux y Windows.
- **Restricciones en el diseño e implementación.**
  - ✓ **RNF 3:** El video sensor estará implementado en lenguaje C++, utilizando como IDE de desarrollo QT Creator y como librería para el tratamiento de las imágenes el Opencv.
- **Software.**
  - ✓ **RNF 4:** Se debe tener instaladas las librerías del OpenCv y la librería cvBlob.
- **Hardware**
  - ✓ **RNF 5:** Memoria RAM 1 GB y microprocesador Core Duo a 2.20 ghz.

2.4 Definición de los casos de uso

Los casos de uso representan un medio intuitivo y sistemático para calcular los requisitos funcionales. Mediante los mismos los analistas se ven obligados a pensar en quiénes son los usuarios y qué necesidades u objetivos pueden cumplir. Proporciona la entrada fundamental para el análisis, diseño y las pruebas del sistema. Estos facilitan un medio para

que los desarrolladores, usuarios finales y trabajadores lleguen a una comprensión común del sistema propuesto. Son empleados además para la validación de la arquitectura a lo largo del desarrollo del software.

2.4.1 Descripción de los actores

Un actor es un rol que cumple un usuario, puede intercambiar información o puede ser un recipiente pasivo de información y representa a un ser humano, a un software o a una máquina que interactúa con el sistema.

Cada actor participa en uno o más casos de uso y juega un papel por cada caso de uso con que colabora. Un actor inicia un caso de uso pero una vez que este ha comenzado el caso de uso puede interactuar con varios actores. Interactúa con el caso de uso y por lo tanto con el sistema o la clase que posee el caso de uso, intercambiando mensajes. La implementación interna de un actor no es relevante en el caso de uso; un actor puede ser caracterizado suficientemente por un conjunto de atributos que definen su estado.

Actor del sistema	**Justificación**
Operario	El operario es la persona encargada de inicializar el caso de uso controlar y crear el perímetro en el flujo de video obtenido desde una cámara IP.

Tabla 1: Descripción del actor operario.

2.4.2 Casos de Uso del Sistema

Los casos de uso son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. Un caso de uso especifica una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores. (Jacobson, et al., 2000)

Los casos de uso tienen la peculiaridad de ser tan globales o específicos como se quiera, por lo que en un caso de uso se pueden encapsular varias acciones realizadas por el sistema. Un caso de uso es un proceso que da un resultado de valor para un actor determinado.

CU-1	Crear perímetro
Actor	Operario

Descripción	El caso de uso consiste en que el operario traza las líneas que conformaran el perímetro siendo estas dibujadas sobre el flujo de video. También determina si el punto está dentro del polígono creado.
Referencia	RF1

Tabla 2: Descripción del caso de uso (crear perímetro).

CU-2	Controlar perímetro.
Actor	Operario
Descripción	El caso de uso consiste en que el operario hace la petición de que se controlen perímetros virtuales por lo que el sistema captura un flujo de video y luego extrae de él los fotogramas, modela el fondo y pasa a detectar los objetos que se encuentran en el video.
Referencia	RF2

Tabla 3: Descripción del caso de uso (controlar perímetro).

2.4.3 Diagrama de caso de uso del sistema

Figura 2: Diagrama de caso de uso del sistema.

2.4.4 Descripción de los Casos de Uso

Descripción del CU (Caso de Uso) controlar perímetro en un flujo de video obtenido de una cámara IP.

Caso de Uso:	Controlar perímetro en un flujo de video obtenido de una cámara IP.
Actores:	Operario
Resumen:	El caso de uso inicia cuando el Operario hace la petición de que se

	controlen perímetros virtuales por lo que el sistema captura un video y luego extrae de él los fotogramas, modela el fondo y pasa a detectar los objetos que se encuentran en el video.
Propósito:	Realizar el control de todas las operaciones sobre el video.
Referencias	RF2
Prioridad	Crítico
Flujo Normal de Eventos	
1. El actor solicita el control del perímetro en un flujo de video proveniente de una cámara IP.	2. Se captura el flujo de video proveniente de la cámara.
	3. Los fotogramas son extraídos del flujo de video.
	4. Se modela el fondo y el frente del video.
	5. Es eliminado el ruido.
	6. Se le da seguimiento a los objetos que se encuentran en el video.

Tabla 4: Descripción detallada del caso de uso (controlar perímetro).

Descripción del CU (Caso de Uso) crear perímetro en un flujo de video obtenido de una cámara IP.

Caso de Uso:	Crear perímetro en un flujo de video obtenido de una cámara IP.
Actores:	Operario
Resumen:	El caso de uso inicia cuando el operario con el mouse selecciona la región donde desea que se cree el perímetro, el sistema pinta el perímetro y termina cuando se determina si el objeto está dentro del perímetro.
Propósito:	Pintar un perímetro en un flujo de video.
Referencias	RF1
Prioridad	Crítico
Flujo Normal de Eventos	

1. El caso de uso se inicia cuando el operario selecciona la región que desea monitorear.	2. El sistema crea el perímetro.
	3. Determina si el punto está dentro del polígono.

Tabla 5: Descripción detallada del caso de uso (crear perímetro).

2.5 Análisis y Diseño

2.5.1 Descripción de la arquitectura

La arquitectura representa la estructura de los componentes de un programa o sistema, sus interrelaciones, los principios y reglas que gobiernan su diseño y evolución en el tiempo. (Matias Simarro, 2004)

La misma es la representación de un alto nivel de la estructura de un sistema o aplicación, que describe componentes que lo integran, interacciones entre ellos, patrones que supervisan su composición y restricciones para aplicar dichos patrones.

Un patrón es una solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en el desarrollo de software. (Lleonart Martín, y otros)

Este permite dar solución a un problema en un contexto y reutiliza soluciones a problemas comunes. Son un esqueleto básico que cada diseñador adapta a las peculiaridades de su aplicación.

El video sensor que se está desarrollando, es un sistema con una arquitectura centrada en los flujos de datos, basada en el patrón “pipe and filter” (tuberías y filtros); donde los componentes presentes en el sistema responden a decisiones arquitectónicas para hacer cumplir requerimientos funcionales y no funcionales. Estos componentes están asociados a flujos de datos y refinamientos sucesivos. O sea, un conjunto de elementos denominados “filtros” conectados entre sí por “tuberías” trasmiten datos desde un componente al siguiente.

Cada filtro trabaja de manera independiente de los componentes. Se diseñan de tal modo que esperan un conjunto de datos en un determinado formato y obtiene como resultado otros datos de salida en un formato específico. Este patrón de arquitectura es particularmente efectivo a la hora de descomponer el problema en pasos independientes,

reutilizar filtros, facilitar el mantenimiento, independencia entre filtros y ejecución concurrente de filtros.

2.5.2 Patrones de diseño

Un patrón es una descripción de un problema y su solución (pareja problema/solución), con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas.
Patrones de diseño o más comúnmente conocidos como "Design Patterns". Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Dichas soluciones están basadas en la experiencia y que se ha demostrado que funcionan.

Es evidente que en la implementación de un software hay problemas que se repiten o que son análogos, es decir, que responden a un cierto patrón. Sería deseable tener una colección de dichos patrones con las soluciones más óptimas para cada caso.

Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo; desde el análisis hasta el diseño y desde la arquitectura hasta la implementación.

El grupo de GoF (Pandilla de los cuatro: Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides) clasificaron los patrones en tres grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

Creacionales: Patrones creacionales tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.

Estructurales: Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

Comportamiento: Los patrones de comportamiento ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

Los Patrones Generales de Software para Asignar Responsabilidades (GRASP) son parejas de problema solución con un nombre, que codifican buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades. Describe los

principios de la asignación de responsabilidades de un objeto es por esto que los siguientes patrones se evidencian en la implementación del video sensor:

Patrón Experto

Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen.

Patrón Creador

El cual plantea la necesidad de asignarle a una clase la responsabilidad de crear una instancia de otra clase siempre y cuando agregue los objetos de la clase, los contenga, registre las instancias de estos objetos y los utilice específicamente.

Patrón Bajo acoplamiento

Acoplamiento bajo significa que una clase no depende de muchas clases, ya que uno de los principios para protegerse frente a los cambios es mantener bajo el acoplamiento entre variedades. Resulta evidente que, cuanto menor sea el acoplamiento entre clases, menor influencia tendrán los cambios.

Patrón Controlador

Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones y seguridad). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema.

Alta Cohesión

Expresa que se debe asignar una responsabilidad de modo que la cohesión siga siendo alta, la cohesión no es más que la medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. El patrón de Alta Cohesión, más que un diseño directamente implementable en código, se trata de un principio director que guiará el diseño. Una clase estará más cohesionada cuanto más enfocado sea su comportamiento. Es decir, al asignar responsabilidades en el diseño, se buscará soluciones que asignen los métodos a las clases de forma coherente, completa y relacionada. De esta forma, se obtendrá clases cohesionadas.

2.6 Modelo de análisis

Durante el análisis, se analizan los requisitos que fueron descritos en la captura de requisitos, refinándolos y estructurándolos. Con el objetivo de conseguir mejor comprensión de los requisitos y una descripción de los mismos que ayude a estructurar todo el sistema, incluyendo su arquitectura. Se estructuran los requisitos de un modo que facilita su comprensión, su preparación, su modificación, y en general, su mantenimiento. Se puede considerar como una primera aproximación al modelo de diseño y es por tanto una entrada fundamental cuando se da forma al sistema en el diseño y la implementación.

2.6.1 Diagramas de clases de análisis

Un diagrama de clases del análisis es un artefacto en el que se representan los conceptos en un dominio del problema. Representa el funcionamiento del mundo real, no de la implementación automatizada del mismo.

Diagrama de clases de análisis del CU (Caso de Uso) controlar perímetro en un flujo de video obtenido de una cámara IP.

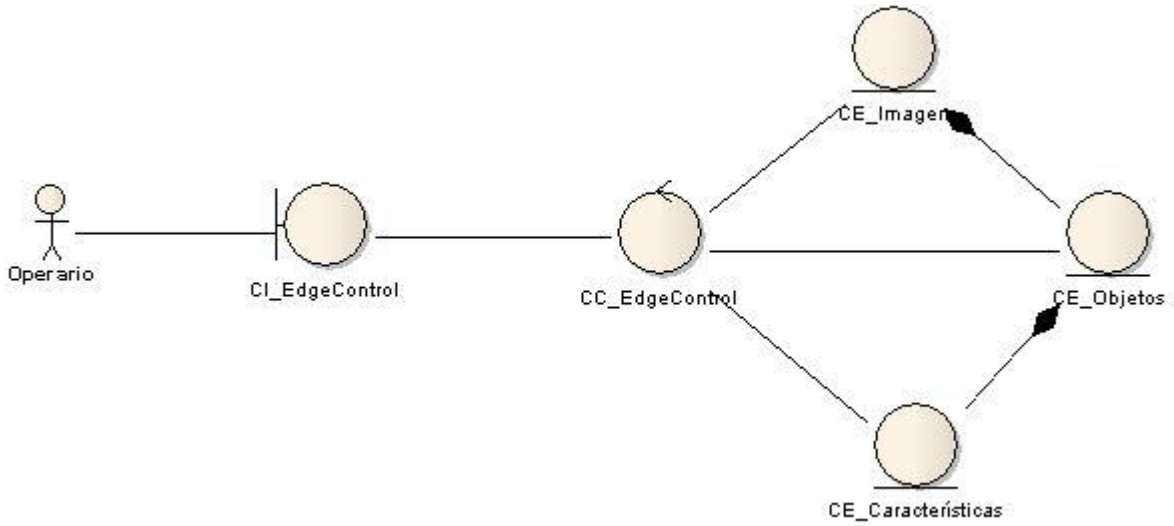

Figura 3: Diagrama de clases del análisis (controlar perímetro).

En la figura 3 se muestra el diagrama de clases del análisis para el caso de uso controlar perímetro, se pueden apreciar las clases entidades CE_Imagen, CE_Características y CE_Objetos con una relación de composición ya que una imagen está compuesta por objetos y estos por características. La clase control CC_EdgeControl es la que maneja todos los pasos que se van a ejecutar para lograr el objetivo propuesto que es controlar perímetros virtuales en un flujo de video obtenido de una cámara IP.

Diagrama de clases de análisis del CU (Caso de Uso) crear perímetro en un flujo de video obtenido de una cámara IP.

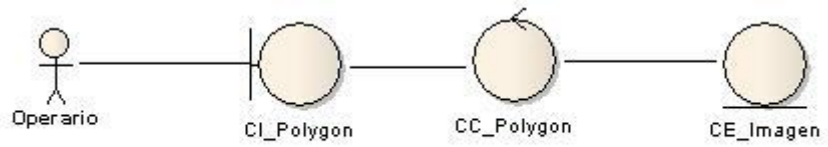

Figura 4: Diagrama de clases del análisis (crear perímetro).

En la Fig. 4 se muestra el diagrama de clases del análisis para el caso de uso crear perímetro. Se muestra la clase interfaz CI_Polygon que es la que se encargará de comunicarse con la clase control CC_Polygon que será la que realizará la creación del perímetro y de determina si el punto está dentro del polígono en el flujo de video obtenido desde una cámara IP.

2.6.2 Diagramas de Interacción

En los diagramas de interacción los objetos interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones. Los diagramas de interacción muestran cómo se comunican los objetos. Representa de forma precisa las interacciones entre los mismos, pero, en esencia, su misión es localizar el comportamiento de los objetos. Existen dos tipos de diagramas de interacción:

- **Diagramas de colaboración.**

Los diagramas de colaboración son útiles en la fase exploratoria para identificar objetos. Permiten representar una disposición espacial de la estructura de los objetos en ejecución. La estructura estática viene dada por los enlaces; la dinámica por el envío de mensajes por los enlaces.

- **Diagramas de secuencia.**

Los diagramas de secuencia muestran la secuencia cronológica de mensajes entre objetos durante un escenario concreto. La vida de cada objeto viene dada por una barra vertical y el tiempo transcurre de arriba hacia abajo.

Diagrama de colaboración para el CU (Caso de Uso) controlar perímetro en un flujo de video obtenido de una cámara IP.

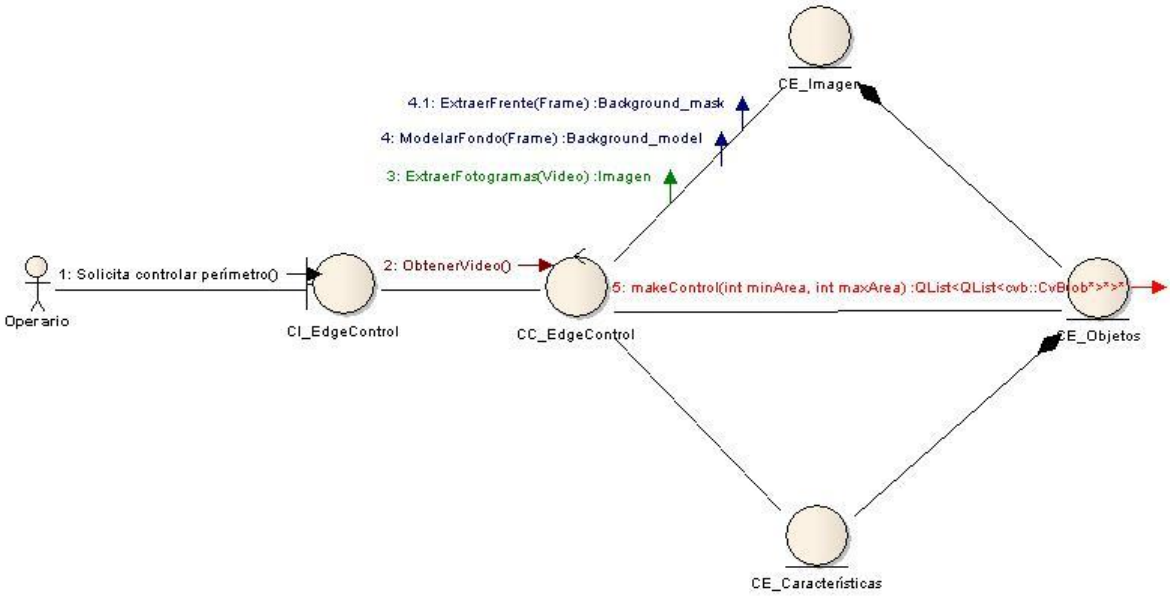

Figura 5: Diagrama de colaboración (controlar perímetro).

En la figura 5 se muestra el diagrama de colaboración para el caso de uso controlar perímetro. Se observan los principales mensajes intercambiados entre las clases para realizar el caso de uso en cuestión.

Diagrama de colaboración para el CU (Caso de Uso) crear perímetro en un flujo de video obtenido de una cámara IP.

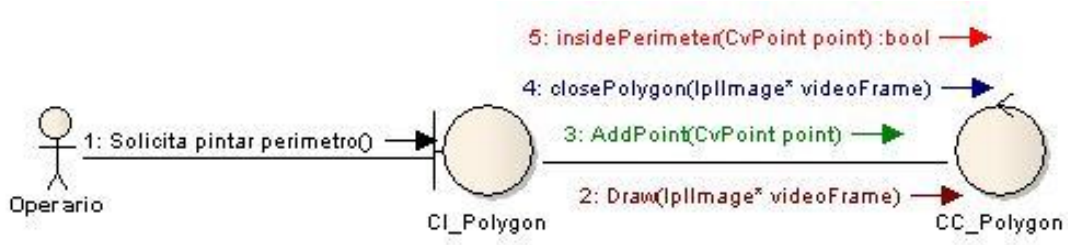

Figura 6: Diagrama de colaboración (crear perímetro).

En la figura 6 se muestra el diagrama de colaboración para el caso de uso crear perímetro. Se observan los principales mensajes intercambiados entre las clases para realizar el caso de uso en cuestión, se obtiene como resultado un perímetro virtual sobre el flujo de video obtenido de una cámara IP.

2.7 Modelo de diseño

En el diseño se modela el sistema para que soporte todo los requisitos, es importante el resultado del análisis ya que el mismo proporciona una descripción detallada de los requisitos y da una estructura del sistema. El mismo es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas en el entorno de la implementación, tienen impacto en el sistema a considerar. Además sirve de abstracción de la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental de las actividades de implementación.

2.7.1 Diagrama de clases del diseño

El diagrama de clases del diseño describe la realización física de los casos de usos, centrándose en como los requisitos funcionales y no funcionales tienen impacto en el sistema. Una clase de diseño, sus objetos y los subsistemas que contienen las clases de diseño, a menudo participan en varias realizaciones de casos de uso. También puede darse el caso de algunas operaciones, atributos y asociaciones sobre una clase específica que son relevantes solo para una realización de caso de uso. Esto es importante para coordinar todos los requisitos que diferentes realizaciones de casos de uso imponen a una clase, a sus objetos y a los subsistemas que contiene.

Diagrama de clases del diseño del CU (Caso de Uso) controlar perímetro en un flujo de video obtenido de una cámara IP.

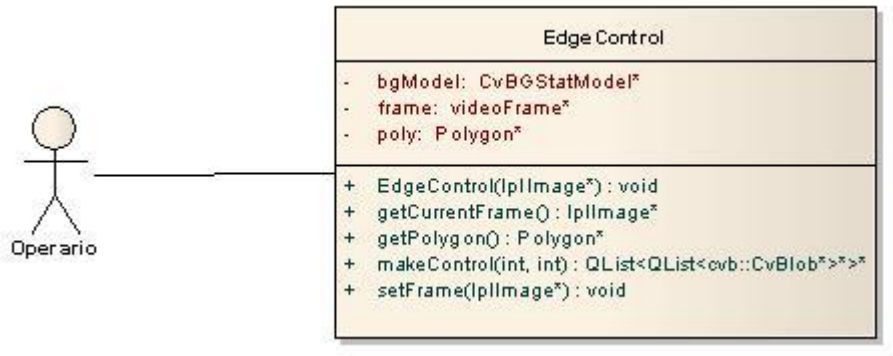

Figura 7: Diagrama de clases del diseño (controlar perímetro).

En la figura 7 se muestra el diagrama de clases del diseño para el caso de uso controlar perímetro. La clase EdgeControl es la encargada de interactuar con el operario y brindarle a este las funcionalidades del componente. Posee los métodos necesarios para cumplir con los requisitos funcionales.

Diagrama de clases del diseño del CU (Caso de Uso) crear perímetro en un flujo de video obtenido de una cámara IP.

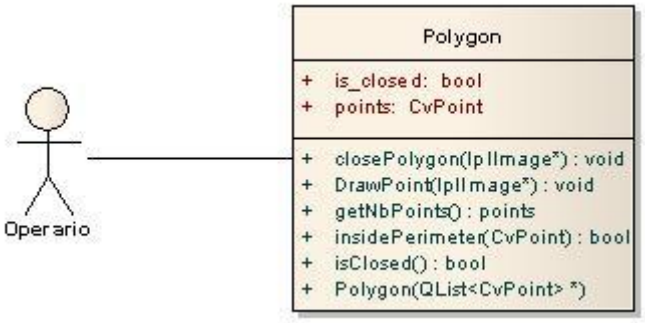

Figura 8: Diagrama de clases del diseño (crear perímetro).

En la figura 8 se muestra el diagrama de clases del diseño para el caso de uso crear perímetro. La clase Polygon es la encargada de interactuar con el operario y brindarle a este las funcionalidades requeridas. Posee los métodos necesarios para la realización del caso de uso en cuestión.

2.7.2 Diagramas de interacción del diseño

Diagrama de secuencia para el CU (Caso de Uso) controlar perímetro en un flujo de video obtenido de una cámara IP.

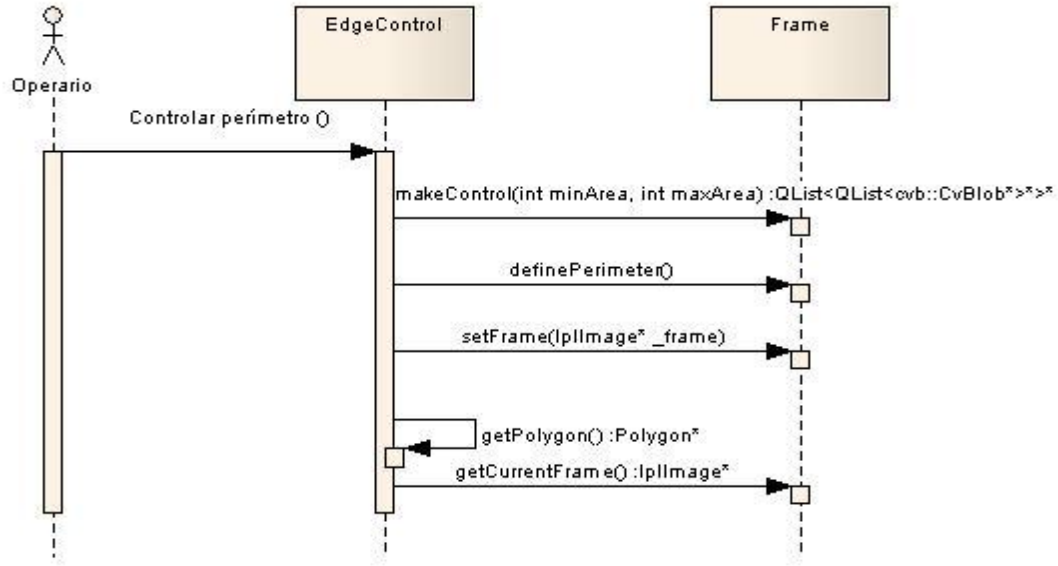

Figura 9: Diagrama de secuencia (controlar perímetro).

En la Fig. 9 se muestra el diagrama de secuencia para el caso de uso controlar perímetro. Se aprecia la interacción en el tiempo entre los objetos y los mensajes enviados entre estos.

El operario interactúa con la clase EdgeControl la cual le ofrece las funcionalidades del video sensor.

Diagrama de secuencia para el CU (Caso de Uso) crear perímetro en un flujo de video obtenido de una cámara IP.

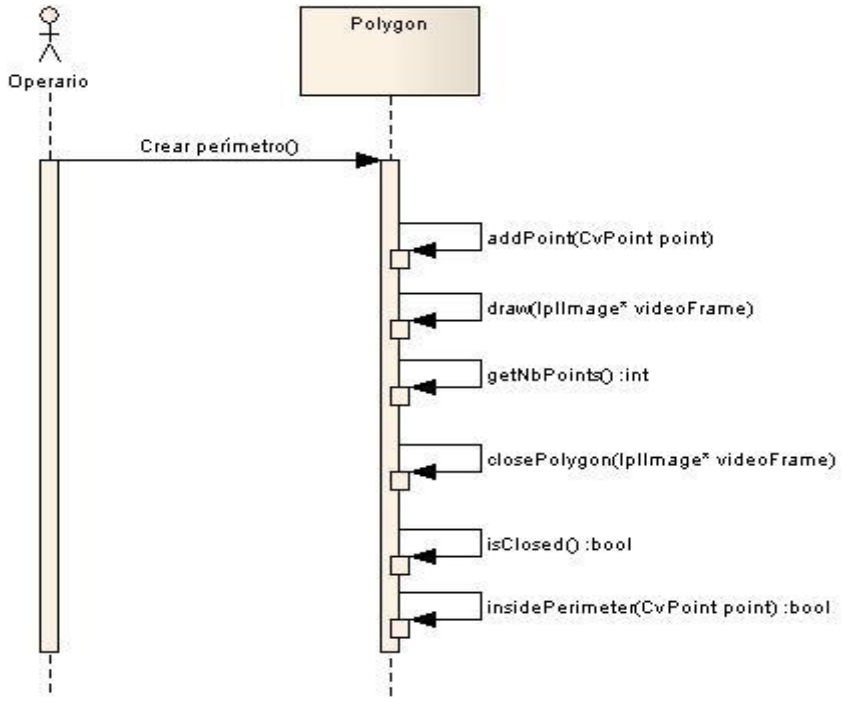

Figura 10: Diagrama de secuencia (crear perímetro).

En la Fig. 10 se muestra el diagrama de secuencia para el caso de uso crear perímetro. Donde se evidencia la interacción en el tiempo entre los objetos a través de mensajes entre el operario y la clase Polygon.

2.7.3 Descripción de las clases

Descripción de las clases del CU (Caso de Uso) controlar perímetro en un flujo de video obtenido de una cámara IP.

Nombre	EdgeControl
Tipo de clase	Controladora
Atributo	**Tipo**
bgModel	CvBGStatModel*
frame	videoFrame*
poly	Polygon*

Para cada responsabilidad	
Nombre	getCurrentFrame
Descripción	Devuelve el frame que está corriendo.
Nombre	getPolygon
Descripción	Devuelve el polígono
Nombre	makeControl
Descripción	Devuelve una lista de blobs que están fuera y dentro del polígono
Nombre	setFrame
Descripción	Cambia el frame actual por el que se le pasa por parámetro.

Tabla 6: Descripción de las clases del caso de uso controlar perímetro.

Descripción de las clases del CU (Caso de Uso) crear perímetro en un flujo de video obtenido de una cámara IP.

Nombre	Polygon	
Tipo de clase	Controladora	
Atributo		**Tipo**
points		QList<CvPoint> *
is_closed		bool
Para cada responsabilidad		
Nombre		insidePerimeter
Descripción		Determina si el punto está dentro del polígono.
Nombre		draw
Descripción		Dibuja el polígono
Nombre		closePolygon
Descripción		Cierra el polígono
Nombre		addPoint
Descripción		Adiciona un nuevo punto a la lista de puntos del polígono.
Nombre		isClosed
Descripción		Devuelve el valor que tiene la variable is_close en ese momento.
Nombre		getNbPoints
Descripción		Retorna la cantidad de puntos del polígono.

Tabla 7: Descripción de las clases del caso de uso crear perímetro.

2.8 Conclusiones Parciales

En este capítulo se ha tratado sobre el modelo de dominio para representar de forma visual el entorno real del proyecto. Se especificaron los requisitos de software llegando así a un entendimiento de lo que hay que hacer y se definieron los casos de uso del sistema, diagrama de caso de uso del sistema y la descripción de los mismos para llegar a una comprensión del sistema propuesto. También se definió el análisis y diseño del sistema ya que este flujo de trabajo permite profundizar en los casos de usos detallándolos de manera que se refleje una vista interna del sistema descrita con el lenguaje de los desarrolladores y se mostró la arquitectura escogida para la realización de la aplicación.

Capítulo 3: Algoritmos.

3.1 Introducción

En este capítulo se explica en detalles los diferentes algoritmos que se emplean en la realización del video sensor para el control de perimetrado virtual.

3.2 Algoritmo de posición relativa polígono –punto

El polígono como entidad geométrica plana puede estar en cualquier lugar del espacio. Por tanto la clasificación de un punto con respecto a este sería igual que con respecto a un plano. Es por esto que la clasificación que se explica es de la posición del punto respecto a un polígono que está contenido en el plano. (Iznaga Benítez, et al., 2006)

La posición relativa de un punto con respecto a un polígono convexo se muestra en la figura 11. El polígono convexo puede ser regular o irregular.

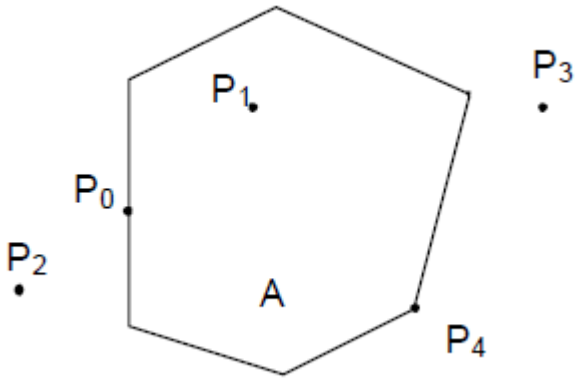

Figura 11: Clasificación del punto con relación a un polígono convexo.

La posición que representa el punto P1 es Dentro del polígono A, P2 y P3 están fuera, y P0, P4 se encuentran en el contorno. Precisando aún más la clasificación el punto P4 se encuentra en un vértice del polígono A y el punto P0 está en la arista.

Primera: Un punto está fuera de un polígono convexo positivo, si se encuentra a la derecha de al menos uno de los segmentos que lo forman. Observe la figura 12.

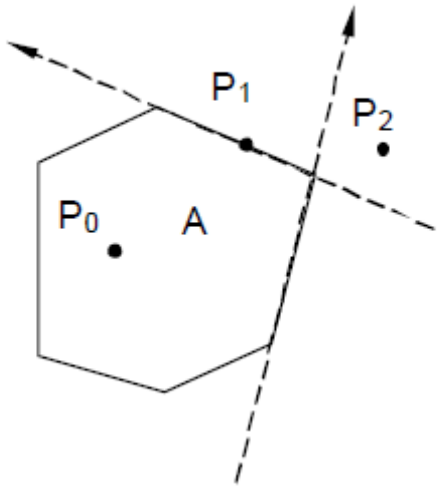

Figura 12: Modelo geométrico para la primera solución.

Estableciendo un recorrido por todos los segmentos del polígono A se puede determinar la posición del punto respecto a la recta que lo contiene. Si la distancia del punto a la recta es positiva entonces el punto estará fuera.

Segundo: Construyendo vectores desde el punto de análisis P0 hasta los vértices del polígono A, se puede decir que el punto está dentro si la suma de los ángulos entre vectores consecutivos es de 360 grados. Observe la figura 13.

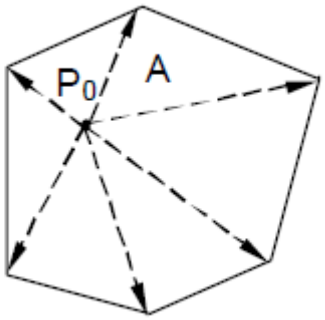

Figura 13: Modelo geométrico para la segunda solución.

Algunos autores plantean que el punto P0 se encuentra dentro del polígono A, si el área formada por los triángulos de los vectores consecutivos es igual al área del polígono.

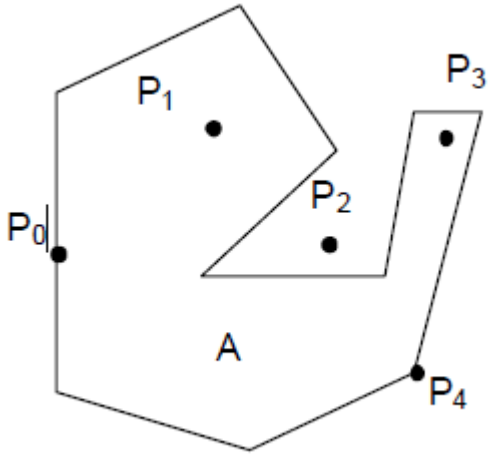

Figura 14: Posición relativa de un punto con respecto a un polígono.

La posición relativa de un punto Pi con respecto a un polígono A convexo o no, se muestra en la figura 14.

La clasificación sigue siendo la misma descrita con anterioridad, lo que cambia es el postulado. Se dice que un punto Pi está dentro del polígono A si la semirrecta que contiene al punto, corta al mismo un número impar de veces. Si el resultado de los corte es un número par entonces el punto está fuera.

El primer aspecto a considerar es cómo trazar una semirrecta que contenga a los puntos. La semirrecta se trazará a partir del punto en dirección al eje x. se hace evidente que no se puede representar en un espacio finito, por tal motivo se toma el mayor valor de la coordenada x del conjunto de puntos que define el polígono adicionándole un incremento. Luego, se cambia la semirrecta por un segmento cuyo extremo derecho queda definido por (Ax max + Incremento, Piy). Observe la figura 15. (Iznaga Benítez, et al., 2006)

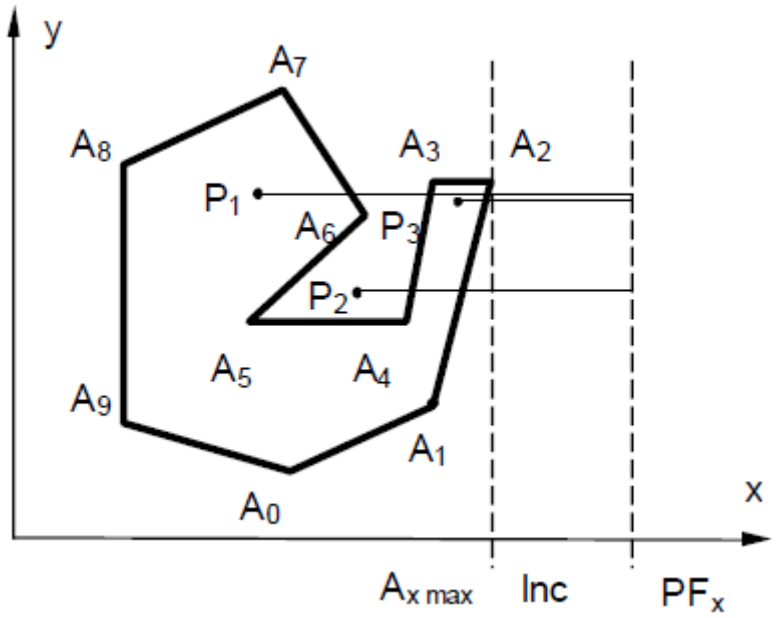

Figura 15: Determinación del punto extremo del segmento.

Los segmentos generados son analizados con los segmentos del polígono, determinando y contabilizando las intersecciones. No siempre los segmentos obtenidos permiten asegurar el postulado que dio origen a esta solución a los mismo se le denominan degenerados. Un ejemplo de un segmento degenerado se puede apreciar en la figura 16.

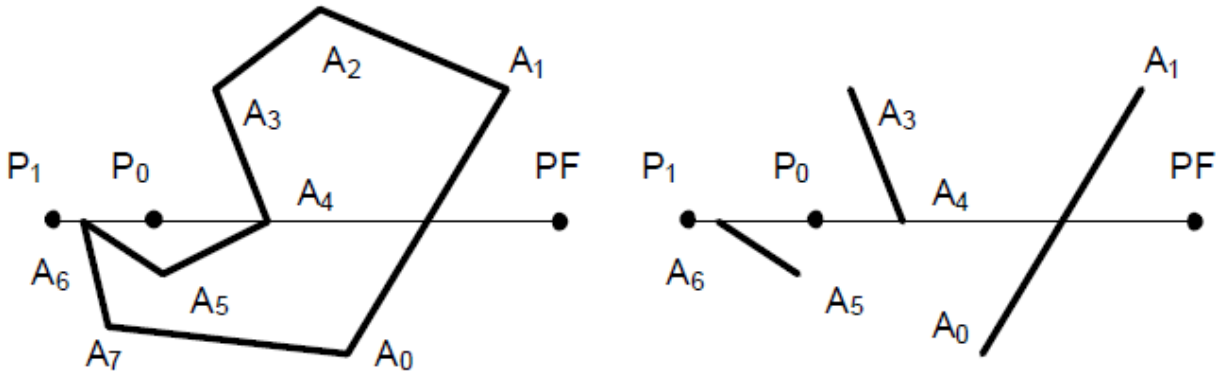

Figura 16: Segmento degenerado.

Los puntos P0 y P1 pasarían por el mismo segmento de recta. Contando una sola vez las intersecciones que pasan por los vértices se tiene como resultado que el P1 posee tres intersecciones, clasificándose como dentro del polígono lo cual no es real.

Se define como segmento degenerado aquel que contiene un vértice del polígono. Para evitar la formación de segmentos degenerados, se puede variar la coordenada y de PF en el valor de un incremento, de continuar siendo degenerado se sigue incrementado hasta

que no lo sea. Sólo así, puede ser tomado el segmento de análisis como válido para determinar la cantidad de intersecciones. (Iznaga Benítez, et al., 2006)

3.2 Algoritmo de estimación y sustracción de fondo

Para controlar cuando un objeto entra al perímetro se debe dar un seguimiento al objeto en la secuencia de video. Por lo que uno de los primeros pasos a seguir consiste en separar el fondo, o background, del primer plano, o foreground. Conceptualmente, se distingue fondo de primer plano definiendo el fondo como la parte estacionaria de la escena, mientras el primer plano estaría formado por los objetos en movimiento o temporalmente estáticos.

Para separar objetos y fondo, se lleva a cabo una sustracción de fondo, calculando la diferencia entre los píxels del fondo, que deben ignorarse, y los píxels de la imagen capturada. Mantener una representación precisa del fondo es importante para detectar con claridad todos los objetos no estacionarios de la escena, y a esto es a lo que se llama estimación de fondo. Para llevar a cavo este proceso se utilizará el algoritmo mezcla de Gaussianas (MoG).

Una vez que los objetos se identifican como foreground, constituyen la información que sirve de entrada a que se realice el seguimiento sobre los objetos.

3.2.1 Mezcla de Gaussianas (MoG)

El modelo MoG se basa en usar K Gaussianas por píxel, que se modelan a través de una media (μ), una desviación típica (σ) y un factor de peso (ω). La media indica el valor más probable de dicho píxel para cada píxel, la desviación típica lo que se puede alejar dicho valor medio por cada uno de los lados, y el peso indica que Gaussiana es la de mayor importancia. Destacar que la suma de los pesos de una Gaussiana para cada píxel es igual a 1. A continuación se explicará paso a paso las operaciones realizadas para modelar el fondo con MoG y extraer el frente.

- **Inicialización de Parámetros.**

  En primer lugar, se inicializan los parámetros que representan el fondo: media, desviación y peso. Para ello, es necesario tomar una serie de decisiones iniciales; se ha inicializado la media de una de las Gaussianas (por ejemplo, la Gaussiana i =1) a la primera imagen y el resto de medias a valores aleatorios. Con ello se considera que la primera Gaussiana modelará el píxel en la primera imagen. Por este motivo, el peso w o porcentaje de distribución para k = 1 debe ser un valor alto (próximo a 1) y para el resto será un valor muy pequeño (próximo a 0), ya que la suma de los pesos de las

Gaussianas debe ser 1. En cuanto a la desviación inicial dependerá del tipo de secuencia, generalmente, se da un valor elevado a las tres componentes.

$$B_1(x,y) = \{\mu_i(x,y),\ \sigma_i(x,y),\ i = 1 \ldots K\} \quad /$$

$$\begin{cases} i = 1 \Rightarrow \mu_{i,1}(x,y) = I_1(x,y) \quad \sigma_{i,1}(x,y) = P \quad w_{i,1} = 1 \\ 1 < i \leq K \Rightarrow \mu_{i,1}(x,y) = aleatorio \quad \sigma_{i,1}(x,y) = P \quad w_{i,1} = 0 \end{cases}, \quad \sum_{i=1}^{K} w_{i,t} = 1$$

- **Estimación del background y actualización de parámetros.**

Para obtener los píxeles pertenecientes al fondo, se calcula la imagen diferencia entre los K modelos posibles; si se encuentra parecido con alguna distribución, es decir, si la diferencia difiere en c (2-3) veces el valor estimado de la desviación correspondiente a la distribución, entonces el píxel se marca como fondo y se actualizan los parámetros de dicha distribución a través de una media móvil:

$$\exists i \in [1, K] / \left| I_t(x,y) - \mu_{i,t}(x,y) \right| \leq c\,\sigma_{i,t}(x,y)$$

$$\Downarrow$$

$$\begin{cases} \mu_{i,t+1}(x,y) = \rho I_t(x,y) + (1-\rho)\mu_{i,t}(x,y) \\ \sigma^2_{i,t+1}(x,y) = \rho\left(I_t(x,y) - \mu_{i,t}(x,y)\right)^2 + (1-\rho)\sigma^2_{i,t}(x,y) \\ w_{i,t+1}(x,y) = w_{i,t}(x,y) \end{cases}$$

Para que la suma de pesos siga valiendo uno, se normalizan los pesos:

$$w_{i,t+1}(x,y) = \frac{w_{i,t+1}(x,y)}{\sum_{i=1}^{j=K} w_{j,t+1}(x,y)}$$

Si no se ha encontrado parecido con ninguna de las Gaussianas que modelaban el píxel, se reemplaza la de menor peso por una nueva Gaussiana de peso muy pequeño y se media el valor del píxel; con ello, se consigue introducir un modo en la distribución del píxel. Una vez actualizados los parámetros, para obtener una imagen de fondo se calcula para cada píxel la media ponderada de las K distribuciones:

$$B_t(x,y) = \sum_{i=1}^{K} w_{i,t}(x,y) \cdot \mu_{i,t}(x,y)$$

- **Obtención del foreground.**

  En el proceso de detección de píxeles de frente y fondo, en primer lugar, se ordenan los pesos de mayor a menor y se escogen como modelo de fondo de cada píxel las B Gaussianas de mayor peso, es decir, aquellas cuyo peso acumulado supere un umbral (por ejemplo, un 0.85). Si para la Gaussiana b que supere dicho umbral la diferencia entre la media y el píxel de la imagen actual es inferior a c veces su desviación, se considerará píxel de fondo. En caso contrario, será objeto en movimiento Ft**.**

$$B = \arg\min_b \sum_{i=1}^{b} w_{i,t}(x,y) > \tau$$
$$\exists b \in [1,B] / \left| I_t(x,y) - \mu_{b,t}(x,y) \right| \leq c\sigma_{b,t}(x,y) \Rightarrow F_t(x,y) = 0$$
$$\forall b \in [1,B], \left| I_t(x,y) - \mu_{b,t}(x,y) \right| > c\sigma_{b,t}(x,y) \Rightarrow F_t(x,y) = 1$$

La mezcla de Gaussianas es un algoritmo complejo algorítmicamente pero produce buenos resultados y con la librería Opencv se facilita más su utilización debido a que solo se llaman funciones que internamente realizan todo el algoritmo explicado anteriormente. Entre sus ventajas destaca la robustez, ya que permite modelar fondos multimodales, es decir, manejar múltiples modos de distribución (o tipos de movimiento). (Bayona Gómez, et al., 2009)

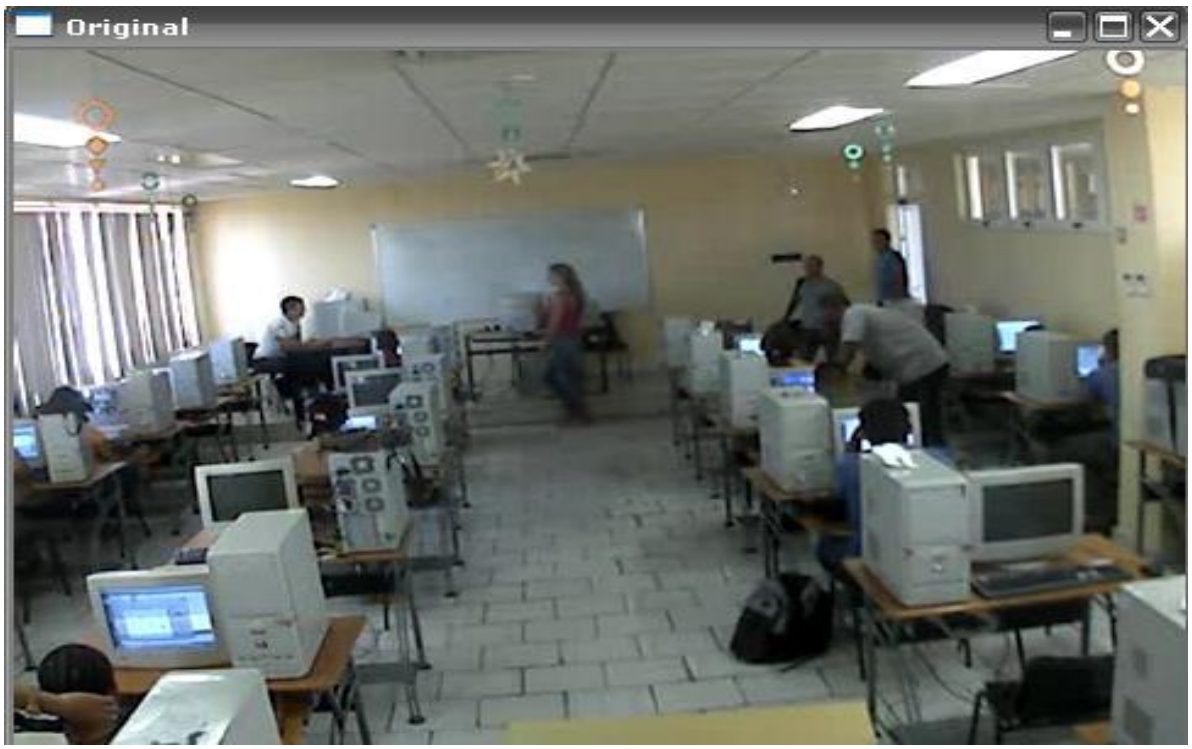

Figura 17: Video original.

Figura 18: Video después de aplicarle la sustracción de fondo.

3.3 Algoritmo de seguimiento de objetos

En una secuencia de video se pueden distinguir objetos activos y pasivos. Los objetos activos tienen una evolución a lo largo del tiempo: se mueven rápidamente o lentamente, se quedan estáticos, se solapan o se alejan unos de otros. Los pasivos son invariables no cambian su posición ni características y constituyen lo que se ha denominado fondo. El fondo no ofrece demasiada información, siendo los objetos activos los que resultan de interés. Por este motivo se han desarrollado técnicas para extraer los objetos activos y caracterizarlos por su tamaño, color, forma, pero aún puede darse un paso más que consiste en caracterizar su comportamiento y acciones a lo largo del tiempo, y a esto se le denomina seguimiento de objetos. En el caso especial del video sensor para el control de perimetrado virtual que se está desarrollando, es necesario realizar un seguimiento a los objetos que se encuentran en movimiento en la escena, para saber donde se encuentran ubicados en cada instante de tiempo y de esta manera determinar cuando entran al perímetro a través del algoritmo de posición relativa que se explica anteriormente.

- Para realizar el seguimiento es necesario determinar cuáles son los objetos de interés en la escena debido a que hay objetos en movimiento que por sus dimensiones u otras características no resultan de interés. Cada objeto tiene un identificador que los hace únicos, además de muchas otras características como el centroide, área y si se le está realizando seguimiento o no. Los mismos son analizados frame a frame comparando la distancia entre ellos, si está en un rango mínimo determinado es porque el objeto del frame anterior es el mismo que el frame que se está analizando, también se analiza la posibilidad que puedan entrar en la escena nuevos objetos, estos son comparados con los que se le está realizando el seguimiento y si no coinciden en el identificador, características o posición se agregan a la lista de objetos a seguir y se les da un identificador nuevo. Si algún objeto deja de ser seguido por un tiempo (número de frames) predeterminado, se elimina de la lista de seguimiento.

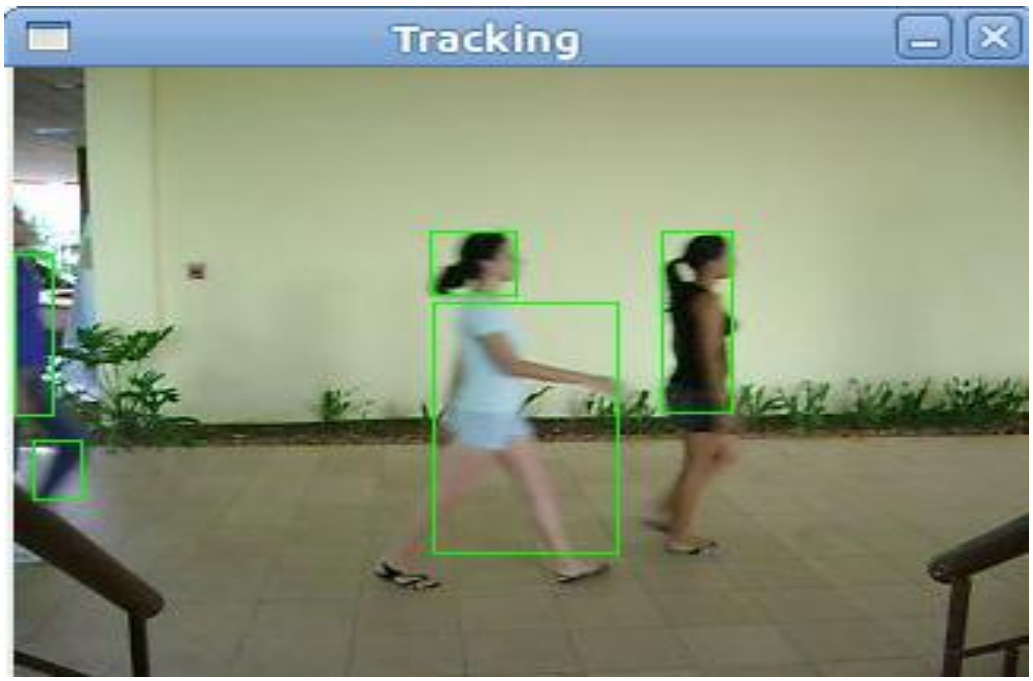

Figura 19: Seguimiento de objetos.

3.4 Conclusiones Parciales

En este capítulo se explicaron en detalle los algoritmos empleados para la realización del video sensor de control de perimetrado virtual. Se determinó el algoritmo de posición relativa polígono –punto para saber en cualquier instante de tiempo si el objeto está dentro del perímetro. También se estableció la utilización del algoritmo para la sustracción de fondo basado en mezcla de Gaussianas así como el de detección de objetos.

Capítulo 4: Implementación y Prueba.

4.1 Introducción

En este capítulo los elementos del modelo del diseño se implementan en términos de componentes como son los ficheros de código fuente, ficheros de código binario y ejecutable. Además se realizan pruebas al sistema para lograr erradicar errores que puedan ser introducidos en la implementación del video sensor y para comprobar que el producto final cumple con los requisitos establecidos en la primera fase de la investigación.

4.2 Descripción de los componentes

Un componente es la parte modular de un sistema, desplegable y reemplazable que encapsula implementación, un conjunto de interfaces y proporciona la realización de los mismos. Un componente típicamente contiene clases y puede ser implementado por uno o más artefactos. Son las piezas reutilizables de alto nivel a partir de las cuales se pueden construir los sistemas. El sistema cuenta con 3 componentes: cvBlob.dll, OpenCV.dll y EdgeControl que se encargan de distribuir las funcionalidades necesarias para la realización del video sensor.

cvBlob: Esta librería se utiliza para trabajar con los Blobs, la misma ofrece diversas funcionalidades que facilitan el trabajo con los objetos que se encuentran en la escena.

OpenCV: Esta librería se utiliza para el tratamiento de imágenes.

EdgeControl: Tiene como objetivo brindar todas las funcionalidades del componente.

4.2.1 Diagrama de Componentes

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software, modela la vista estática de un sistema. Se representa como un grafo de componentes software unidos por medio de relaciones de dependencia (compilación, ejecución), pudiendo mostrarse las interfaces que estos soporten. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. Cada diagrama describe un apartado del sistema. Los diagramas de componentes tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes.

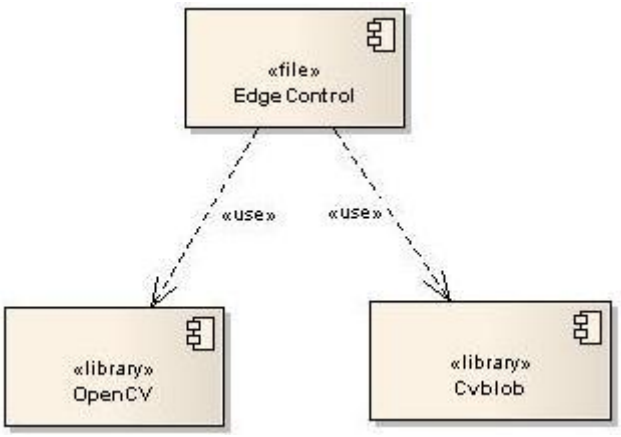

Figura 20: Diagrama de componentes.

4.3 Pruebas de software

La calidad de un sistema está determinada, entre otras cosas, por la coincidencia entre lo que se programó y los requisitos establecidos en la primera fase. Para comprobar el grado de cumplimiento de estos requisitos se usan las pruebas del sistema. Estas definen un conjunto amplio de acciones de comprobación que abarcan todas las características que determinan la calidad de un software. Se comprueban las funcionalidades diseñando casos de prueba que definen cómo proceder. Estos casos de prueba incluyen los juegos de datos a usar que son los válidos o esperados y los no válidos o no esperados por el programa. Además establecen los resultados a alcanzar en correspondencia de la lógica del programa y los datos ingresados. Describen las condiciones generales en las que se debe aplicar las pruebas para obtener los objetivos propuestos.

El objetivo de los casos de prueba es forzar al máximo el sistema en los puntos críticos para encontrar fallos y detectar defectos. Las pruebas se deben aplicar durante todo el ciclo de vida del software e invariablemente se le debe dedicar una gran parte del esfuerzo total del desarrollo. Se deben planificar correctamente desde el inicio y establecer qué hacer, cómo hacer, quién va a hacer y en qué condiciones hacer las comprobaciones. Es beneficioso que los desarrolladores prueben su producto pero que no falte la mano de terceras personas que no intervinieron en el proyecto directamente ya que así se detecta mayor cantidad de fallas. (S. Pressman)

4.3.1 Pruebas unitarias

Las pruebas unitarias permiten probar, como su nombre lo indica, cada unidad independiente del software. Actúan esencialmente sobre el código fuente y sobre los elementos básicos de la interfaz de cada módulo. Pressman (S. Pressman) plantea que los casos de prueba que se generan durante las pruebas de unidad deben estar encaminados a verificar los siguientes elementos:

- ✓ **Interfaz**: "Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada".
- ✓ **Estructuras de datos locales**: "Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente, conservan su integridad durante todos los pasos de ejecución del algoritmo".
- ✓ **Condiciones límites**: "Se prueban las condiciones límites para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento".
- ✓ **Caminos independientes**: "Se ejercitan todos los caminos independientes (caminos básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez".
- ✓ **Caminos de manejo de errores**: "Se prueban todos los caminos de manejo de errores".

Con las pruebas unitarias es posible aislar una parte del código de manera que pueda ser analizado. Un ejemplo de esto es evaluar las funciones o métodos, a los cuales se les realiza una entrada de datos para obtener los datos de salida correctos. Este tipo de pruebas validan la forma en la que las funciones y métodos trabajan en cada caso particular.

Método de caja negra

Las Pruebas de Caja Negra deben su nombre a los elementos que estas revisan y las condiciones en que se hace la revisión. Estas se basan en los requerimientos funcionales del sistema y se llevan a cabo desde el exterior de la aplicación. Este tipo de prueba es importante a la hora de medir el grado de cumplimiento de los requerimientos solicitados por el cliente y se aplican sobre la interfaz de la aplicación observando las respuestas del sistema antes determinadas acciones y los datos de salida para determinados datos de entrada. (S. Pressman)

Método de caja blanca

Las pruebas de Caja Blanca se nombran de esta forma porque a diferencia de las pruebas de Caja Negra que actúan sobre la interfaz, estas revisan la parte interna del software, específicamente sobre el código fuente. Se basan en el examen minucioso de los detalles procedimentales. Se comprueban los caminos lógicos del sistema generando casos de prueba que ejerciten las estructuras condicionales y los bucles. Existen varios métodos que analizan diferentes partes del programa y se complementan entre sí para garantizar la calidad del sistema.

La técnica del camino básico se utiliza para comprobar la complejidad lógica de un diseño procedimental, permite diseñar casos de prueba para cubrir todas las sentencias de un programa a partir de la obtención de un conjunto de caminos independientes. La complejidad ciclomática, como resultado fundamental de estas pruebas, acota la cantidad mínima de casos de prueba que se deben ejecutar. La prueba de las Condiciones es un método que se encamina hacia la ejercitación de las condiciones. Se basa en el principio de que si un conjunto de casos de prueba es capaz de ejercitar todas las condiciones contenidas en un bloque de código, este mismo conjunto serviría para encontrar más errores en el programa que no tengan que ver directamente con las condiciones.

La prueba del Flujo de Datos verifica la validez en el uso de las variables para manipular los datos de la aplicación. Selecciona los casos de prueba atendiendo a las definiciones y los usos de las variables. El procedimiento indica que se debe encontrar las sentencias donde se define cada variable y las sentencias donde se hace uso de las mismas. Luego se encuentran las cadenas de definición, uso que representan el ciclo de vida de las variables y se diseñan casos de prueba que las ejecuten en su totalidad.

La prueba de los bucles se centra en la validez de las estructuras cíclicas o bucles. El objetivo es probar el comportamiento de estas estructuras en sus valores límites de iteración. Los bucles se clasifican en cuatro tipos: Bucles simples, Bucles anidados, Bucles concatenados y Bucles no estructurados. Aunque la esencia es la misma, cada tipo se prueba de forma diferente. De lo anterior pudiera pensarse que las pruebas de Caja Blanca logran enmendar todos los errores del programa. La desventaja de estas es entonces de tipo logística ya que resulta imposible abarcar todo el código fuente de un sistema medianamente grande. El tiempo necesario para realizarlas sería considerable y se torna compleja su aplicación sobre algoritmos críticos.

Al analizar los métodos de prueba de Caja Blanca y Caja Negra se llega a la conclusión de que es más factible en este contexto aplicar las pruebas de Caja debido a que el video sensor implementado no cuenta con interfaz.

Dentro de la prueba de caja blanca, la técnica que se utilizará es la del camino básico. Esta prueba permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución, además también garantiza que durante la prueba en los casos de prueba obtenidos a través del camino básico se ejecute cada sentencia del programa por lo menos una vez.

Para aplicar la técnica del camino básico se debe introducir la notación para la representación del flujo de control, este puede representarse por un Grafo de Flujo en el cual:

- Cada nodo del grafo corresponde a una o más sentencias de código fuente.
- Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo.
- Se calcula la complejidad ciclomática del grafo.

Un grafo de flujo está formado por 3 componentes fundamentales (nodos, aristas, regiones) que ayudan a su elaboración y comprensión, estos brindan información para confirmar que el trabajo se está haciendo adecuadamente. Para realizar la técnica de prueba del camino básico es necesario calcular antes la complejidad ciclomática del algoritmo o fragmento de código a analizar. A continuación se enumeran las sentencias de código del procedimiento realizado sobre el método insidePerimeter (CvPoint point) el cual se encarga de decir cuando un blob se encuentra dentro o fuera del perímetro.

```
bool Polygon::insidePerimeter(CvPoint point)
{
    1 if(!this->is_closed)
        2 return false;
    3 int counter = 0;
    3 int i;
    3 double xinters;
    3 CvPoint p1,p2;
    3 p1 = this->points->at(0);
    4 for (i = 1;i <= this->points->length();i++)
    {
        5 p2 = this->points->at(i % this->points->length());
        6 if (point.y > MIN(p1.y, p2.y))
        {
            7 if (point.y <= MAX(p1.y,p2.y))
            {
                8 if (point.x <= MAX(p1.x,p2.x))
                {
                    9 if (p1.y != p2.y)
                    {
                        10 xinters = (point.y-p1.y)*(p2.x-p1.x)/(p2.y-p1.y)+p1.x;
                        11 if (p1.x == p2.x || point.x <= xinters)
                            12 counter++;
                    }
                }
            }
        }
        13 p1 = p2;
    }
    14 if (counter % 2 == 0)
        15 return OUTSIDE;
    else
        16 return INSIDE;
} 17
```

Figura 21: Representación del algoritmo método insidePerimeter (CvPoint point).

A continuación se representa el Grafo de flujo asociado al algoritmo anteriormente descrito:

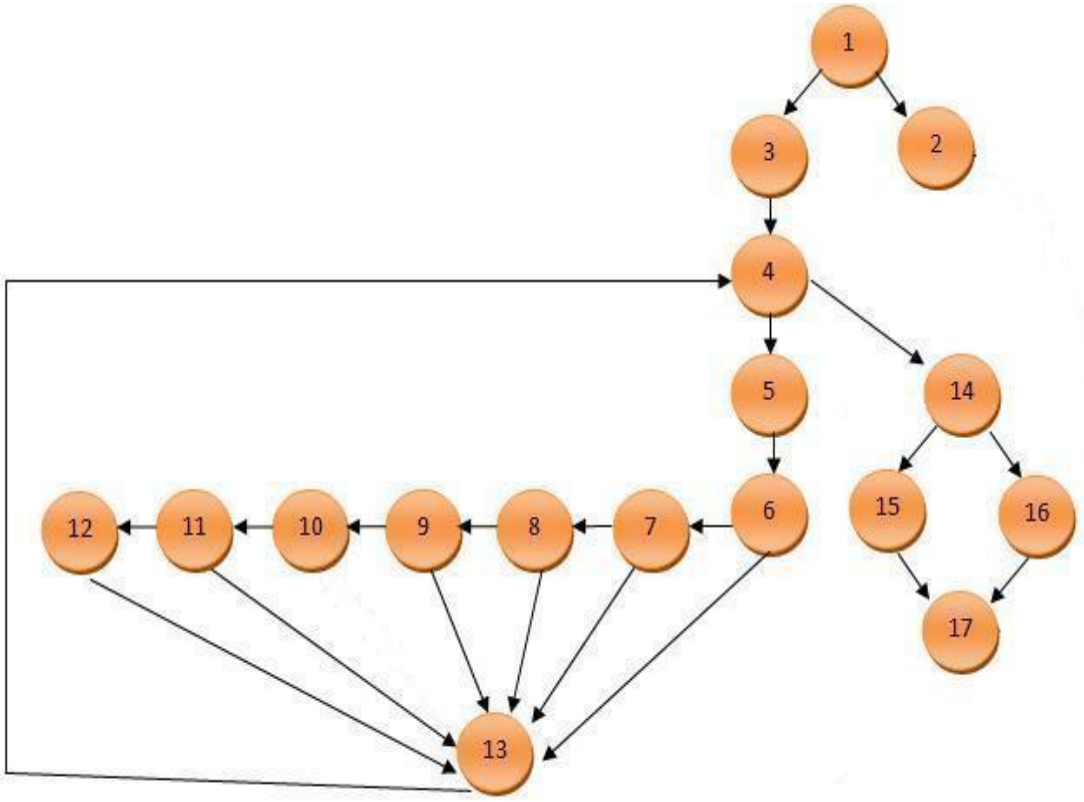

Figura 22: Grafo de flujo asociado al algoritmo método insidePerimeter (CvPoint point).

Cálculo de la complejidad ciclomática a partir de un segmento de código

Para efectuar el cálculo de la complejidad ciclomática del código es necesario tener varios parámetros como son la cantidad total de aristas del grafo, cantidad total de nodos para la siguiente fórmula:

V (G) = (A – N) + 2

V (G) = (24 – 17) + 2

V (G) = 9

Cantidad de Caminos independientes:

Número	Camino básico
	1-2
	1-3-4-14-15-17
	1-3-4-14-16-17
	1-3-4-5-6-13-14-15-17
	1-3-4-5-6-7-13-14-15-17
	1-3-4-5-6-7-8-13-4-14-15-17
	1-3-4-5-6-7-8-9-13-4-14-15-17

	1-3-4-5-6-7-8-9-10-11-13-4-14-15-17
	1-3-4-5-6-7-8-9-10-11-12-13-4-14-15-17

Tabla 8: Caminos independientes.

Después de haber extraído los caminos básicos del flujo, se procede a ejecutar los casos de pruebas para este procedimiento, se debe realizar al menos un caso de prueba por cada camino básico. Para realizarlos es necesario cumplir con las siguientes exigencias:

- Descripción: Se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o no se entre algún dato erróneo.
- Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada para ver el funcionamiento del procedimiento.
- Entrada: Se muestran los parámetros que entran al procedimiento.
- Resultados Esperados: Se expone resultado que se espera que devuelva el procedimiento.

Caso de prueba para el camino básico 1

Camino 1: 1-2

Caso de prueba: Probando la función insidePerimeter (CvPoint point)

Parámetros de entrada: El valor de la variable is_close es true.

Resultados: Se comprueba que se cumpla la condición y que el método retorne falso.

Caso de prueba para el camino básico 2

Camino 2: 1-3-4-14-15-17

Caso de prueba: Probando la función insidePerimeter (CvPoint point).

Parámetros de entrada: El valor de la variable is_close es false, la variable de chequeo del for i=5, la longitud de la lista de punto points->length()= 4 y la variable counter es un número par.

Resultados: Se comprueba que la variable is_close = false, que la variable i > points->length(), la variable counter sea un número par para que se cumpla la condición y que el método retorne OUTSIDE.

Caso de prueba para el camino básico 3

Camino 3: 1-3-4-14-16-17

Caso de prueba: Probando la función insidePerimeter (CvPoint point).

Parámetros de entrada: El valor de la variable is_close es false, la variable de chequeo del for i=5, la longitud de la lista de punto points->length()= 4 y la variable counter es un número impar.

Resultados: Se comprueba que la variable is_close = false, que la variable i > points->length(), la variable counter sea un número impar para que se cumpla la condición y que el método retorne INSIDE.

Caso de prueba para el camino básico 4

Camino 4: 1-3-4-5-6-13-4-14-15-17

Caso de prueba: Probando la función insidePerimeter (CvPoint point).

Parámetros de entrada: El valor de la variable is_close es false, la variable de chequeo del for i=4, la longitud de la lista de punto points->length()= 4, la variable point.y = 20, la variable p1.y = 30, la variable p2.y =10, la variable point.x = 10, la variable p1.x = 10, la variable p2.x= 15, la variable p1.y = 30 y la variable p2.y =10.

Resultados: Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.

Caso de prueba para el camino básico 5

Camino 5: 1-3-4-5-6-7-13-4-14-15-17

Caso de prueba: Probando la función insidePerimeter (CvPoint point)

Parámetros de entrada: El valor de la variable is_close es false, la variable de chequeo del for i=4, la longitud de la lista de punto points->length()= 4, la variable point.y = 20, la variable p1.y = 30, la variable p2.y =10, la variable point.x = 10, la variable p1.x = 10, la variable p2.x= 15, la variable p1.y = 30 y la variable p2.y =10.

Resultados: Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.

Caso de prueba para el camino básico 6

Camino 6: 1-3-4-5-6-7-8-13-4-14-15-17

Caso de prueba: Probando la función insidePerimeter (CvPoint point)

Parámetros de entrada: El valor de la variable is_close es false, la variable de chequeo del for i=4, la longitud de la lista de punto points->length()= 4, la variable point.y = 20, la variable p1.y = 30, la variable p2.y =10, la variable point.x = 10, la variable p1.x = 10, la variable p2.x= 15, la variable p1.y = 30 y la variable p2.y =10.

Resultados: Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.

Caso de prueba para el camino básico 7

Camino 7: 1-3-4-5-6-7-8-9-13-4-14-15-17

Caso de prueba: Probando la función insidePerimeter (CvPoint point)

Parámetros de entrada: El valor de la variable is_close es false, la variable de chequeo del for i=4, la longitud de la lista de punto points->length()= 4, la variable point.y = 20, la variable p1.y = 30, la variable p2.y =10, la variable point.x = 10, la variable p1.x = 10, la variable p2.x= 15, la variable p1.y = 30 y la variable p2.y =10.

Resultados: Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.

Caso de prueba para el camino básico 8

Camino 8: 1-3-4-5-6-7-8-9-10-11-13-4-14-15-17

Caso de prueba: Probando la función insidePerimeter (CvPoint point)

Parámetros de entrada: El valor de la variable is_close es false, la variable de chequeo del for i=4, la longitud de la lista de punto points->length()= 4, la variable point.y = 20, la variable p1.y = 30, la variable p2.y =10, la variable point.x = 10, la variable p1.x = 10, la variable p2.x= 15, la variable p1.y = 30 y la variable p2.y =10.

Resultados: Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.

Caso de prueba para el camino básico 9

Camino 9: 1-3-4-5-6-7-8-9-10-11-12-13-4-14-15-17

Caso de prueba: Probando la función insidePerimeter (CvPoint point)

Parámetros de entrada: El valor de la variable is_close es false, la variable de chequeo del for i=4, la longitud de la lista de punto points->length()= 4, la variable point.y = 20, la variable p1.y = 30, la variable p2.y =10, la variable point.x = 10, la variable p1.x = 10, la variable p2.x= 10, la variable p1.y = 30 y la variable p2.y =10.

Resultados: Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.

Resultados de la prueba:

Luego de haberse aplicado la prueba de caja banca, en especifico la prueba del camino básico, seleccionando diferentes caminos a través del Cálculo de la complejidad ciclomática a partir de un segmento de código, se ha arribado a la conclusión de que los resultados obtenidos fueron satisfactorios ya que se pudo comprobar que el flujo de trabajo de la función está correcto debido a que cumple con las condiciones necesarias que se habían planteado.

Camino	Caso de Prueba	Función del Código	Resultado
1	Probando la función insidePerimeter (CvPoint point)	Se comprueba que se cumpla la condición y que el método retorne falso.	Satisfactorio
2	Probando la función insidePerimeter (CvPoint point)	Se comprueba que la variable is_close = false, que la variable i > points->length(), la variable counter sea un número par y que el método retorne OUTSIDE.	Satisfactorio
3	Probando la función insidePerimeter (CvPoint point)	Se comprueba que la variable is_close = false, que la variable i > points->length (), la variable counter sea un número impar y que el método retorne INSIDE.	Satisfactorio
4	Probando la función insidePerimeter (CvPoint point)	Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.	Satisfactorio

5	Probando la función insidePerimeter (CvPoint point)	Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.	Satisfactorio
6	Probando la función insidePerimeter (CvPoint point)	Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.	Satisfactorio
7	Probando la función insidePerimeter (CvPoint point)	Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.	Satisfactorio
8	Probando la función insidePerimeter (CvPoint point)	Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.	Satisfactorio
9	Probando la función insidePerimeter (CvPoint point)	Se comprueba que la variable is_close = false, que la variable i es igual a la longitud de la lista, la variable counter sea un número par y que el método retorne OUTSIDE.	Satisfactorio

Tabla 9: Resultados de la prueba de caja blanca.

4.4 Tasa de Recall y Presicion

Precision y *Recall* son dos indicadores ampliamente utilizados para evaluar la exactitud y eficiencia de los sistemas de recuperación de información. Pueden ser vistos como versiones extendidas de *Precision*, un indicador simple que calcula la fracción de los casos para los que se devuelve el resultado correcto. Cuando se utiliza la *Precision* y el *Recall*, el conjunto de etiquetas posibles para un caso determinado se divide en dos subgrupos, uno de los cuales se considera "relevante" a los efectos de la métrica. *Recall* entonces se calcula como la fracción de casos correctos entre todas las instancias que en realidad

pertenecen al subconjunto relevante, mientras que la *Precision* es la fracción de casos correctos entre los que el algoritmo considera que pertenecen al subgrupo relevante.

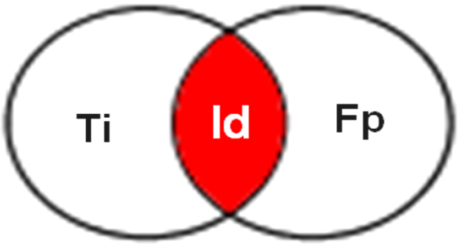

Figura 23: Recall y Presicion rate.

Precision puede ser visto como una medida de la exactitud o fidelidad, mientras que *Recall* es una medida de integridad. En términos aún más simple, un alto *Recall* significa que no se ha perdido nada, pero se puede tener una gran cantidad de resultados inútiles para filtrar (lo que implica poca *Precision*). Alto *Precision* significa que todo lo retornado es un resultado relevante, pero podría no haber encontrado todos los elementos relevantes (lo que implicaría bajo *Recall*). A menudo, existe una relación inversa entre la *Precision* y el *Recall*, donde es posible aumentar una a costa de la reducción de la otra, ya que a mayor *Recall*, menor *Precision*.

En el marco de la investigación realizada y el algoritmo desarrollado para el control de perimetrado virtual el *Recall* y el *Presicion* pueden ser definidos como:

$$\boldsymbol{Recall} = \frac{\mathrm{Id}}{\mathrm{Id} + \mathrm{Fp}}$$

$$\boldsymbol{Precision} = \frac{\mathrm{Id}}{\mathrm{Ti}}$$

Donde:

Id: Intrusiones detectadas correctamente.

Fp: Falsos positivos.

Ti: Total de intrusiones.

Intrusión: Acción de introducirse sin derecho en una jurisdicción, cargo o propiedad. (WordReference.com, 2011)

Resultados de la evaluación de los indicadores tasa de Recall y Presicion.

En los videos en los cuales se realizó el experimento de la tasa de *Recall* y *Precision* el sistema detectó correctamente 21 personas de 21 que se introducen en la región marcada de la escena, con valor de 4 falsos positivos, esto sucede por los cambios de iluminación a la hora de segmentar los objetos en la escena, lo que arrojó un valor de la tasa *Precision* del 100% y de *Recall* de 87.5%. Los resultados experimentales demuestran la eficacia del video sensor para el control del perimetrado virtual en un conjunto de imágenes de video. En la siguiente tabla se muestran los valores Ti, Id y Fp por cada video, así como el total.

Videos	Ti	Id	Fp	Recall	Presicion
Video1	4	4	1	87.5 %	100%
Video2	1	1	1		
Video3	3	3	1		
Video4	13	13	1		
Total	21	21	4		

Tabla 10: Tabla de resultados de *Recall* y *Presicion*.

Figura 24: Falso positivo.

4.5 Conclusiones Parciales

En este capítulo se realizó el diagrama de componente logrando así establecer una dependencia lógica entre componentes de software EdgeControl, cvblob y OpenCV. También se validó el diseño a través de las pruebas de caja blanca, en específico la prueba

del camino básico en la que se seleccionaron 9 caminos a través del cálculo de la complejidad ciclomática, a partir de un segmento de código de una de las funcionalidades más importantes del video sensor. En general se obtuvieron resultados favorables porque se pudo probar que los procesos implementados realizan las funcionalidades requeridas con la calidad y eficiencia necesaria para que la aplicación cumpla con los requisitos especificados.

Conclusiones generales

Durante el desarrollo del presente trabajo se llevaron a cabo una serie de fases que permitieron dar cumplimiento al objetivo planteado y que abarcaron todas las tareas investigativas propuestas.

Por lo tanto se concluye que:

- El estudio realizado de temas referentes a la evolución de los sistemas de video sensores demostró que la principal deficiencia de los sistemas existentes es que están implementados bajo software propietario. Además posibilitó una mayor comprensión en la realización del video sensor para el control de perimetrado virtual.
- Se logró obtener un modelo diseño a partir de la descripción de los requisitos y se modeló la solución propuesta.
- Se definió una arquitectura centrada en el flujo de datos, implementando el patrón “Filtros y Tuberías”, acorde a las necesidades operacionales, posibilitando la implementación de un componente con alto grado de reusabilidad del código.
- Se implementó un componente que permite crear y monitorear un perímetro en un flujo de video obtenido de una cámara IP para el sistema de Video Vigilancia. El cual cumple todos los requisitos especificados y se puede integrar a otros componentes y módulos del proyecto.
- La validación del diseño a través de la prueba de caja blanca demuestra que el componente desarrollado posee la calidad requerida para cumplir con los requisitos especificados.
- La utilización del Presicion y Recall rate para evaluar el video sensor implementado, demuestra la eficiencia y calidad del mismo.

Recomendaciones

Durante el desarrollo del sistema han surgido ideas, que aportarían al video sensor mayor robustez, por lo que se recomienda:

- Que el monitoreo se pueda realizar en más de un perímetro en un flujo de video obtenido de una cámara IP.
- Lograr que el algoritmo utilizado para el seguimiento de los objetos tenga en cuenta el historial de posición de los objetos y modelos de apariencia (histogramas de color, características de formas).
- Eliminar la posibilidad de que el sistema permita pintar polígonos no convexo.
- Optimizar el algoritmo de posición relativa punto – polígono con la utilización del algoritmo Radial.

Glosario de términos

IDE: Entorno de Desarrollo Integrado.

Píxel: abreviatura de Picture Element, es la menor unidad posible con la que se compone cualquier imagen digital en una computadora.

Librería STL: La Standard Template Library es una colección de estructuras de datos genéricas y algoritmos escritos en C++.

Mono: es la plataforma de desarrollo de software libre basada en .NET, desarrollado para Unix, Linux y MacOS

.Net: plataforma de desarrollo de software creada por Microsoft.

Framework: estructura conceptual y tecnológica de soporte definida, normalmente con artefactos de software concretos, mediante la cual otro proyecto de software puede ser organizado y desarrollado

Herramientas CASE: Ingeniería de Software Asistida por Computadora.

GNU: es un acrónimo recursivo para "Gnu No es Unix". Comenzó en 1984 a desarrollar un sistema operativo completo, con la principal propiedad de ser Software Libre.

LGPL: GNU Library General Public License.

GPL: General Public License

Plugin: Pequeño programa que se adhiere a otro para poder ejecutar cierto tipo de archivos o para aportarle una función nueva, generalmente muy específica.

Polígono convexo: es un polígono en el que todos los ángulos interiores miden menos de 180 grados ó π radianes y todas sus diagonales son interiores

Gaussianas: función matemática en honor a Carl Friedrich Gauss.

Frame: fotograma o cuadro, es una imagen particular dentro de una sucesión de imágenes que componen una animación.

Referencias bibliográficas

autores, Colectivo de. 2010. Fase de Construcción. Flujo de Trabajo de Implementación. Modelo de implementación. 2010.

Axis Communications. Axis Communications. [Online] [Cited: octubre 22, 2010.] http://www.axis.com/es/corporate/contact.htm.

Bayona Gómez, Álvaro and San Miguel Avedillo, Juan Carlos. 2009. Detección de objetos abandonados/robados en secuencias de video-seguridad. Madrid : Universidad Autónoma de Madrid, 2009.

Calderón, Amaro, Sarah, Damaris and Valverde Rebaza, Jorge Carlos. 2007. Metodologías Ágiles. Trujillo : s.n., 2007.

CASEL. **Innovación Seguridad y Eléctrónica. 2007.** Argentina : Trigolo S.L.R, 2007, Vol. 33.

Colomer, Antonio Albiol. 2003. Seguimiento de objeto en secuencia de video. España : s.n., 2003.

Datys. 2010. Datys. [Online] 2010. [Cited: octubre 23, 2010.] http://www.datys.cu/wpinfnoticias.

Disca(departamento de informática de sistemas y computadoras). Disca. [Online] Universidad Politécnica de Valencia. [Cited: noviembre 15, 2010.] http://web-sisop.disca.upv.es/~imd/cursosAnteriors/2k3-2k4/copiaTreballs/serdelal/trabajoIMD.xml.

Electronic Dreams. Cámara IP. [Online] [Cited: octubre 20, 2010.] http://www.camara-ip.es/.

Gamma, Erich, Helm, Richard and Johnson, Ralph. 1995. Desing patterns: Elements of Reusable Object-Orientd. s.l. : Addison-Wesley, 1995.

Garzás, Javier. 2010. Diagramas de interacción. Habana : s.n., 2010.

Hernández León, Rolando Alfredo and Coello González, Sayda. 2002. EL paradigma cuantitavo de la investigacion científica. Habana : Universitaria, 2002. 959-16-0343-6.

Herrero Martín, Sonsoles. 2009. Análisis comparativo de técnicas de segmentación de secuencias de video basadas en el modelado del fondo. Madrid : Universidad autónoma de Madrid, 2009.

IBM Corporation. 2007. Rational Unified Process. s.l. : IBM, 2007.

Instituto Nacional de Estadistica e Informatica. 1999. Instituto Nacional de Estadistica e Informatica. [Online] 1999. [Cited: noviembre 10, 2010.] http://www.inei.gob.pe/biblioineipub/bancopub/Inf/Lib5103/Libro.pdf.

Intekio. Intekio. [Online] [Cited: Octubre 22, 2010.] http://intekio.com/index.php?option=com_content&task=view&id=33&Itemid=65).

Iznaga Benítez, Arsenio and Pérez Mallea, Ivan. 2006. Fundamentos de la gráfica por computadora. 2006.

Jacobson, Ivar, Booch, Grady and Rumbaugh, James. 2000. El proceso unificado de desarrollo de software. Madrid : Pearson educación, 2000. 84-78-29-036-2.

La revista informática. La revista informática. [Online] [Cited: Noviembre 3, 2010.] http://www.larevistainformatica.com/C1.htm.

Lenguajes de programación. Lenguajes de programación. [Online] [Cited: Noviembre 5, 2010.] http://www.lenguajes-de-programacion.com/programacion-java.shtml.

Lleonart Martín, Eva and García-Menacho, Asunción. Patrones. Valencia : Universidad Politécnica de Valencia.

Patrones. Valencia : Universidad Politécnica de Valencia.

Matias Simarro, Juan. 2004. Aplicación del análisis de dependencias a la arquitectura software. España : s.n., 2004.

Rumbaugh, James and Jacobson, Ivar. 2000. El lenguaje unificado de modelado. 2000.

S. Pressman, Roger. Ingeniería de software:Un enfoque práctico.

2010. Sistema de vigilancia IP. [Online] 2010. [Cited: diciembre 3, 2010.]

Universidad de las Ciencias Informática. 2010. Introducción a la disciplina de análisis y diseño. Habana : s.n., 2010.

Visconti, Marcello and Astudillo, Hernán. Fundamentos de la ingeniería de software. s.l. : Universidad Técnica Federico Santa María.

Du, Yingzi y Chang, Chein-I. 2003. Automated system for text detection in. s.l. : University of Maryland Baltimore County, 2003.

WordReference.com. 2011. WordReference.com Diccionario de la lengua española. [En línea] 2011. [Citado el: 6 de junio de 2011.] http://www.wordreference.com/definicion/intrusi%C3%B3n.