UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Detección electrónica de víctimas no superficiales

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A

Carlos Ignacio García Sánchez

DIRECTOR DE TESIS

M.I Larry H. Escobar Salguero

Ciudad Universitaria, Cd. Mx., julio 2016

Agradecimientos

En primera instancia agradezco a los programas PAPIIT IT102615 “Robots no convencionales para tareas de exploración y búsqueda” y PAPIME PE100616 “Servidor para prácticas de procesamiento digital de señales en tiempo real por proporcionar el equipo electrónico y de cómputo para desarrollar este trabajo.

Le doy gracias al maestro Yukihiro Minami por el apoyo y confianza que me brindó para llevar a cabo el proyecto. A mi director, el maestro Larry H. Escobar por aceptar nuestra propuesta de trabajo, por apoyarme y aconsejarme en las diferentes etapas que hicieron posible este escrito.

Agradezco de corazón a mis padres, a mis hermanas y a mi abuelita que durante toda la carrera me brindaron su respaldo y ánimo para poder llegar a la conclusión de este grado junto con todas las aventuras que viví en mis actividades extracurriculares.

A mi querida amiga de cabello chino que me acompañó en los momentos difíciles y que siempre ha estado ahí para motivarme a salir adelante y creer que siempre habrá días mejores.

A mi entrañable mejor amigo de derecho, quien sin duda alguna me ha enseñado a dar más de lo que nos exige de vida y a desear ser alguien que pueda colaborar en mejorar las condiciones en las que vive nuestro país.

A mi amiga Yoloxóchitl quién me guió en el estudio del procesamiento de imágenes y en el planteamiento y desarrollo de los algoritmos que se proponen para la detección de víctimas.

También agradezco al maestro Rubén quién me respaldo y acompañó junto con todos mis compañeros del club de robótica a emprender todos nuestros proyectos que nos llevaron a concursos internacionales y que al día de hoy me han encaminado a continuar con estudios de posgrado.

Índice desglosado

Índice de imágenes

Índice de imágenes

Índice de imágenes

Índice de tablas

Página

Glosario de términos

Término	Definición
BLOB	Región homogénea de imagen formada por la aplicación de un filtro
BREC	Búsqueda y Rescate en Estructuras Colapsadas
Centroide	Punto geométrico definido a partir de la forma de una región, el cual divide en dos partes iguales el área
Cuadro de visión	Cobertura del campo de visión de la lente de una cámara
DARPA	Defense Advanced Research Projects Agency
eBUSd	Controlador parte de la SDK que establece la comunicación con la cámara FLIR A35
FOV	Field Over View -- campo de visión de una lente
FPA	Arreglo plano focal, unidad de detectores sensibles a la radiación infrarroja
FPS	Frames per second - cuadros por segundo
GEV	Interfaz estándar GigE Vision
IR	Infrarrojo
LWIR	Long Wave infrared radiation - radiación infrarroja de longitud de onda larga
MIMO	Múltiples entradas, múltiples salidas
MWIR	Medium Wave infrared radiation - radiación infrarroja de longitud de onda media
NASA	National Aeronautics and Space Administration
NIR	Near infrared radiation wave - radiación infrarroja de longitud de onda corta
Nodo	Programa ejecutable en el sistema ROS
OpenCV	Biblioteca de procesamiento de imágenes (Open source Computer Vision

Término	Definición
ROS	Robot Operating System -- sistema de comunicación
SDK	Kit de herramientas de software para el desarrollo de aplicaciones (software development kit)
UDP	User Datagram Protocol
USAR	Búsqueda y Rescate Urbano
UVC	Clase de dispositivos de video que funcionan bajo el estándar USB 2.0
UWB	Ultra wide band (Ultra ancho de banda)
V4L2	Repositorio de ubuntu para el manejo de dispositivos UVC
Víctima no superficial	Persona atrapada bajo escombros a una profundidad tal que no puede ser detectada desde la superficie

Capítulo 1

Introducción

Durante años el desarrollo tecnológico ha sido impulsado principalmente por la industria privada y algunas dependencias gubernamentales con diferentes fines, por ejemplo, para cubrir o satisfacer alguna necesidad o carencia de un país y/o población (como la investigación en energías renovables), impulsar su economía, implementar sistemas de defensa militar, entre otros. Desafortunadamente junto con estos objetivos de bienestar social, suelen existir propósitos opuestos por parte de los encargados de estos proyectos o desarrolladores, como la búsqueda de poder, el control de una población o la sobreexplotación de recursos naturales.

En contraste el trabajo de investigadores, científicos e ingenieros extrañamente se efectúa con fines bélicos o destructivos, al contrario, se busca mejorar la calidad de vida, impulsando tecnologías que promuevan el cuidado del medio ambiente así como para continuar descubriendo el universo que nos rodea.

A pesar de existir personas con grandes ideas, otra de las fuentes de inspiración, además de la naturaleza, son las novelas de ciencia ficción, algunas de ellas actualmente han sido llevadas a la pantalla grande o se han tomado como base para crear historias, por ejemplo *Runaround* [M.1], escrito por Isaac Asimov y publicado en una revista en 1942. La trascendencia de esta historia, se atribuye al planteamiento final de las Tres Leyes de la Robótica, producto de anteriores obras del mismo autor y que fueron retomadas en sus posteriores publicaciones, han sido tan populares que incluso se han considerado como una normativa en el desarrollo de robots.

Otro foco de atención que ha sido motor del desarrollo tecnológico, son las diferentes necesidades y circunstancias que se presentan en situaciones de emergencia o catástrofes causadas por diferentes razones como por fenómenos naturales (como los huracanes, tornados, erupciones volcańicas, terremotos, etc), accidentes industriales (como lo ocurrido en 1986 en la planta nuclear de Chernobyl), atentados terroristas, entre otras. El factor común de todas, es el poner en riesgo la seguridad y supervivencia de las personas, en estas situaciones adversas, donde la tecnología podría contribuir a dar respuesta tanto en forma preventiva como en materia de acción.

El reto para los desarrolladores es grande, al tener que considerar las situaciones y circunstancias presentes en la zona donde ocurra la emergencia, basta con mencionar los diferentes tipos de derrumbes que se pueden dar, la presencia de materiales tóxicos, incendios, personas atrapadas bajo escombros, entre otros, siendo este último punto el tema de interés de este trabajo.

Tras el derrumbe de una construcción, sin importar cuál haya sido la causa, algunas de las personas que lograron resguardarse así como los testigos del evento, buscan brindar auxilio a todas aquellas personas heridas o atrapadas superficialmente por escombros, los cuales no representan un problema para su remoción. Sin embargo, cuando se trata de edificaciones grandes es muy probable que gente se haya quedado atrapada bajo pilas de escombros en espacios confinados, a profundidades cuya detección y rescate sólo pueda llevarse a cabo por equipos especializados, a dichas víctimas a lo largo del desarrollo de este trabajo se les denominará como víctimas no superficiales.

Con el paso del tiempo, al ser testigos de diferentes acontecimientos, grupos de rescate y dependencias gubernamentales han elaborado registros sobre las condiciones, técnicas y respuestas que se han presentado e implementado en diferentes emergencias. Analizando dicha información, se han generado manuales y protocolos para mejorar la respuesta a ciertos eventos, basados en la similitud y patrones observados, ejemplo de estos planes de acción y protocolos, son el Manual de campo de Búsqueda y Rescate en Estructuras Colapsadas (BREC) [1] y el manual de entrenamiento de Búsqueda y Rescate Urbano (USAR por sus siglas en inglés) [2] donde se establecen diferentes procedimientos, planeaciones y sugerencias a seguir para atender diferentes situaciones de emergencia, como el auxilio de personas sepultadas bajo escombros.

A pesar de los avances en tecnología auxiliar para la búsqueda de víctimas no superficiales, que se mostrarán a continuación, los resultados obtenidos no han logrado la perspectiva deseada, teniendo como uno de los múltiples objetivos el detectar signos vitales de las personas atrapadas para brindarles la atención adecuada y rescatarlas en el menor tiempo posible, argumentos que inspiraron la realización de este trabajo

1.1. Antecedentes

Para la detección de víctimas no superficiales, diferentes grupos de investigación y empresas han propuesto diferentes ideas e implementado equipos que auxilian a los rescatistas a lograr encontrar soluciones que permitan optimizar el tiempo de búsqueda. En esta labor los rescatistas contribuyen con sus experiencias de las diferentes situaciones a las que han acudido, punto clave que ha permitido estimar en cierto grado las condiciones y parámetros de diseño con los que se tiene que trabajar, obteniendo como resultados diferentes dispositivos y sistemas, algunos de ellos a continuación se describirán brevemente, sin cuestionar su funcionalidad.

1.1.1 Equipos para la detección de sonidos bajo escombros

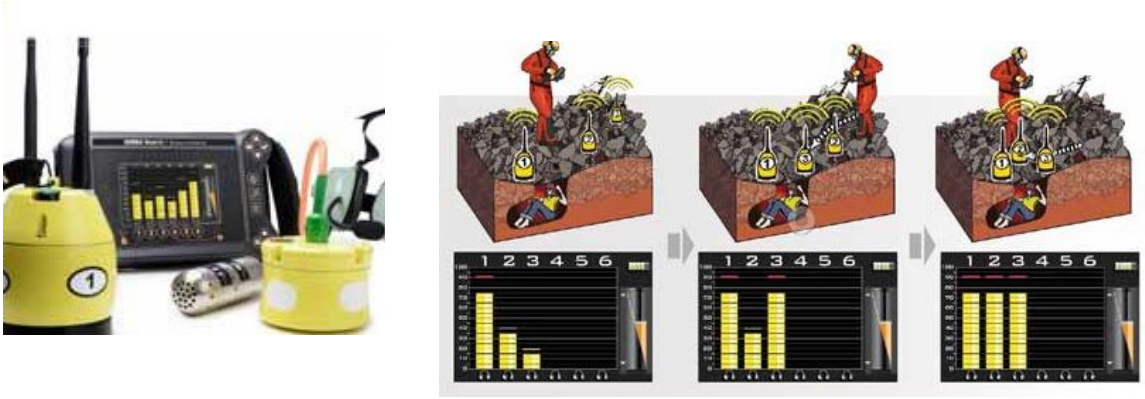
Son sistemas que perciben ondas sonoras transmitidas principalmente en medios sólidos utilizando geófonos y micrófonos susceptibles a la detección de llamados de auxilio de volumen moderado o percusiones periódicas que pueda llegar a hacer una persona golpeando algún objeto en algún escombro de tamaño considerablemente grande, como en columnas.

Para realizar la búsqueda, se utiliza normalmente una combinación de ambos sensores, con la finalidad de estimar la ubicación de fuentes sonido que provengan debajo de los escombros, para lo cual en primera instancia se secciona el área a escanear y se procura aminorar el ruido ambiente y demás fuentes que puedan llevar a obtener resultados falsos.

En cada sección se posicionan por lo menos 3 sensores, sobre escombros que puedan llegar a facilitar la detección, para triangular la intensidad de la fuente acústica que probablemente tenga como origen a una víctima, se analiza dicha información por medio de los diferentes procesamientos con los que cuente el equipo sumado a la experiencia y habilidad del operador.

Una vez que se logra estimar una posible víctima, los rescatistas se intentan comunicar con ella de viva voz o mediante sonidos periódicos, puesto que se desconoce la profundidad a la que se pueda encontrar. En caso contrario, se colocan los detectores en otras sección para continuar con las búsqueda [M.2].

La empresa LEADER comercializa un equipo de este tipo, el cual ha sido considerado como el más moderno, al contar con sensores de conexión inalámbrica a la unidad de adquisición y procesamiento, el cual se muestra junto con su forma de trabajar en la Figura 1.1.



A B

Figura 1.1 Equipo Lead Search y su implementación. La imagen A, muestra el equipo Leader Search con algunos de sus sensores, módulo de control y auriculares para el operador. La imagen B, ilustra la implementación del equipo para detectar a una persona atrapada bajo escombros, se puede observar como el operador toma lecturas moviendo los sensores para lograr triangular a la víctima.

Otra alternativa acústica, son los micrófonos colocados en soportes cilíndricos semirrígidos que se introducen en los espacios accesibles entre los escombros o a través de barrenos hechos por los rescatistas en losas o paredes, esperando detectar algún indicio de sonido o llamado de auxilio de alguna persona atrapada, antes de realizar alguna operación con maquinaria pesada e incluso poder utilizar explosivos.

1.1.2 Equipos de detección por inspección visual

Son parecidos a los micrófonos descritos en el párrafo anterior, porque consisten de una videocámara acompañada, generalmente acompañada de una lámpara con control de luminosidad, montados en el extremo de un bastón rígido o estructura semi flexible, para poder introducirlo en espacios inaccesibles para los rescatistas y perros de búsqueda, de la misma manera en los micrófonos. El video se transmite a una pantalla portátil donde el rescatista busca de forma visual encontrar a una víctima o de inspeccionar el área para llevar a cabo alguna maniobra de remoción de escombros o apuntalamiento. Los equipos más modernos han incorporado algoritmos de procesamiento de imágenes para detectar personas, sin embargo sus resultados son sujetos a la luminosidad y acceso del equipo.

Otro dispositivo que únicamente proporciona imagen es el fibroscopio, constituido principalmente por un cable de fibra óptica, cuyas proporciones permiten buscar con mayor facilidad dentro de los escombros, introduciendolo en espacios más estrechos, solucionando el problema del tamaño, pero no la necesidad de tener una fuente de luz visible.

Como alternativa, también se ha experimentado con cámaras térmicas, que ofrecen buenas características para la detección de personas, entre ellas, que pueden proporcionar información en ausencia de luz visible, sin embargo, continúa siendo constante la limitante de realizar búsquedas superficiales y el encontrar la forma de introducir el equipo, por ello se comenzó a tratar de superar esta problemática, teniendo como una propuesta de solución la implementación de radares.

1.1.3 Radares para detectar personas

De las misma forma en la que se detectan objetos en campo abierto por medio de ondas de radio, se planteó la idea de crear radares capaces de encontrar personas atrapadas bajo escombros identificando características inconfundibles de los seres humanos, como los signos vitales, o movimientos naturales del cuerpo, como lo son el pulso cardíaco o el movimiento del pecho al respirar.

Los conceptos de implementación del radar, utilizan un arreglo de antenas que emiten un barrido de ondas electromagnéticas en un intervalo de frecuencias, del orden de los gigahertz, con un ancho de banda mayor que los 500 MHz, denominado como Ultra Ancho de Banda, (UWB por sus siglas en inglés), intervalo en el que las ondas podrían transmitirse por los escombros esperando que se reflejen exclusivamente en personas.

Actualmente esta tecnología está desarrollándose por diferentes grupos de investigación de carácter público y privado, mostrando la viabilidad de este concepto los resultados obtenidos de los primeros prototipos, como el proyecto FINDER, cuyas siglas significan Finding Individual for Disaster and Emergency Response, que es un radar producto de la colaboración entre de la *National Aeronautics and Space Administration* (NASA), el laboratorio de Propulsión Jet de la NASA (JPL) y el Departamento de Dirección de Ciencia y Tecnología de Seguridad Nacional (DHS) [M.3].

Este radar ha sido probado en ambientes controlados a unos cuantos metros de distancia, en escenarios simulados obteniendo buenos resultados y puesto en práctica en condiciones reales para buscar víctimas tras el terremoto de Nepal, en donde se reportaron resultados similares a las pruebas en ambiente controlado. Razón por la que se cuestionan el funcionamiento de equipos de empresas privadas que afirman brindar mejores resultados.

El equipo consiste de un contenedor que transporta el arreglo de antenas y permite colocarlas sobre las pilas de escombros y de una especie de tableta electrónica donde se visualizan los resultados, además de permitir realizar configuraciones al equipo. Al igual que los equipos citados hasta el momento, se escanea un área seccionada para no buscar dos veces en el mismo lugar, a no ser que sea necesario [4]. La Figura 1.2 se muestran algunos de estos radares, así como su método de implementación.

A B

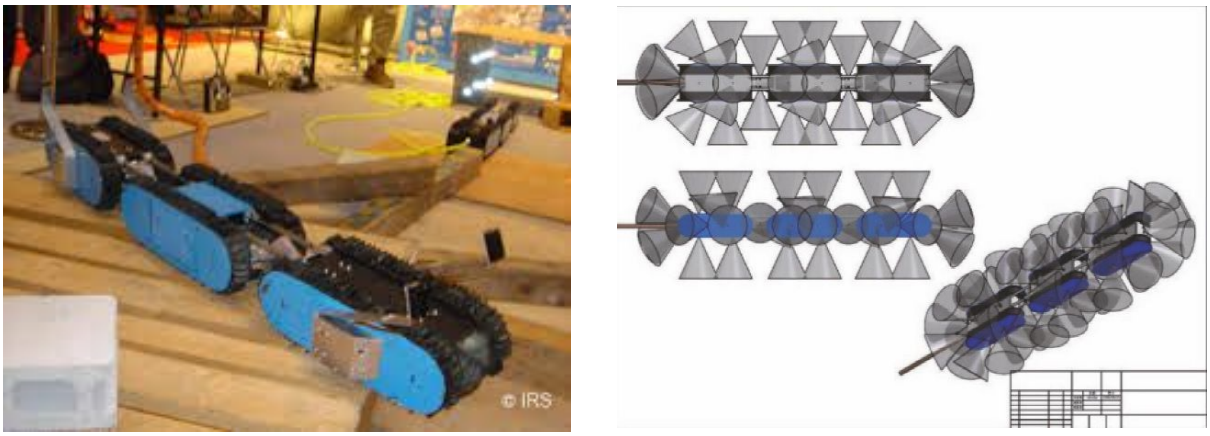
Figura 1.2. Equipo Lead Finder y su implementación. La imagen A, muestra el equipo Leader Scan constando únicamente de su módulo de control y del contenedor que tiene el radar. La imagen B, ilustra la forma en la que es implementado el equipo para la búsqueda de víctimas bajo escombros.

Los equipos anteriormente mencionados tienen un punto en común, todos requieren zonas accesibles para que los operadores para que puedan colocar los instrumentos e implementarlos, a pesar de que algunos, como las cámaras de búsqueda, puedan ser introducidos de forma parcial entre los escombros o derrumbes. Motivo por el cual un área de la robótica se ha interesado por desarrollar robots móviles capaces de explorar zonas del siniestro inaccesibles y peligrosas para un ser humano o perro, incorporando equipos de vídeo, audio o de cualquier otro tipo que permitan detectar víctimas.

1.1.4 Robótica para búsqueda y rescate

En paralelo al desarrollo de sistemas de detección de víctimas no superficiales, también se trabajan con plataformas móviles que puedan llevar alguno de estos dispositivos de búsqueda a zonas de difícil acceso o peligrosas. Como propuestas de solución se ha planteado la idea de utilizar robots móviles con sistemas de tracción que permitan desplazarse en espacios estrechos y sobre las superficies hostiles que se dan tras un derrumbe.

El Dr. Satoshi Tadokoro encabeza un grupo de trabajo que ha desarrollado un conjunto de prototipos de herramientas y robots móviles que tienen como fin auxiliar a los rescatistas en diferentes actividades. Uno de ellos es el es el proyecto Souryu IV [4], que es un robot con forma de serpiente capaz de introducirse entre los escombros y buscar por medio de sus 32 cámaras de vídeo alguna víctima, la Figura 1.3 muestra una fotografía del robot y la distribución de sus cámaras.



A B

Figura 1.3. Robot Souryu. La imagen A, es un fotografía del prototipo Souryu IV, robot diseñado para introducirse en espacios reducidos y entre los escombros además de poder desplazarse a través de ellos. La imagen B, muestra la configuración de sus 32 cámaras con las cuáles el operador, desde la estación de control puede ver todo lo que rodea el robot.

La miniaturización tanto los sistemas de detección como de los móviles que podrían portarlos, como los robots, es una tarea compleja que depende directamente del desarrollo tecnológico y del ingenio de los desarrolladores, por eso de forma alterna se han propuesto entrenar de insectos como las cucarachas o ratones que contribuyan a la búsqueda de víctimas no superficiales por su habilidad de poder introducirse sin problema alguno bajo los escombros.

Sin embargo, como el avance en materia electrónica ha mantenido un ritmo constante, al grado de colocar en nuestras manos dispositivos cada vez más pequeños en cuestión de pocos años, un concepto propuesto a finales del siglo pasado se podría emplear para la detección de víctimas, por lo que se describe brevemente a continuación.

1.1.5 Smartdust

El concepto surgió en la primer mitad de la década de los 90 en la corporación Research And Development (RAND) y en Defense Advanced Research Projects Agency (DARPA), por una serie de estudios y talleres de trabajo. La propuesta se basa en la tecnología de manufactura de MEMS (Micro ElectroMechanical Systems) y consiste en crear pequeños sistemas (cuyo volumen sea menor o igual que un centímetro cúbico) como robots o computadoras que puedan monitorear por medio de sensores o transductores diferentes tipos de variables físicas, cuyos datos puedan ser transmitidos por medio de una red local inalámbrica para realizar su procesamiento o controlar diferentes tareas en conjunto, sin una invasión macroscópica [M.4].

En 2001 el concepto fue divulgado con la finalidad de lograr su desarrollo, partiendo de un interés militar. Esta idea se podría materializar con el fin de detectar personas, puesto que solo

se tendría que distribuir los detectores en forma de polvo en toda la zona del derrumbe y de alguna manera hacer que proporcionen información de la ubicación donde la persona está atrapada, sus signos vitales, entre otras ideas.

El enfoque de la investigación y del trabajo para detectar víctimas no superficiales, tiende a la detección de personas por sistemas con un determinado grado de inteligencia artificial, buscando identificar por lo menos un signo vital, permitiendo encontrar víctimas inconscientes atrapadas en lugares inaccesibles para un perro de búsqueda o rescatista.

Los avances en diferentes campos de la ciencia e ingeniería, reflejados en artículos y revistas científicas así como en diferentes prototipos de sistemas para la detección de personas atrapadas bajo escombros, dan pie a múltiples opciones que pueden ser el tema de este trabajo, por ello en el siguiente el capítulo se describe el análisis del problema.

1.2. Objetivos

Objetivo general

Diseñar, construir e implementar un sistema base para detectar víctimas no superficiales mediante el procesamiento de señales, enfocado a trabajar en condiciones reales y que permita implementar algoritmos para la detección de algún signo vital.

Objetivos particulares

I. Proponer un sistema electrónico compacto para la detección de víctimas atrapadas dentro de los escombros.

II. Diseñar el sistema para ser enfocado a detectar algún signo vital.

III. Implementar el sistema mínimo para la ejecución de algoritmos de búsqueda.

IV. Realizar pruebas en ambientes simulados de los algoritmos de detección propuestos.

1.3. Planteamiento del problema

Los derrumbes de construcciones provocados por alguna explosión, fallos estructurales o por consecuencia de un terremoto generalmente dejan atrapadas a personas bajo escombros.

El estado de salud y las condiciones en las que se encuentren las víctimas no superficiales es muy diverso, sin embargo, se tiene noción de que la caída de paredes, techos y otros elementos de la construcción provocan golpes que pueden dejar inconscientes a las personas, herirlas en diferentes magnitudes e incluso inmovilizar de forma total o parcial a la víctima, por citar algunos daños colaterales. En el mejor de los casos, la persona pueda estar solo atrapada en un espacio confinado, por ejemplo debajo de algún mueble o en algún triángulo de seguridad, sin lesión alguna.

Rescatar estas víctimas implica ocuparse tanto por la seguridad del personal de rescate como de la víctima. Actualmente se detectan a estas personas atrapadas bajo escombros, utilizando perros de búsqueda que olfatean el área del derrumbe, indicando donde podría localizarse una persona, teniendo resultados en función del entrenamiento del animal. El éxito de su detección está sujeto a períodos de trabajo y condiciones ambientales, por ello también se implementan instrumentos y sistemas de detección electrónicos operados por personal capacitado.

Sin embargo, estas acciones tienen lugar después de inspeccionar el lugar del derrumbe, puesto que primero se hace una planeación y coordinación de todas las operaciones a realizar inicialmente. El tiempo de supervivencia de una víctima depende totalmente de las condiciones en las que se encuentre atrapada, acorde a las estadísticas basadas en hechos reales, el protocolo internacional de búsqueda y rescate contempla las primeras 72 horas sucedido el evento para localizar y auxiliar a la mayor cantidad de víctimas.

Con el objetivo de reducir el tiempo para iniciar las acciones de búsqueda, recientemente se están empleando robots móviles de exploración capaces de desplazarse sobre y entre los escombros, equipados con diversas cámaras y sensores para identificar los riesgos presentes en el área del derrumbe, así como para detectar a víctimas.

México por su ubicación geográfica está sujeto a diferentes fenómenos naturales y actividad sísmica, los cuales desafortunadamente han cobrado vidas por falta de preparación para hacer frente a dichas circunstancias. Uno de los acontecimientos que ha provocado un gran desastre en la zona centro del país, fue el terremoto de 1985, hecho que resaltó la necesidad de ocuparse tanto en materia de prevención como en la capacidad de respuesta ante este tipo de desastres e impulsó la realización de este trabajo.

1.4. Resumen de capítulos

Con la finalidad de ofrecer un amplio panorama de las partes que forman este trabajo escrito, a continuación se describe el contenido de cada uno de los capítulos desarrollados.

El ***capítulo 1*** presenta el tema de interés, describiendo el panorama actual respecto a los equipos electrónicos auxiliares para la búsqueda de víctimas no superficiales, con objetivo de definir las metas propuestas y de plantear el problema.

Posteriormente se analiza el planteamiento del problema en el ***capítulo 2,*** donde se proponen diferentes opciones de solución, basados en la documentación realizada respecto del tema, comprendiendo el estudio de la técnica e información de protocolos de acción de organizaciones internacionales especialistas. Dichas opciones se estudian y analizan para definir la solución propuesta que se desarrolla en este trabajo.

El proceso de diseño del sistema de detección es desglosado en el ***capítulo 3***, definiendo las especificaciones de diseño acorde a los objetivos y a la solución propuesta, se analizan las opciones disponibles de dispositivos para construir el sistema, incluyendo su procedimiento de selección y el razonamiento que definió cuáles utilizar. También se presentan los diseños conceptuales de operación del sistema, definiendo la implementación para ponerlo en operación.

La integración de todos los elementos que componen el hardware del sistema, se define en el ***capítulo 4*** al exponer los requerimientos de software de cada uno de los dispositivos, la forma en la que entregan la información así como su interconexión. Se describen los protocolos de comunicación tanto de adquisición de datos de los equipos como de su intercambio en todo el sistema dando lugar a la arquitectura completa del sistema de detección, listo para ejecutar los procedimientos para la detección de víctimas.

Uno de los signos vitales que se pretende detectar, implementando la cámara térmica, es el pulso cardíaco, para lograr esta meta, en el ***capítulo 5*** se describen los algoritmos de procesamiento de video para permitir identificar esta señal de vida, por ello se abordan los temas de detección de fuentes de calor, el reconocimiento total y parcial de partes del cuerpo de una persona, la segmentación de imagen para definir regiones de interés y la presentación de los resultados de las aplicaciones mencionadas.

Los resultados de las pruebas realizadas en ambientes controlados y simulados, de la implementación de los algoritmos de detección mencionados en el capítulo anterior, se muestran en el ***capítulo 6***.

El ***capítulo 7*** comprende las conclusiones del trabajo, que consta del análisis de los resultados obtenidos en las diferentes pruebas realizadas y las recomendaciones del trabajo que se podría desarrollar para mejorar los resultados.

Finalmente, se presentan las diferentes fuentes de información consultadas para el desarrollo de este trabajo, que consistieron en artículos científicos de investigaciones realizadas así como artículos en línea en diferentes sitios en la red. Además de adjuntar en el anexo algunos de los procedimientos de configuración, códigos y diagramas del trabajo implementado.

Capítulo 2

Análisis del problema

Para definir la línea de trabajo se realizó un análisis considerando como modelo el proceso de diseño en ingeniería [14], con la intención de contribuir en medida de lo posible, a la búsqueda de tecnología auxiliar para situaciones de emergencia, particularmente para la detección de víctimas no superficiales. El análisis consistió de las siguientes 5 etapas; documentación referente a los equipos existentes y trabajos en desarrollo (estado del arte expuesto en la introducción), especificación del problema de estudio, búsqueda y propuesta de soluciones, el análisis y evaluación de la soluciones planteadas.

En el desarrollo de este capítulo se mostrará el estudio de las diferentes condiciones, factores y escenarios presentes en las áreas de algún derrumbe así como aquellas en las que se podría encontrar una persona atrapada entre los escombros, los tipos de espacios confinados formados tras el derrumbe de una construcción y el posible estado de salud de la persona. Se analizarán los diferentes equipos electrónicos de búsqueda comprendidos en los antecedentes, con la finalidad de proponer ideas de solución que serán ponderadas con base en los criterios de evaluación establecidos y de las recomendaciones indicadas por rescatistas en los diferentes manuales y protocolos consultados así como la información obtenida al entrevistar al personal de entrenamiento de los perros de búsqueda de la UNAM y finalmente con toda esta documentación, se tome una decisión del trabajo a desarrollar.

2.1. Especificación del problema

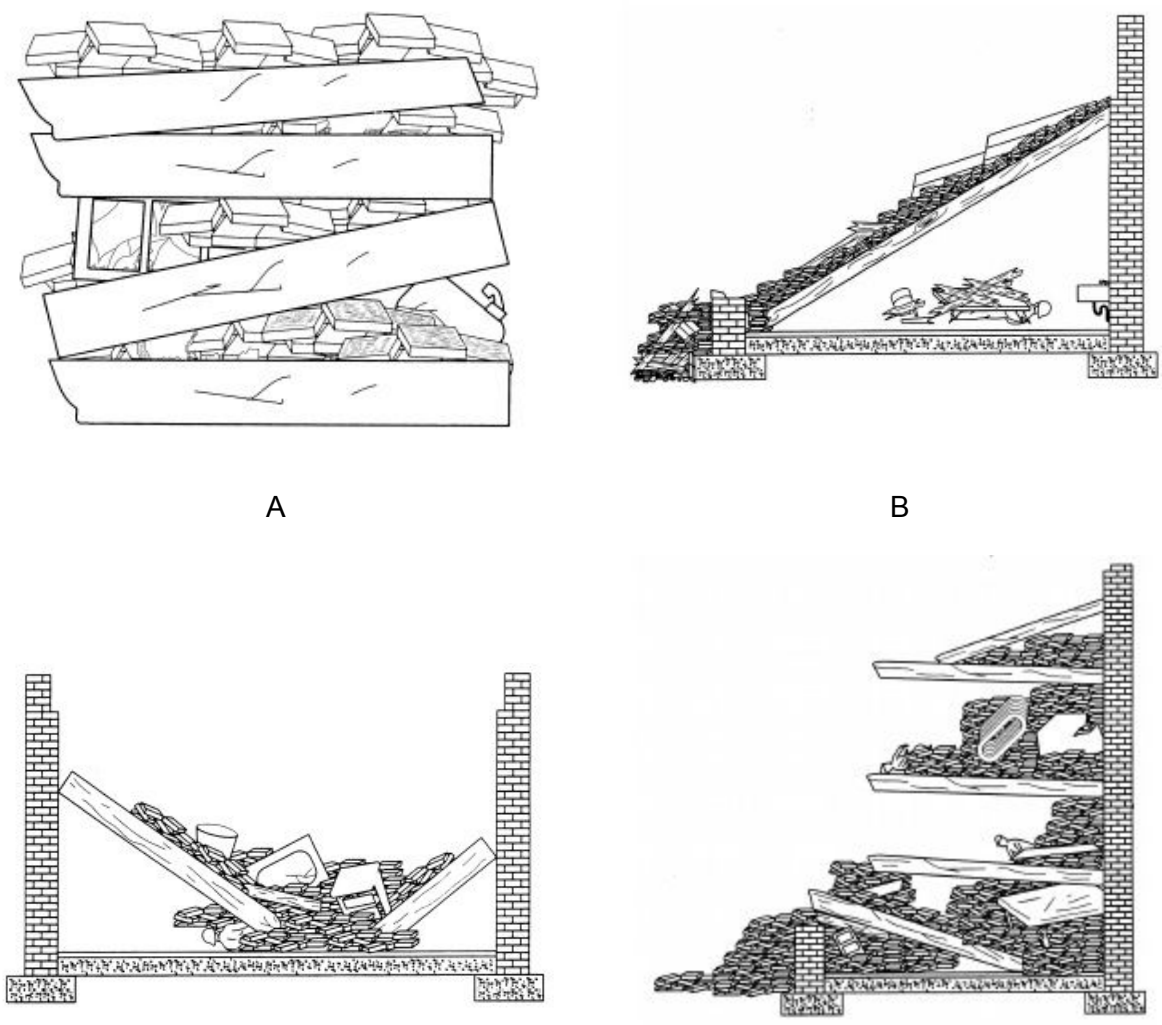
Actualmente la detección de víctimas no superficiales se realiza combinando diferentes técnicas utilizando equipo electrónico especial, personal capacitado para su uso y perros de búsqueda. Las condiciones en las que se puede encontrar una víctima son variables, pero no totalmente desconocidas, gracias a la recopilación de información de diversos acontecimientos, se cuenta con la observación de determinados patrones sobre los lugares y condiciones en las que se puede localizar a una persona bajo los escombros.

Dicha información fue posible consultarla en los diversos manuales y libros de entrenamiento de personal de rescate disponibles en la red, elaborados por diferentes organizaciones civiles y gubernamentales, entre ellas la *United States Agency International Development* (USAID) y *Federal Emergency Manager Agency* (FEMA). A continuación se presenta un resumen de toda esta información, que definen hechos, situaciones y escenarios que competen el funcionamiento del sistema de detección.

2..1.1 Daños en construcciones colapsadas

Previo a la realización de cualquier actividad, la brigada de rescatistas debe hacer una evaluación de la zona del siniestro, con el fin de identificar los daños ocasionados en las edificaciones, riesgos latentes e investigar con los testigos de lo ocurrido si tienen noción de personas atrapadas entre los escombros, para ello se emplean los procedimientos establecidos indicados en el manual BREC [1] y USAID [2], donde además contempla el auxilio de personas lesionadas pero no atrapadas y de las víctimas superficiales bajo escombros que pueden removerse sin complejidad alguna.

Al tener diferentes tipos de edificaciones por consiguiente, tenemos diferentes materiales de construcción, tipos de colapso y/o derrumbes de las construcciones así como distintas fallas estructurales, acorde a la causa que haya dado lugar al daño. Teniendo conocimiento de ello, es posible lograr identificar junto con la información obtenida de los testigos, algunos posibles lugares en donde se podrían encontrar personas con vida, acorde al tipo de colapso presente, como los que se muestran en la Figura 2.1



A B C D

Figura 2.1 Tipos de colapsos; (A) de apilamiento, donde los espacios vitales son muy limitados y difíciles de acceder, (B) apoyado al piso, suele dejar un espacio vital aislado triangular, (C) en forma “v”, debido a su forma suele dejar dos espacios vitales aislados y (D) suspendido que es peligroso e inestable

Acorde al tipo de colapso bajo del cual se encuentra una víctima así como del equipo y herramientas disponibles, se toman y establecen las medidas y prioridades necesarias para poder rescatar a la persona, de ser esto posible.

Realizar las inspecciones y evaluaciones de la zona de emergencia consume parte del tiempo, conocido como “las 72 horas de oro”, intervalo que inicia a partir de la culminación del hecho que provocó el derrumbe y que corresponde al período en el que una persona tiene mayor probabilidad de sobrevivir al haber quedado atrapada bajo escombros. Por ello toda acción a

realizar por el equipo de rescate es perfectamente planeada con el fin de rescatar a la mayor cantidad de personas posibles sin que sufran más daños o se ponga en riesgo a los rescatistas.

2.1.2 Zonas de seguridad y espacios vitales

Acorde al hecho que dio pie al derrumbe de una construcción y de su diseño, normalmente los elementos de construcción al colapsar (sean columnas, vigas, paredes, entre otros), en ocasiones acorde al tipo de derrumbe tienden a formar espacios confinados debajo de ellos, capaces de proteger a las personas, de forma parcial de la caída de escombros [2].

Por reglas de protección civil estos espacios que se pueden formar, conocidos como triángulos de vida, deben de estar señalizados principalmente en las edificaciones que pueden albergar una gran cantidad de gente, sobretodo si cuentan con más de dos pisos, ya que en caso de un sismo, no es posible desalojar a todos, por los riesgos presentes al hacerlo. Como parte de la cultura de prevención, al estar cerca de uno de estos señalamientos, las personas deben ubicarse en estas áreas indicadas o de no ser posible, debajo de muebles que puedan brindar protección como los triángulos de vida, ante un evento que pueda ocasionar el colapso de la construcción.

Una persona resguardada en un triángulo de vida o en un espacio similar a este, tiene alta probabilidad de sobrevivir, a pesar de estar cubiertos grandes pilas de escombros, lejos de la superficie, convirtiéndolos en los puntos estratégicos de búsqueda para el sistema de detección.

2.1.3 Condiciones y estado físico de las víctimas no superficiales

La caída de fragmentos y elementos de construcción en un derrumbe provocan diferentes daños y lesiones a las personas que quedan atrapadas bajo los escombros. En la detección y búsqueda es una obligación ver por la seguridad del personal y de las víctimas porque llevar a cabo algunas acciones para rescatar, como la remoción de escombros grandes con maquinaria pesada, podría ocasionar diferentes daños mortales colaterales para ambas partes e incluso poner en riesgo la vida de otras víctimas que aún no hayan sido detectadas.

Poder estimar el estado de salud de las víctimas no superficiales, ayudaría a las brigadas de rescate en la planeación de las acciones de auxilio y a la asistencia médica de la persona, por ello el enfoque actual de los equipos de detección tienen por objetivo el detectar signos vitales que permitan poder llegar a proporcionar esta información.

Al quedar atrapada una persona bajo o entre escombros, se puede mantener con vida acorde a las condiciones en las que se encuentre y a otros factores, algunos de ellos son:

- Estar confinada en un espacio estrecho con poca iluminación
- Quedar atrapada en un triángulo de vida
- Estar en presencia de agentes tóxicos
- La magnitud de las lesiones heridas que le haya provocado el derrumbe
- Tener atrapada alguna parte de su cuerpo, e incluso llegar a tenerla mutilada
- Estar inconsciente
- Su estado psicológico frente a la situación
- La edad de la víctima

USAID en su manual de campo BREC [5], expone detalladamente las condiciones citadas previamente y en función de esta información realiza una estimación del índice y probabilidad de supervivencia de las personas bajo escombros, así como una planeación de acciones sugeridas para dar respuesta, en función del tiempo transcurrido iniciando este al finalizar el evento que dio lugar al derrumbe, dicha estimación se muestra de forma gráfica en al Figura 2.2

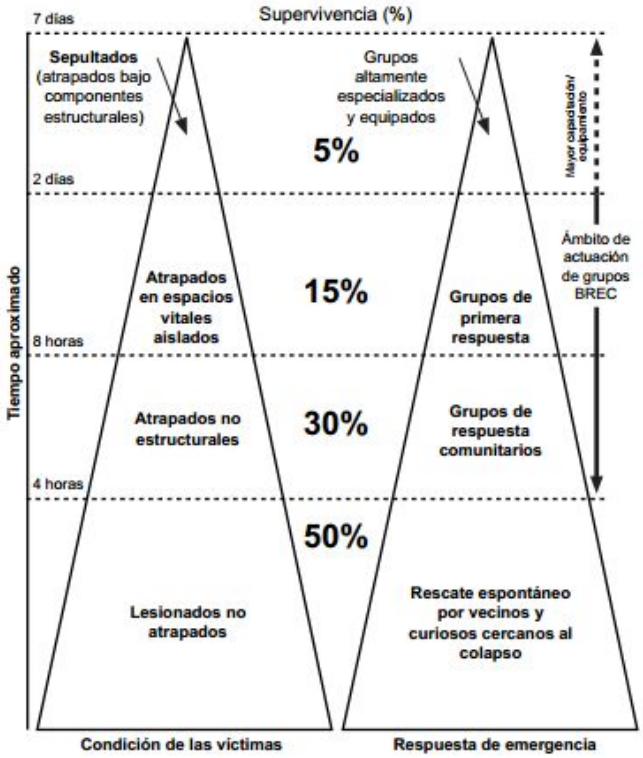


Figura 2.2 Desarrollo de actividades de búsqueda y rescate en el transcurso del tiempo finalizado el derrumbe. Los porcentajes señalados indican la probabilidad de localizar sobrevivientes que hayan quedado atrapados entre los escombros.

2.1.4 Factores meteorológicos

Además de los factores adversos descritos anteriormente, se suman las condiciones climatológicas que se presenten al realizarse todas las acciones de respuesta a la intemperie. El desempeño del personal, de los equipos electrónicos de detección y de los perros de búsqueda es afectado en diferentes formas y proporciones por la presencia de condiciones [1], como:

- Fuertes corrientes de aire
- Tormentas eléctricas
- Lluvias
- Fuentes de calor y temperatura ambiente
- Humedad
- Elevada contaminación acústica
- Altas temperaturas ambiente

Parte de la respuesta a estos factores, son los sistemas de protección ambientales de los equipos electrónicos contra diferentes aspectos, como la humedad, polvo, entre otros. Sin embargo no dejan de necesitar además operadores capacitados para buscar víctimas en las probables zonas donde puedan estar.

2.1.4 Coordinación de voluntarios

Generalmente, testigos o gente aledaña a la zona en donde ocurrió el derrumbe, se ofrecen de voluntarios para realizar la búsqueda de personas, remover escombro, entre otras actividades, sin embargo su falta de preparación e instinto de solidaridad llega a entorpecer el trabajo de las brigadas de rescate, e incluso llegan a poner en riesgo su vida y la de aquellas víctimas no superficiales.

Por ello en los protocolos y manuales citados anteriormente se plantean estrategias para coordinar a todos los voluntarios que son indiscutiblemente necesarios, para que se resguarde su seguridad y se puedan desarrollar el plan de acciones para atender la emergencia. Se hace mención de los voluntarios, porque sin duda alguna las actividades que se realicen en toda la zona de búsqueda provocarán diferentes factores que pueden llegar a afectar al funcionamiento de un sistema electrónico de detección.

2.1.5 Recomendaciones para el desarrollo de tecnología auxiliar

Como resultado de las múltiples condiciones y dificultades que han enfrentado los equipos de rescate, en diferentes situaciones de emergencia, se han creado registros y documentación sobre las formas de respuesta, a su vez con base en sus experiencias, también han hecho recomendaciones y propuestas para el desarrollo de equipo auxiliar, lo cual retroalimenta a los

desarrolladores, tanto para la detección como para las actividades de rescate. Algunas de estas sugerencias son:

1. Diseñar equipos compactos, portátiles y ligeros, con la protección suficiente para afrontar las condiciones del campo de trabajo.
2. Proporcionar resultados sin la necesidad del criterio de apreciación del operador.
3. Tener un funcionamiento más simple que prescinda de largas capacitaciones.
4. Soporte para largas jornadas de trabajo
5. Que no expongan al personal a riesgos innecesarios, al momento de utilizar el equipo entre los escombros.

Estos puntos, serán considerados en el análisis de los antecedentes y también en la evaluación de las soluciones propuestas, para poder incorporarlos como criterios que contribuyan a definir la solución final, aprovechando estos información contenida en los diferentes manuales y protocolos de búsqueda y rescate [1].

2.2. Búsqueda de soluciones

2.2.1 Análisis de antecedentes

El objetivo general del proyecto es la detección de víctimas no superficiales, lo que implica localizar sobrevivientes tras el derrumbe de una construcción, detectando a través o introduciendo el sistema, entre las pilas de escombros.

Las herramientas, técnicas y equipos que se utilizan actualmente son funcionales, pero requieren de que los operadores puedan tener acceso a la zona de búsqueda, además de contar con condiciones específicas de trabajo, por ejemplo, la eliminación de la mayor cantidad de fuentes de sonido, para la implementación de equipos de audio. Sin embargo ninguno de estos equipos por sí solo, puede proporcionar una detección certera por las condiciones en las que trabajan, es por ello que los métodos son combinados.

Los perros de búsqueda son la forma más eficaz para detectar víctimas bajo escombros, sin embargo su rendimiento no es el adecuado para responder a este tipo de emergencias, debido a que su olfato se satura después de un breve período de tiempo, sin olvidar todo lo que el entrenamiento del animal. Por ello se comenzó a pensar en alternativas tecnológicas, en desarrollar equipos electrónicos capaces de ubicar personas detectando a distancia algún signo vital con la finalidad de mejorar las actividades de rescate y asignar prioridades acorde al estado de salud estimado de la víctima.

Algunos de los radares UWB que están desarrollándose, tienen como objetivos de detección localizar el pulso cardíaco y/o el movimiento del pecho generado por la respiración de una persona, entre los escombros a profundidades menores a dos metros (dato de pruebas reales,

no ideales) además de estimar la distancia a la que se encuentra la víctima. Sin embargo, su funcionamiento está sujeto a que las ondas de radio logren viajar a través de los escombros y sean reflejadas específicamente y únicamente por los puntos de interés, lo que permitiría encontrar personas que hayan quedado inconscientes o que no puedan pedir auxilio.

Otra manera de identificar la respiración es detectando la emisión de dióxido de carbono durante la exhalación, el cual es detectado y mesurado por medio de radiación infrarroja (IR), como en el equipo médico capnógrafo [M.5], lo que da pie a visualizar la presencia del gas por medio de una cámara infrarroja, logrando detectarlo sin la necesidad de tener que estar a una muy corta distancia de la fuente de emisión del gas, como es el caso de algunos sensores.

En la búsqueda de soluciones, también se han propuesto el uso de robots para las tareas de inspección, búsqueda, apuntalamiento y rescate en lugares peligrosos para una persona e incluso inaccesibles, lo que representa trabajar en los diversos sistemas que integren a los robots. El desarrollo de estos proyectos por parte de diferentes institutos, universidades y empresas privadas ha sido impulsado e incentivado con la realización de competencias internacionales, cuyos objetivos van desde la exploración de una área de desastre simulada hasta la creación de barrenos en paredes, todo por parte de un robot autónomo o de operación remota.

Algunos de los principales eventos en los que se prueban prototipos y comparten conocimientos en esta área, son la Robocup [M.8], donde en su categoría de Rescue reta a construir un robot tipo “rover” capaz de navegar en un ambiente simulado de escombros, que realice un mapa de la zona que explora, transmitir video en tiempo real a una estación de control, detectar víctimas simuladas con muñecos que emiten CO_2, emanan calor como una persona, hacen ruidos y presentan ciertos movimientos, ocultos entre escombros, entre otras tareas.

Y el Robotics Challenge organizado por DARPA, cuyo propósito es llegar a crear un robot humanoide capaz de realizar actividades de búsqueda y rescate de víctimas, en situaciones de desastre, donde el robot tiene que ser capaz de cerrar válvulas de paso, hacer barrenos en paredes y hasta conducir un vehículo. Estos robots son sistemas integrados por módulos que realizan diferentes funciones, partiendo desde la más básica hasta la más compleja, por ello, también se ha contemplado el dividir todo este trabajo en robots especializados, que trabajen de forma colaborativa.

El uso de equipos de video para la detección está supeditado al acercamiento de la cámara al lugar que se desea explorar, además de contar con iluminación. El introducir la cámara implica tener que afrontar las diferentes condiciones en una zona de desastre, como la presencia de partículas de polvo del material de construcción que pueden obstruir la lente, limitando el campo de visión. A raíz de esto se han implementado diferentes ideas para solucionar estos problemas, una de ellas como se mencionó en los antecedentes, es el uso de cámaras

térmicas las cuales son capaces de estimar la temperatura a distancia, proporcionar una imagen en completa oscuridad e incluso a través de humo.

Las cámaras térmicas son equipos utilizados principalmente para monitorear procesos de carácter industrial y en sistemas de videovigilancia, por ello diferentes empresas han avanzado en el desarrollo de esta tecnología a tal grado, que actualmente ya existen dispositivos de alta definición, mayor sensibilidad y de menores dimensiones, además de que los costos de estos equipos han ido a la baja.

Sin embargo, a pesar de ser un equipo industrial, algunos investigadores han realizado trabajos con estas cámaras térmicas orientados al procesamiento de imágenes y video aplicados a sistemas de vigilancia para la detección de personas [12], usos biomédicos, como en la medición de CO_2 emitido durante la exhalación en la respiración, para detectar ciertas enfermedades [11], e incluso en la medición del pulso cardíaco. Ejemplo de esta última aplicación es el algoritmo propuesto en la universidad de Louisville, EUA, que puede medir el pulso cardíaco observando las pulsaciones perceptibles de la arteria carótida en ciertas partes del cuello y en la cara [5].

Implementar una cámara térmica para la detección de víctimas no superficiales, implicaría introducir la cámara entre y bajo los escombros, puesto que no puede proporcionar imágenes a través de éstos, resuelto dicho problema se podrían implementar los algoritmos propuestos en diferentes trabajos para la detección y monitoreo de signos vitales, además de proponer algunas modificaciones con base en los factores y circunstancias que se se podrían presentar en condiciones reales.

2.2.2 Soluciones propuestas

Con base en los antecedentes y en el análisis realizado, a continuación se presentan las soluciones propuestas para el problema planteado.

Propuesta 1. Sistema de seguridad inteligente

En materia de prevención se contempló la elaboración de un *sistema de seguridad inteligente* que se instale en cualquier edificación, cuyo control de mando sea protegido por un contenedor tan resistente como una caja negra, la idea es que el sistema sea capaz de monitorear las diversos parámetros en la casa o edificio en todo momento para guardar y tener el registro en caso de presentarse un derrumbe, de los lugares donde se pudiesen haber quedado atrapadas personas.

Para detectar la presencia de personas se podría usar cámaras de videovigilancia que identifiquen personas de forma automática y/o sensores piroeléctricos, conectados a un sistema embebido con conexión inalámbrica, así por medio de una aplicación en un teléfono celular o tablet los recatistas puedan acceder a esta información. A su vez se le podría

incorporar una conexión al sistema de alerta sísmica, así como sirenas o señalamientos que ayuden a localizarlo en caso de no poder obtener la información vía inalámbrica.

Propuesta 2. Radar UWB

Al ser un equipo en vías de desarrollo que comprende varias áreas de ingeniería, la propuesta es iniciar a construir una unidad de este tipo contando con la colaboración de más estudiantes afines a los requerimientos del radar, diseñarlo e implementarlo para probar diferentes algoritmos para detectar los dos objetivos propuestos, la detección del pulso cardíaco y el movimiento del pecho debido a la respiración.

Empresas privadas como US Leader y dependencias gubernamentales como la NASA, ya cuentan con prototipos funcionales, que brindado resultados en pruebas de campo sujetos a las condiciones presentes en la zona del derrumbe, sin embargo estos avances han demostrado ser una opción viable, con mucho trabajo pendiente por desarrollar, por ello se propone trabajar sobre esta tecnología para tratar de contribuir en la materialización de este dispositivo, ya que puede ser operado desde la superficie sin invasión alguna dentro de las pilas de escombros.

Propuesta 3. Sistema de detección de pulso cardíaco

Basados en algunas aplicaciones biomédicas, se propone crear un *sistema* que sea *capaz de detectar el pulso cardíaco* de una persona por medio de procesamiento de video de una cámara térmica, para ello se podría identificar la silueta del cuerpo de una persona o partes específicas que permitan realizar la detección, por ejemplo brazos, cuello y cabeza, donde se podría trabajar en distinguir las variaciones de calor debidas al torrente sanguíneo en determinadas arterias.

Este equipo deberá de ser diseñado para ser introducido entre los escombros, con la protección necesaria para resistir las condiciones de trabajo, para que realice la detección de personas sin involucrar el criterio de su operador el cual observará los resultados obtenidos de forma gráfica junto con la transmisión de video. Dicho en otras palabras, desarrollar una cámara inteligente.

2.3. Decisión de solución

Se han puesto en práctica diferentes formas e ideas para detectar víctimas no superficiales, incluso los conceptos que ya se habían utilizado y descartado, han vuelto a ser considerados con un enfoque diferente por nuevos grupos de trabajo, obteniendo buenos resultados y conclusiones, como la estimación de la posición de un robot serpiente por medio de ecolocalización [13]. A continuación se aborda el análisis de las soluciones propuestas.

2.3.1 Análisis de soluciones propuestas

Dos de las tres ideas propuestas como solución, tienen por objetivo principal la detección de por lo menos un signo vital, factor que resulta ser muy útil puesto que se pueden encontrar a personas inconscientes. Al combinar cualquier método y equipo que detecte alguna señal de vida, con los perros de búsqueda, otorgaría mejores resultados sobre la ubicación de la víctima además de permitir tener noción de su estado de salud.

Sin embargo, ambas ideas tienen obstáculos que superar, además de requerir determinadas condiciones para funcionar correctamente, entre ellas la correcta transmisión de las ondas de radio a través de los escombros, direccionar los detectores hacia los objetivos, poder introducir el sistema de detección dentro de los escombros, proteger los equipos de polvo, humedad, entre otros.

El sistema de seguridad inteligente es una buena medida de prevención al ser parte de las edificaciones. Su implementación requeriría de una infraestructura y servicios básicos, que pueden ser parte de una casa inteligente, logrando incluso incorporar el monitoreo de signos vitales de las personas que se encuentren dentro del lugar. Ante un derrumbe total o parcial de la construcción, el sistema debería de ser capaz de dar indicaciones para que la gente se resguarde en determinados lugares e informar las posiciones al personal de búsqueda y rescate.

Esta propuesta está basada en los protocolos de respuesta ante emergencias llevados a cabo en grandes edificios, combinada con uno de los enfoques del concepto de *Smardust* donde computadoras pequeñas y sensores se integrarían a los materiales de construcción para que de alguna forma puedan ayudar a encontrar a las víctimas no superficiales, considerando que es una tecnología en desarrollo, el enfoque de trabajo tendría que ser orientado al crear estos microdispositivos o en implementar un sistema de domótica con la tecnología actual.

A pesar de considerar diferentes aspectos y parámetros para el diseño de la construcción, se requiere de grandes inversiones para implementar en dichas edificaciones sistemas de seguridad eficaces contra la actividad sísmica, como es el caso de los rascacielos en Tokio. Sin embargo, independientemente de las consideraciones que se realicen, las causas de un derrumbe pueden ser tan diversas, como lo sucedido con el World Trade Center en Nueva York, por lo que se puede concluir que a pesar de implementar sofisticados sistemas de prevención, hasta ahora no se puede garantizar que dejen de haber víctimas no superficiales en un derrumbe, incluso en el edificio más moderno, puesto que todo lo que pueda suceder, pasará.

Razones que han dado lugar al desarrollo de tecnología orientada a dar respuesta a situaciones de emergencia, siendo de especial interés la detección de personas atrapadas bajo escombros. Basados en la información consultada, sabemos que un solo método de detección

no es suficiente para definir la ubicación de una persona atrapada bajo escombros, por lo que se propone integrar un sistema que procese la información de dispositivos y sensores que permitan detectar personas sin la intervención del criterio de un operador mediante implementando el procesamiento de todas las fuentes de información, con la capacidad de buscar directamente debajo de los escombros, a diferencia del concepto del casco para rescatistas propuesto por Lee [15].

El análisis anterior obtuvo como resultado proponer la solución que se describe a continuación, para implementarla, ponerla en operación y analizar los resultados obtenidos para dimensionar su alcance, acorde a los objetivos planteados.

2.3.2 Definición de la solución propuesta

Diseñar, construir e implementar un sistema de procesamiento de señales que permita aplicar algoritmos para la detección de víctimas a través de diversos dispositivos que proporcionen señales útiles para el propósito, mediante procesamiento digital de señales, abordando en este trabajo el análisis de señales de video provenientes de una cámara térmica, auxiliada de video convencional, orientado a la detección del pulso cardíaco.

El concepto del equipo abordará las recomendaciones de los rescatistas, procurando hacerlo portable, compacto con el objetivo de incorporar este sistema a robots móviles (terrestres o aéreos) capaces de explorar sobre, entre y debajo de los escombros de un derrumbe para buscar signos vitales, accediendo a zonas peligrosas o inalcanzables para un animal o ser humano.

Como antecedente directo de este proyecto, compañeros del Taller de Robótica Abierta, de la Facultad de Ingeniería (FI) presentaron en la RoboCup del 2014 un robot tipo rover de búsqueda en entornos de desastre, en la competencia de Rescue Major. El prototipo detectaba las víctimas simuladas, empleando un termógrafo, un sensor de CO_2 y algoritmos de procesamiento de video para detectar el movimiento.

Si un solo robot todo terreno es capaz de identificar víctimas, un equipo de varios robots de tipo enjambre (trabajo colaborativo) con el sistema propuesto, podría reducir el tiempo de inspección e incrementar la cantidad de personas rescatadas, resultados al dividir esta tarea.

2.3.3 Justificación de la solución

La decisión de la solución planteada anteriormente es producto de los siguientes argumentos y razones:

1) Los conocimientos en materia de procesamiento de imágenes y video, así como del apoyo de maestros y doctores especialistas en procesamiento de señales en la facultad, permitirían

llevar a la última instancia el desarrollo de este proyecto, llevándolo hasta las pruebas de campo.

2) La tecnología que se podría emplear, esta a nuestro alcance gracias al apoyo del proyecto IT102615 "Robots no convencionales para tareas de exploración y búsqueda" para su financiamiento.

3) De forma paralela al desarrollo de este trabajo, se están desarrollando un conjunto de robots de enjambre diseñados para poder explorar entre escombros y comunicarse entre ellos, dichos robots podrían portar el sistema propuesto y llevarlo a detectar víctimas no superficiales.

4) Los resultados obtenidos por investigadores en la detección de signos vitales, como el pulso cardíaco mediante termografía, detección de movimiento del pecho y de la emisión de CO_2 tienen un mayor soporte comparado los obtenidos aplicando ondas de radio a través de los escombros, hasta el momento de la redacción. Por lo que se podrían sentar las bases para poder realizar el procesamiento necesario para la detección de este signo vital.

5) Acorde a la experiencia del equipo de entrenadores de perros de búsqueda de la UNAM, las fuentes de calor son factibles de identificar debajo de los escombros, debido a que la temperatura que se llega a presentar es templada, a pesar de las condiciones que se tengan en la superficie de las pilas de escombros.

2.4 Resumen

En este capítulo se revisaron y registraron las principales especificaciones del problema, documentando los diferentes comportamientos de las estructuras después de un derrumbe, los espacios confinados que se llegan a formar, (triángulos de vida), donde las víctimas pueden resguardarse y tener mayores probabilidades de supervivencia así como los probables estados de salud y condiciones en las que se podrían encontrar las personas.

Se realizó el análisis de los antecedentes documentados, señalando las condiciones que inhiben el funcionamiento y comprometen el desempeño de los actuales equipos de detección, así como las líneas de investigación actuales que se ocupan de la detección de víctimas no superficiales. Considerando las recomendaciones consultadas en manuales y protocolos de acción ante este tipo de emergencias así como de la entrevista realizada a los entrenadores de perros de búsqueda de nuestra casa de estudios, se expusieron tres opciones de solución al problema planteado, las cuales se analizaron concluyendo con la formulación de la solución propuesta que definió la línea de trabajo de este proyecto, presentando la razones que definieron trabajar con procesamiento de imágenes térmicas.

Capítulo 3

Diseño del sistema

Con la solución propuesta para este trabajo, definida al término del capítulo anterior, se desarrollará el diseño del sistema, estableciendo los parámetros y especificaciones del proyecto acorde a los objetivos establecidos, indicados en el capítulo 1.

Para elegir los dispositivos que integrarán el hardware de detección, se documentó y analizó cada uno de ellos, presentando la metodología y el procedimiento que se siguió. En el caso de la cámara térmica se incluye una breve explicación de su principio físico de funcionamiento así como de los cálculos necesarios para identificar la sensibilidad del detector que permita identificar las variaciones de temperatura debidas al flujo sanguíneo de una persona, cerrando este análisis con 3 equipos propuestos para utilizar.

Posteriormente se continúa con la selección de una cámara de video, considerando que los equipos funcionarán en cualquier sistema operativo, soportado por los sistemas embebidos propuestos para controlar y extraer la información de las cámaras.

Finalizados los procesos de preselección, se presentan las propuestas de diseño conceptual considerando los dispositivos mencionados anteriormente, proponiendo configuraciones tanto de implementación como de funcionamiento del sistema, así como las formas de integrar todos los componentes.

Acorde a los conceptos expuestos, se explican las razones por las cuáles se seleccionan específicamente los dispositivos que se ocupan para la construcción y puesta en operación del sistema, comprendiendo su implementación.

3.1. Especificaciones del proyecto

Para comenzar a diseñar, es necesario establecer los criterios y especificaciones basados en los objetivos que debe tratar satisfacer la solución propuesta, la cual contempla trabajar con el sistema de detección involucrando el procesamiento de video con el fin de integrarlo a robots móviles. Por ello, los criterios de diseño más importantes de forma general se muestran a continuación:

I. Todos los elementos de hardware de sistema de detección deberán de tener una autonomía energética de mediano plazo, es decir, que sean capaces de funcionar utilizando baterías recargables.

II. Las dimensiones de los dispositivos deberán de procurar ser lo más pequeñas posibles, sin comprometer características necesarias para la detección.

III. Prescindir en medida de lo posible, de intermediarios en la adquisición de la información de las cámaras para evitar la pérdida de información, procurando que los datos sean en algún formato digital.

IV. Implementar un sistema que permita el intercambio de información de los dispositivos para dividir tareas de control o de procesamiento, además de permitir incorporar otros elementos de detección.

Con estos criterios, a continuación se describe el diseño conceptual del sistema de detección para la solución propuesta.

3.2 Diseño conceptual

Independientemente de los elementos que integren el sistema de detección, se podría trabajar teniendo todos los elementos concentrados en un solo módulo o dividir los diferentes dispositivos para adquirir las señales a procesar, la elección de la configuración estará en función de las condiciones presentes en la zona del derrumbe. Ambas propuestas pueden ser trabajadas a partir de un módulo de pruebas para implementar la adquisición de datos de las cámaras, la transmisión de video, implementar los algoritmos enfocados a la detección de víctimas no superficiales.

A su vez, en cualquiera de los dos casos de implementación planteados, se considerará la integración del sistema a robots de búsqueda individuales y colaborativos, para llevar las

cámaras y demás dispositivos que podrían incorporarse para obtener mejores resultados en la detección de víctimas. A continuación se plantean y definen ambas configuraciones propuestas, describiendolas por medio diagramas de bloques para presentar los diseños conceptuales sobre los que se trabajará hasta llegar a su integración.

3.2.1 Sistema base de detección

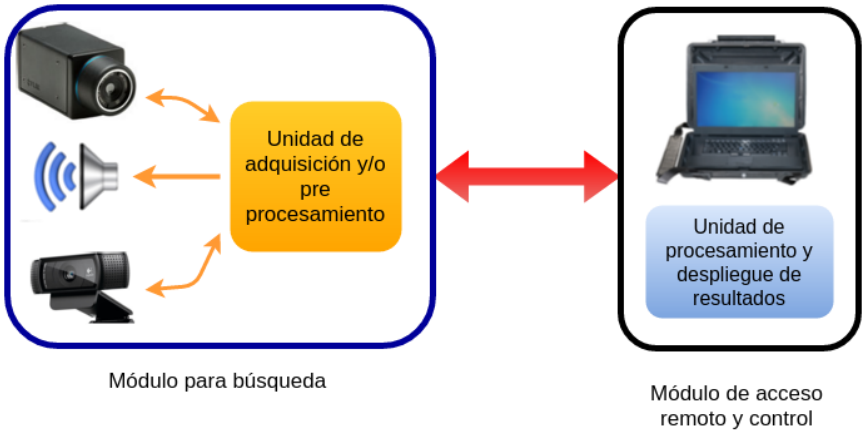


Figura 3.1 Estructura general del sistema de detección

Como se puede observar en la figura 3.1 el sistema base consiste en dos partes debido al desconocimiento de la cantidad de recursos necesarios para implementar los algoritmos de detección y para atender la parte de diseño compacto. En primer instancia, se tiene el *módulo de búsqueda*, encargado de adquirir las señales de ambas fuentes de video y de otros dispositivos útiles para la detección de víctimas, y transmitir dicha información para ser procesada por el *módulo de control,* que permitirá monitorear las variables recibidas, analizarlas y procesarlas para desplegar los resultados de la búsqueda.

El hardware que se comunicará con las cámaras, también podría manejar otros dispositivos del módulo de búsqueda e incluso hasta del robot móvil que lo transporte, siempre y cuando tenga la capacidad suficiente de procesamiento y características para dar soporte a estos elementos, consideración que será analizada posteriormente en este mismo capítulo. En la figura 3.2 podemos observar cómo se podría implementar el sistema base tanto para realizar pruebas en ambientes controlados como en robots de exploración.

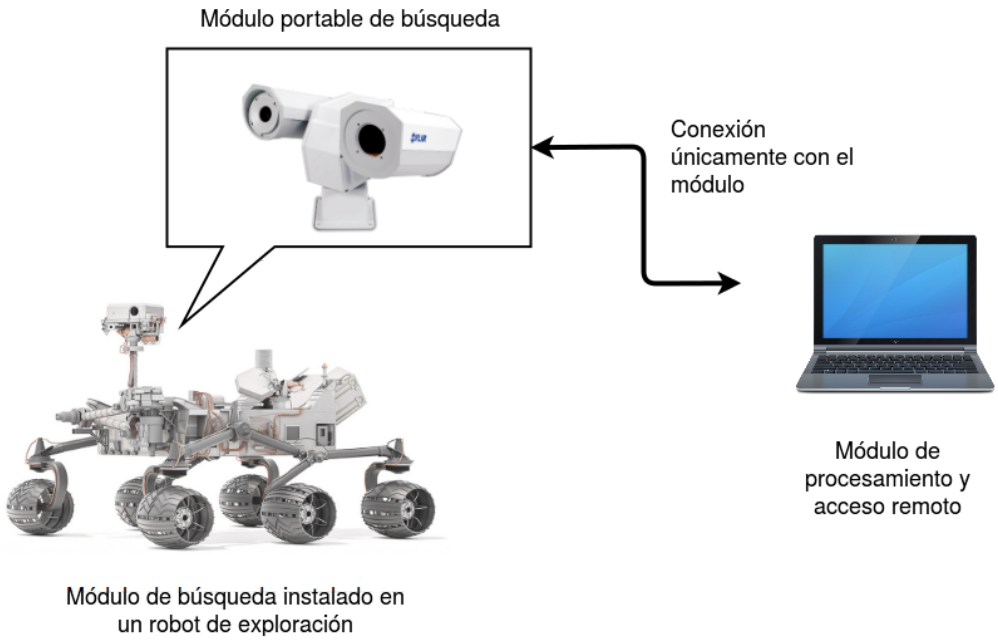


Figura 3.2 Conceptos de implementación del sistema base

3.2.2 Sistema de detección de Múltiples Entradas y Múltiples Salidas (MIMO)

Una vez que el sistema base de detección sea puesto en operación, ya sea únicamente con las cámaras o agregando más dispositivos para la detección de víctimas no superficiales, para agilizar la búsqueda se podría expandir el sistema bajo una arquitectura MIMO, en un conjunto de robots enjambre de exploración, en vehículos aéreos no tripulados (UAV), entre otros, que envíen toda la información a un servidor encargado de realizar diferentes tipos de procesamiento para la detección de víctimas no superficiales, y pueda desplegar la transmisión de video así como los resultados de los algoritmos aplicados en diferentes salidas. La Figura 3.3 describe la arquitectura del sistema de detección MIMO.

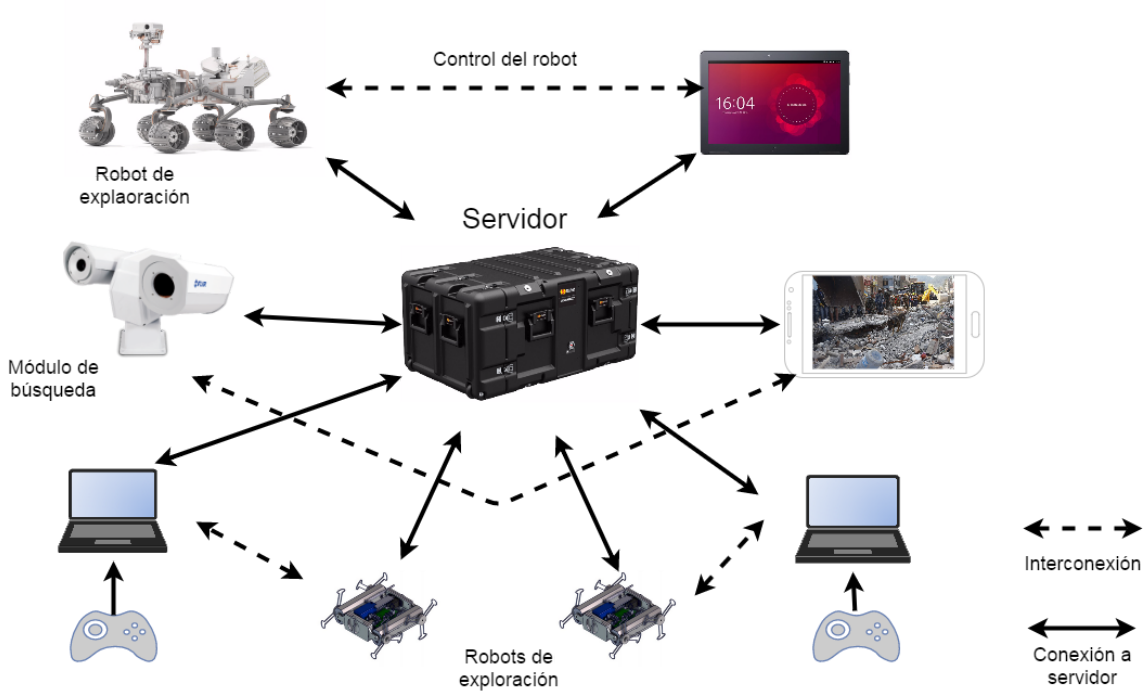


Figura 3.3 Concepto del sistema de búsqueda de víctimas no superficiales MIMO

Cada uno de los módulos de búsqueda podría llevar una combinación diferente de los sensores y dispositivos útiles para detectar signos vitales, con el fin de tener múltiples variables que permitan tener resultados más certeros comparado con analizar sólo una fuente de información. Al igual que el sistema base, solo con un punto de acceso se podrían desplegar los resultados de todas las entradas de datos o incorporar la conectividad con teléfonos inteligentes o tabletas electrónicas para dividir el despliegue de toda esta información, permitiendo que más de una persona pueda estar buscando.

Ambos modos de funcionamiento se abordan en este trabajo, implementado el sistema de detección base bajo una estructura de software y hardware que permita incorporarlo a robots, establezca la base de la arquitectura MIMO, bajo la línea de procesamiento de imágenes térmicas.

A continuación se presenta con detalle cada una de los equipos que se utilizarán para el sistema, mostrando en algunos casos su principio de funcionamiento, el estudio de mercado realizado para la selección de cada uno y su proceso de selección para su implementación.

3.3. Cámara termográfica

Las imágenes termográficas son representaciones gráficas en el espectro visible, de la radiación infrarroja en determinados intervalos de longitudes de onda, producto del sistema electrónico de las cámaras que procesan la información recibida del medio por detectores sensibles a la IR. Una de las aplicaciones más comunes de las cámaras termográficas (también denominadas térmicas) es monitorear y estimar la temperatura de algún cuerpo u objeto sin contacto alguno, siempre y cuando esté dentro del campo de visión el objetivo de interés [8] y [9].

A diferencia de las cámaras de video convencional, por su principio de funcionamiento son inmunes a los cambios de intensidad de luz visible, logrando proporcionando imagen o video incluso en completa oscuridad, característica que las hace ampliamente utilizadas en sistemas de vigilancia nocturna.

El funcionamiento de las cámaras térmicas, de forma general se consiste de la siguiente forma; la lente enfoca la radiación IR emitida y reflejada de todos los objetos dentro de su campo de visión, incluyendo la atmosférica debida al sol, hacia el detector que es un arreglo matricial focal (FPA) de determinado orden equivalente al tamaño de la imagen, cuyos elementos son celdas del tamaño de micrómetros fabricadas de diferentes materiales acorde a la sensibilidad para la que sea fabricada, que constituyen los píxeles. La información recibida se digitaliza y es procesada por un sistema electrónico para construir las imágenes o video según sea el caso, en formatos que permitan al usuario visualizar la información a través de interfaces de software. En la Figura 3.4 se muestra una diagrama de bloques de los principales componentes de una cámara termográfica.

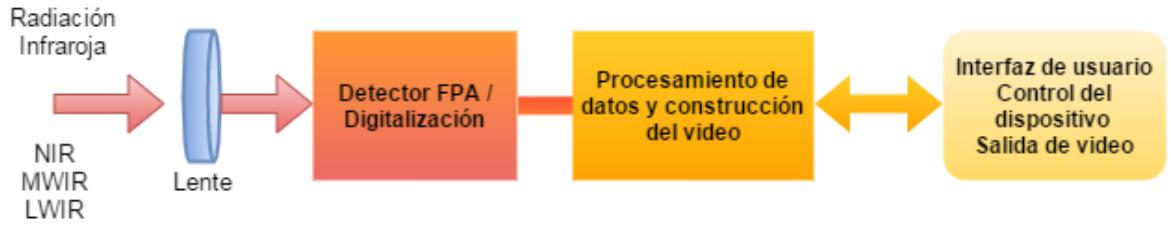


Figura 3.4 Diagrama de bloques básico de una cámara térmica

Los equipos comerciales son sensibles a tres intervalos de la banda del espectro electromagnético que corresponde a la radiación infrarroja, los cuales son; longitud de onda corta (NIR) que comprenden de 0.7 a 2.5 μm , longitud de onda media (MWIR) abarcando de 2 a 5.6 μm y longitud de onda larga (LWIR) en la banda de 8 a 14 μm. En la figura 3.6 se muestran las bandas de detección de longitud de onda de diferentes tipos de detectores.

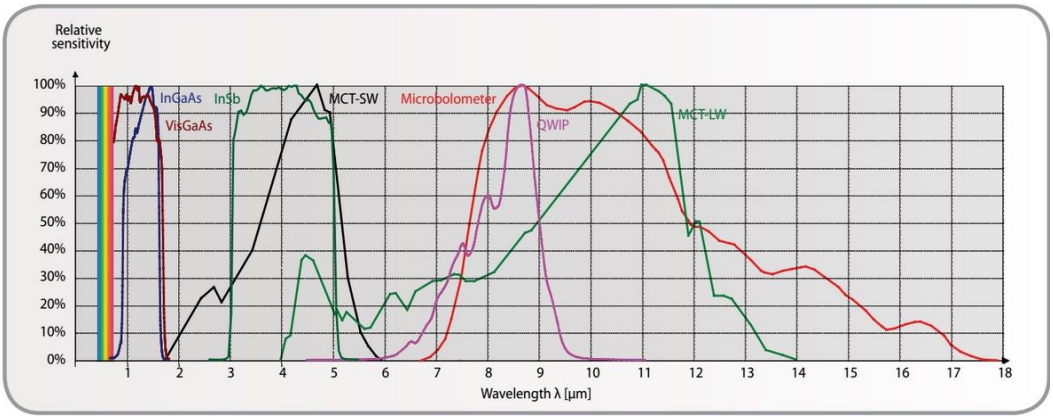


Figura 3.5 Curva de sensibilidad relativa de los diferentes tipos de detectores de las cámaras térmicas

3.3.1 Principio físico de funcionamiento

Todos los objetos desde su superficie, emiten cierta cantidad de energía la cual varía acorde a la temperatura, la longitud de onda de la radiación que incide sobre él y de sus propiedades. Si el objeto es más frío que 500°C la energía emitida coincide con el intervalo de longitudes de onda correspondiente a cierta sección de la banda del infrarroja del espectro electromagnético (también conocida como banda de infrarrojo térmico).

La radiación infrarroja fue descubierta en 1800 por el astrónomo William Herschel al realizar un experimento, donde observó el comportamiento térmico de la banda visible del espectro electromagnético. La implementación consistió en incidir un haz de luz blanca (del sol) en un prisma, para descomponerla y proyectarla en una mesa para realizar mediciones de temperatura en cada una de las franjas de color. Al hacerlo descubrió que la temperatura variaba acorde a la banda en la que posicionará el termómetro, incrementando del color violeta al rojo. Al terminar de registrar la banda visible, se le ocurrió medir más allá del color rojo y observó que la temperatura continuaba en aumento, hasta que logró registrar una lectura estable, por encima de la banda roja [9].

Al difundir los resultados del experimento, comenzó el reto de explicar este hecho, tratando de modelar su comportamiento, una de las expresiones matemáticas relevantes que describieron parte del experimento de Herschel, fue la ley de Stefan-Boltzman que describe la rapidez a la que un objeto radia energía es proporcional a la cuarta potencia de su temperatura absoluta, expresada en forma de ecuación como

$$W = \varepsilon\sigma T^4 \left(\frac{W}{m^2}\right) \qquad (3.1)$$

Donde W es la potencia en watts de las ondas electromagnéticas radiadas de la superficie del objeto, σ es la constante de Stefan-Boltzmann cuyo valor es $5.669 \times 10^{-8}(W/m^2K^4)$, ε es el

coeficiente de emisividad y T es la temperatura superficial en grados Kelvin. La emisividad es la proporción de la radiación térmica emitida por una superficie u objeto debido a su temperatura, término vinculado con la absortividad, que es la fracción de la radiación incidente que absorbe la superficie. Una superficie negra tiene alta absortividad y alta emisividad, observación hecha por el físico Gustav Kirchhoff en 1862, al introducir el concepto teórico del cuerpo negro, que pretendía describir teóricamente la radiación térmica.

Las propiedades radiativas de los objetos son usualmente descritas en relación con el concepto ideal del cuerpo negro (el perfecto emisor). Si la potencia radiada por el cuerpo negro es denotada por W_{bb} y para un objeto normal a la misma temperatura como W_{obj}, la relación entre los dos valores describen el valor de la emisividad ε de un objeto, como

$$\varepsilon = W_{obj} / W_{bb} \qquad (3.2)$$

ε toma valores en el intervalo de 0 a 1, donde el cuerpo negro tiene un coeficiente $\varepsilon = 1$ [16]. Las mejores propiedades radiativas de un objeto se tienen cuando la emisividad es cercana a la unidad. Los objetos que tienen la misma emisividad para todas las longitudes de onda, son llamados cuerpos grises, cuya potencia de radiación es expresada por la ecuación 1.

Sin embargo, al continuar aplicando la teoría clásica no se logró describir el comportamiento de la radiación térmica, pero a partir de las observaciones y registros de múltiples experimentos se establecieron las bases para hacerlo, dando por resultado la ley de Stefan-Boltzman, la ley del desplazamiento de Wien que describe el comportamiento de los puntos máximos de radiación en función de la distribución de la longitud de onda y el modelo de Rayleigh-Jeans.

En el año 1900 Max Planck, desarrolló una teoría que describe la distribución de la energía de un cuerpo negro, en dicho trabajo consideró las bases mencionadas e introdujo los primeros conceptos de física cuántica. La ley de Planck, resultado de su trabajo, describe el espectro de radiación térmica en función de la temperatura de la superficie del objeto y de a longitud de onda, expresada de la siguiente forma:

$$P_\lambda = \frac{2\pi hc^2}{\lambda^5\left[e^{\frac{hc}{\lambda kT}} - 1\right]} \qquad (3.3)$$

donde P_λ es la potencia emitida (W/m^2) a una particular longitud de onda $\lambda\,(m)$, h es la constante de Planck $6.626 \times 10^{-34}(Js)$, *c* es la velocidad de la luz $3 \times 10^8\ (m/s)$ *K* es la constante de Boltzmann $1.38 \times 10^{-23}(J/K)$ y *T* es la temperatura en grados Kelvin (K) [17]. En la figura 3.6 se muestra el comportamiento de esta ecuación seleccionando diferentes temperaturas.

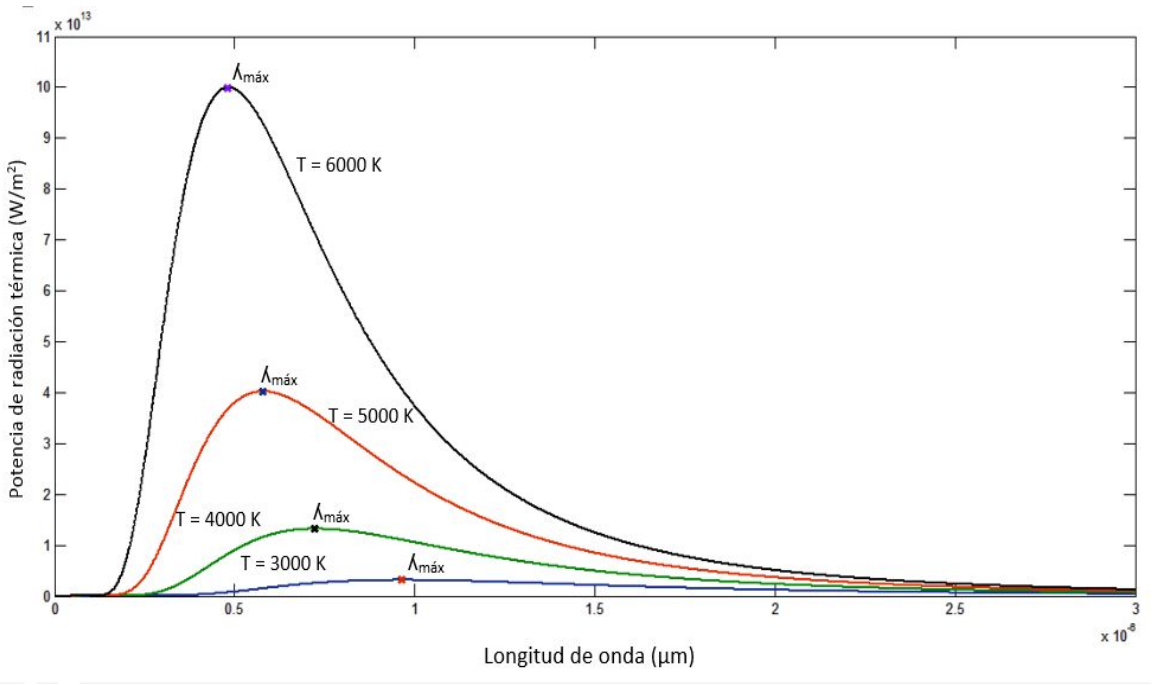


Figura 3.6 Espectro de radiación térmica del cuerpo negro descrito analíticamente por ley de Planck.

La radiación emitida (área bajo la curva) aumenta al incrementarse la temperatura. El máximo punto de la gráfica se ubica en diferentes longitudes de onda acorde a la temperatura y se desplaza hacia intervalos de longitudes de onda menores al incrementar T. Este comportamiento describe una constante, definida por la ley del desplazamiento de Wien como:

$$\lambda_{max} T = 2.898 \times 10^{-3} [mK] \qquad (3.4)$$

donde T es la temperatura absoluta del cuerpo negro medido en grados Kelvin y λ_{max} la longitud de onda donde se presenta la máxima intensidad.

Integrando el espectro de radiación emitido para todas las longitudes de onda de 0 a infinito, se puede obtener el valor de la potencia total de la radiación emitida del cuerpo, a determinada temperatura, por medio la ley de Stefan-Boltzmann (ecuación 3.1).

Basado en la teoría de la radiación térmica infrarroja, la primera cámara termográfica se construyó en 1929 por el físico húngaro Kalman Tihanyi, para el sistema de defensa antiaérea de Gran Bretaña [9]. La potencia de radiación por unidad de área depende de la temperatura absoluta del cuerpo y de su coeficiente de emisividad, sin importar la forma de la superficie, conclusión que nos permite calcular los datos necesarios para elegir el intervalo de longitudes en donde la radiación del cuerpo humano es más intensa, acotando el tipo de detector que se ajusta a los objetivos de este trabajo.

3.3.2 Medición de la temperatura por radiación térmica

La cámara térmica recibe radiación la radiación infrarroja del objeto de interés, sino también de aquella reflejada y absorbida por los diferentes objetos que lo rodean así como de la emitida por el sol. Sin embargo, todas las fuentes de energía diferentes a nuestro objetivo se atenúan en determinada proporción por la atmósfera. La figura 3.7 ilustra de forma representativa la energía que capta la cámara.

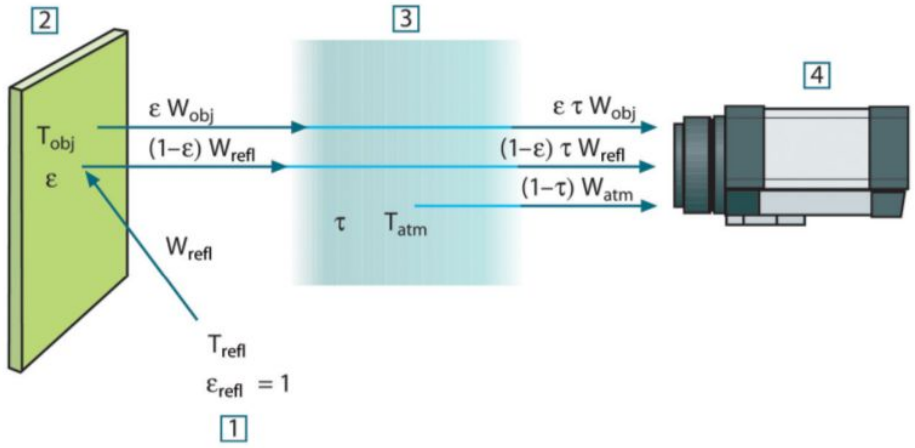


Figura 3.7 Representación gráfica de la medición general de la radiación infrarroja medida con una cámara térmica. [1] contribuciones de los objetos que rodean al objeto de interés, [2] objeto de interés, [3] atmósfera y [4] cámara

La radiación total aproximada que incide en la cámara puede ser calculada con la ecuación 3.5:

$$W_{tot} = \varepsilon\tau W_{obj} + (1-\varepsilon)\,\tau W_{refl} + (1-\tau)\,W_{atm} \qquad (3.5)$$

Donde la potencia de radiación emitida por el objeto de interés es igual a $\varepsilon\tau W_{obj}$, es la emisividad del objeto y τ es la transmisividad de la atmósfera. La emisión reflejada por las diferentes fuentes del medio es igual a $(1-\varepsilon)\tau W_{refl}$ donde $(1-\varepsilon)$ es la reflectancia del objeto asumiendo que la temperatura ambiente es la misma para todas las superficies emisoras, además de considerar que la emisividad de los objetos del entorno es igual a 1, lo cual es correcto acorde a la ley de Kirchhoff; toda la radiación que incide sobre las superficies circundantes, con el paso del tiempo será absorbida por las mismas superficies. Finalmente la radiación que contribuye la atmósfera es igual a $(1-\tau)W_{atm}$, donde $(1-\tau)$ es la emisividad de la atmósfera [8].

Los equipos comerciales, adaptan la ecuación 3.5 para realizar las mediciones estimadas de la temperatura, acorde a las características de su sistema, sin embargo, el operador necesita ingresar cierta información al software de la cámara además de ingresar algunos parámetros

para que pueda calibrar y poder realizar la medición aproximada. Concluyendo que la cámara térmica detecta gradientes de calor, debidos a la radiación infrarroja que sea captada por el FPA y en función de ello es posible estimar la temperatura.

Cálculo de la banda infrarroja para la detección de personas

Utilizando las ecuaciones descritas sobre la radiación térmica del cuerpo negro se puede definir las características más importantes que contribuyan a elegir la cámara más adecuada para el proyecto.

La piel humana es muy parecida a un cuerpo negro, su emisividad promedio es de 0.98 sin importar la raza o color de piel [6]. Con la ley de la radiación de Planck (ecuación 3.3) y la temperatura promedio del cuerpo humano, podemos obtener el intervalo de longitudes de onda donde la intensidad de radiación térmica es mayor. La figura 3.8 muestra 3 curvas de Planck a diferentes temperaturas, incluyendo la del ser humano.

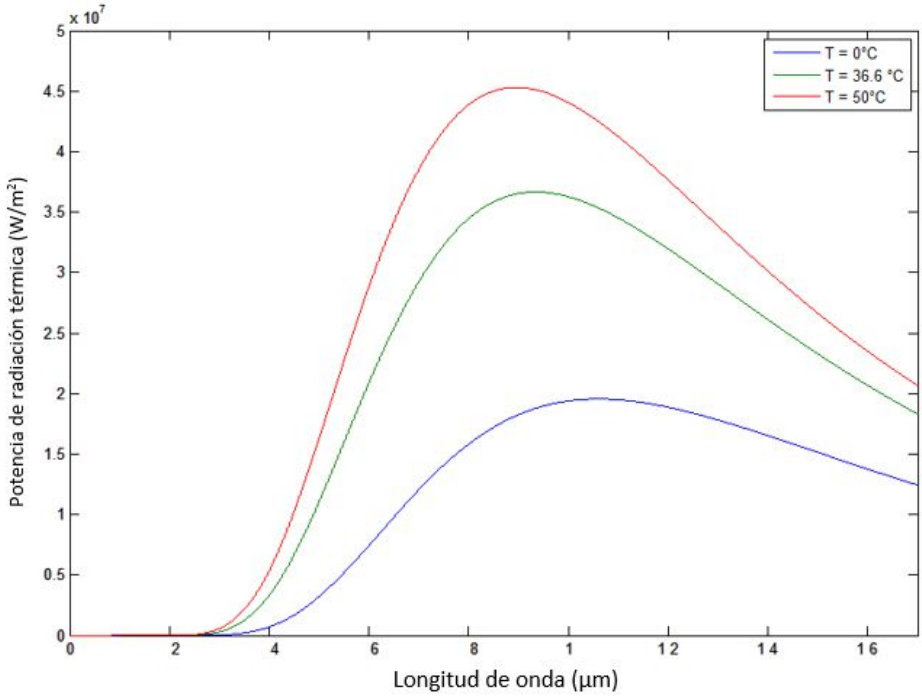


Figura 3.8 Espectro de radiación de tres temperaturas diferentes, incluida la del cuerpo humano.

Podemos observar de la gráfica, en la curva correspondiente a la temperatura promedio de una persona, que la distribución de energía emitida es mayor dentro del intervalo de 7 a 12 µm de longitud de onda, banda que se cubre para las cámaras que detectan longitudes de onda largas LWIR. Aplicando la ecuación 4, obtenemos la abscisa donde se ubica el pico de esta curva:

$$T = 36\,^{\circ}C = 309,15\,^{\circ}K$$

$$\lambda_{max} = 2898\ [\mu m/K] / 309,15\ [K] = 9,37\ \mu m$$

En caso de requerir ajustes la cámara, la potencia radiada es obtenida por la ley de Stefan-Boltzmann (ecuación 3.1)

$$W_{hb} = \varepsilon\sigma T^4\ [W/m^2] = 0.98 \times 5.67 \times 10^{-8}[W/m^2K^4] \times (309,15\ K)^4$$

$$W_{hb} = 507.56\ [W/m^2]$$

Las cámaras LWIR no requieren de muchos ajustes para poder ser utilizada en cualquier lugar, puesto que en su calibración se involucra la atenuación atmosférica, ya que ésta se comporta como un filtro paso altas, teniendo como frecuencia central 7,5 μm. Estos cálculos se adicionan a los criterios para la elección de la cámara termográfica más adecuada para el proyecto.

3.3.3 Selección de la cámara térmica

Para iniciar la búsqueda de equipos que puedan ser útiles para el proyecto, se establecieron criterios para investigar toda la información referente sobre dichos dispositivos. Los criterios de búsqueda considerados fueron los siguientes:

I. Sensibilidad térmica mínima de 80 mK (necesaria acorde a los resultados obtenidos por Chekmenev [5] y Togawa [6]).

II. Salida de video digital, permitiendo la captura de imágenes y grabación.

III. Soporte de protocolos para la transmisión de video mediante una red de comunicación.

IV. Dimensiones físicas.

V. Independencia de la utilización de un solo software para acceder a la información.

VI. Disponibilidad y soporte de la cámara.

VII. Bajo consumo de energía.

Se realizó una documentación sobre los equipos disponibles en el mercado que se ajustarán a los requerimientos mencionados así como a los datos obtenidos por los cálculos realizados en la sección anterior, obteniendo como resultado la preselección de 3 equipos a considerar para tomar la decisión final, cuyas características se visualizan en la tabla 3.1.

Característica	Cámara térmica		
	Equipo A FLIR A35	Equipo B optris pi 400	Equipo C Xenics Gobi 384 Gige
Máximo tamaño de imagen (píxeles)	336 x 256	382 x 288	384 x 288
FPS [Hz] (modelos disponibles)	60 / 30 / 7,5	80	84 / 9
Resolución térmica [mK]	50	80	60
Protocolo de comunicación y transmisión de información (video e imágenes)	Giga Ethernet / GeniCam / GigE Vision	USB 2.0	Giga Ethernet / GigE Vision / CameraLink
Resolución de píxel	8 o 14 bits (configurable)	Información no proporcionada por el fabricante	16
Rango espectral [um]	7,5 - 13	7,5 - 13	8 - 14
Alimentación eléctrica	PoE clase 0 tipo B 56 V / Conector M12 12/24 V 2.5 W	5 V vía USB	12 V / 4.5 W vía conector M4
Peso [g]	200	320	263
Tamaño [cm]	10,6 x 4 x 4,3	4,6 x 5,6 x 9	7 x 7,4 x 6,5
Volumen [cm cúbicos]	182,32	231,184	336,7

Tabla 3.1 Características principales de las cámaras térmicas preseleccionadas.

La primer diferencia que resalta en la Tabla 3.1, es la resolución térmica, es decir la diferencia mínima de temperatura que puede ser detectada, siendo superior el equipo A, además de sobresalir también en los campos de volumen y peso. Sin embargo, el equipo B requiere de una menor cantidad de diferencia de potencial para funcionar, además de poder suministrar dicho voltaje por medio de un conector USB tipo A y de transmitir la información por el estándar USB 2.0. Pero el equipo C cuenta con una mayor definición de pixeles, ya que maneja un convertidor analógico digital (ADC) de 16 bits aunque esto no incide en la resolución térmica.

Cada uno de estos equipos cuenta con un software de fábrica, algunos de ellos con limitados recursos y herramientas, además, la mayor parte de la documentación de las cámaras está orientada solo a para estos ambientes, que solo llegan a ser compatibles con ciertos sistemas operativos. Los equipos A y C pueden manejar la información capturada por medio del estándar GigE Vision (GEV), el cual cuenta con un kit de desarrollo de software (SDK por sus siglas en inglés) desarrollada y soportada por *Pleora Technolgies Inc* que permite el programar aplicaciones para el control y streaming de estas cámaras, independientemente del software proporcionado por los fabricantes.

Considerando que el equipo B solo puede funcionar bajo su software de fábrica, es descartado, porque se desea procesar el video en tiempo real sin depender de un software específico, convirtiendo a los equipos A y C en finalistas, además, al trabajar con cámaras GigE Vision se obtiene la libertad de trabajar con cualquier modelo de cámara térmica dentro de dicho estándar.

3.4. Cámara de video convencional

Durante la búsqueda de los diferentes modelos de cámaras termográficas, al revisar los modelos de la empresa FLIR, se observó que algunos equipos contaban también con cámaras de vídeo convencional de baja definición, con una tecnología que les permite ver ambas fuentes por separado o combinadas, gracias a una configuración similar a las cámaras estéreo. Esta herramienta podría compensar algunas de las desventajas implícitas de los equipos térmicos, como no contar con un autoenfoque o zoom.

Esta idea incorporaría al sistema otro elemento de inspección, el cual podría llegarse a combinar con la cámara termográfica para lograr mejores resultados de detección. Actualmente las cámaras web son utilizadas en muchos proyectos de visión artificial, porque además de ser compactas tienen un gran soporte en diferentes sistemas operativos e interfaces, por esta razón, la búsqueda de un equipo de video convencional se restringe a este tipo de dispositivos. Los principales aspectos que se consideraron para hacer una documentación al respecto, fueron los siguientes:

I. Salida de video digital, con alta resolución tanto de imagen como de video y tamaños de imagen configurables.

II. Ser un dispositivo de video de clase USB (UVC por sus siglas en inglés) por el amplio soporte que tienen este grupo de cámaras.

III. Formatos de imágen y video convencionales.

IV. Contar con micrófono para realizar extracción de audio.

V. Tener autoenfoque y preferentemente zoom digital

VI. Ser ligero y ocupar el menor espacio posible

Aplicando el mismo procedimiento que se realizó con la cámara térmica, se obtuvo como resultado la Tabla 3.2 que contiene las características de 3 cámaras web para contemplarse para construir e implementar el sistema.

Característica	Cámara Web		
	Equipo A Microsoft LifeCAM HD 3000	Equipo B Logitech HD Pro C920	Equipo C Logitech HD C525
Máximo tamaño de imagen	1280 x 720	1920 x 1080	1280 x 720
FPS [Hz]	30	30 a 1080p	30 a 480p
Máxima calidad de video (Megapíxels)	0,3	2,1	2,1
Máxima calidad de imagen (Megapíxels)	1	15	8
Tipo de enfoque	Automático	Automático	Automático
Zoom digital	4x	4x	0x
Tamaño [cm]	3,8 x 4,1 x 10,9	9,4 x 7,1 x 4,3	15,2 x 7,6 x 4
Peso [g]	96	162	88,00
Extras	Control automático de brillo, adquisición de audio mono con reducción de reuido	Balance de blancos, control automático de brillo, compresión de video H.264, adquisición de audio estéreo con reducción automática de ruido	Control automático de brillo, adquisición de audio mono con reducción de ruido

Tabla 3.2 Características principales de los modelos de cámaras web preseleccionadas

Para seleccionar estos tres modelos se consultaron varios foros y sitios de internet, a partir de los cuáles se comenzó a revisar las características de un determinado grupo. Cualquiera de los tres modelos presentados se podría implementar en el sistema, aportando funciones propias de cada modelo. Sin embargo se tiene como referencia, el modelo B puesto que en numerables foros y análisis comparativos, la posicionan como la mejor webcam hasta el momento, además de ser citada en la implementación de proyectos multimedia y de procesamiento de imágenes.

3.5. Unidad de adquisición de datos y/o procesamiento

Actualmente existen muchas plataformas que permiten desarrollar diferentes proyectos gracias a sus diferentes unidades de control, microcontroladores, microprocesadores controladores y módulos que les brindan características específicas a los sistemas embebidos.

La solución propuesta, requiere como mínimo, poder extraer la información de las diferentes fuentes disponibles en el sistema y enviarla a una unidad de proceso para que pueda ser analizada y monitoreada, como se propone en el diseño conceptual de la figura 3.1. Los criterios considerados para la búsqueda de sistemas embebidos, son los siguientes:

I. Soporte para los formatos de video y de los controladores de ambas cámaras.

II. Capacidad de procesamiento suficiente para la adquisición de datos de manera simultáneamente.

III. Contar con acceso remoto tanto de forma alámbrica como inalámbrica.

IV. Versatilidad para incorporar nuevas fuentes de información.

V. Procurar tener dimensiones compactas sin sacrificar la capacidad de procesamiento ni los recursos de hardware necesarios para el manejo de las cámaras.

Continuando con el proceso de investigación aplicado con las cámaras, se buscaron los sistemas disponibles en el mercado que cumplieran con la mayor parte de los criterios de diseño, dando como resultado la Tabla 3.3 con los candidatos y sus características principales.

Uno de los factores comunes de todos los candidatos, es su capacidad de trabajar con diferentes distribuciones de Linux, abriendo un abanico de posibilidades para el desarrollo del sistema de detección. Y dos principales diferencias relevantes para el proyecto, son el control de puertos giga ethernet y el microcontrolador o microprocesador (según sea el caso) que dirige utilizado en la plataforma.

		Sistemas embebidos				Mini computadoras	
		BeagleBone Black rev C	Raspberry 2 B+	Banana Pro	Up board	Intel NUC DC3217IYE	Fitlet-i Barebone
Características básicas del sistema	Procesador	Sitara AM3358BZCZ100 1 GHz	Broadcom 2836 Quad-core ARM Cortex-A7 900 MHz	Allwinner A20* (ARM Cortex-A7 dual-core, 1GHz, Mali400MP2 GPU	Intel atom x5-Z8350 QuadCore 1.44Ghz (1.92GHz)	Intel Core i3-3217U (3M Cache, 1.80 GHz)	AMD A4 Micro-6400T SoC quadcore 1 GHz 2M Cache
	RAM	512 MB	1 Gb	1 Gb	2 Gb	4 Gb	Soporte para 8 Gb DDR3, de venta por separado
	Unidad de almacenamiento	uSD y 4 Gb eMMC	uSD	uSD y SATA 2.0	uSD y 16/32 Gb eMMC	SD y mSATA 3.0	Soporte para mSATA 3.0, eSATAp 3.0 y uSD. De venta por separado
	Sistema(s) Operativo(s)	Distribuciones personalizadas y nativas ligeras de linux	Distribuciones personalizadas de linux, Windows 10	Distribuciones personalizadas de linux	Distribuciones de Linux, Windows, Android	Linux, Windows, iOS	Windows y Linux
	No. de bits de la Arquitectura	32 bits	32 bits	32 bits	64 bits	64 bits	64 bits
Conectividad	Ethernet	1 Puerto 10/100 Mbps	1 Puerto 10/100 Mbps	1 Puerto 10/100/1000 Mbps	1 Puerto 10/100/1000 Mbps	1 Puerto 10/100/1000 Mbps	2 Puertos 10/100/1000 Mbps
	USB	1 Puerto USB 2.0	4 Puertos USB 2.0	4 Puerto USB 2.0	4 Puertos USB 2.0 y 1 Puerto USB 3.0 tipo OTAG	3 Puertos USB 2.0	4 Puertos USB 2.0 y 2 USB 3.0
	Puertos	Digitales I/O, PWM, ADC, I2C, UART, CAN	Digitales I/O, UART, I2C	Digitales I/O, UART, I2C	Digitales I/O, UART, I2C, SPI, PWM	-	-
	Wifi	Soporte de ciertos modelos de adaptadores, de máximo 300 Mbps	Soporte de ciertos modelos de adaptadores, de máximo 300 Mbps	Antena incluida con soporte para bandas b/g/n	Soporte de ciertos modelos de adaptadores, de máximo 300 Mbps	Por medio de adaptadores externos USB	WLAN 802.11ac (2.4/5GHz dual band Intel 7260HMW) NIC incluida
Otras características	Alimentación	5 V / 2 A	5 V / 2 A	5 V / 2 A	5 V / 2 A	12 V	10 - 15 V 4.5 - 10.5 W
	Tamaño [cm]	8.6 x 5.3	9.2 x 6	9.2 x 6	8.5 x 5.6	10.1 x 10.1 x 3.8	10.8 x 8.3 x 2.4

Tabla 3.3 Principales características de los sistemas embebidos preseleccionados

Los primeros dos sistemas embebidos mostrados en la Tabla 3.3, fueron considerados a pesar de no contar con una de los dos características resaltadas en el párrafo anterior, ya que se podría analizar la opción de utilizar una tarjeta de red USB - Giga Ethernet compatible en alguna de estas dos, en el caso de que se elija una cámara térmica con este puerto. Sin embargo, la tarjeta Banana Pi Pro, ofrece mejores prestaciones de hardware que la Raspberry pi 2 B+ y la BeagleBone Black rev C, pero carece del amplio soporte y documentación que poseen estas dos últimas plataformas.

Los 3 sistemas citados hasta ahora, tienen como CPU microcontroladores con arquitecturas ARM de 32 bits y los modelos posteriores cuentan con microprocesadores que aportan más recursos para la implementación. Siendo un punto medio la tarjeta Up board, con un procesador Intel Atom X5 Z8350 de 64 bits, puertos de entrada y salida de propósito general, entre otras cosas.

Además, se incluyeron en la Tabla 3.3 dos mini computadoras que permitirían procesar la información en el módulo de búsqueda y desplegar los resultados en otro medio conectado. Su tamaño y conectividad permitirían incorporar más elementos en un futuro al sistema, por ejemplo microcontroladores que realicen otras tareas, como el control mecánico del robot. Todos estos elementos serán considerados en la siguiente sub capítulo para definir que equipos utilizar.

3.6 Selección de hardware

Para definir los dispositivos que integrarán el sistema base para detectar víctimas no superficiales, realizando procesamiento de imágenes y video, se analizaron todas las características citadas en el subcapítulo anterior, para seleccionar aquellos que permitan lograr los objetivos planteados.

La compatibilidad y conexión de todos ellos serán los aspectos más importantes a considerar para decidir cuáles utilizar, además de las especificaciones de diseño citadas al inicio de este capítulo. A continuación se enlistan los dispositivos seleccionados implementar el sistema, así como las razones de su elección.

3.6.1 Cámara térmica FLIR A35

Es el modelo con mayor resolución térmica de 50 [mK], el más ligero con 200 [g], el más bajo consumo eléctrico 2.5 [W] y es la de menor volumen. Esta elección fue hecha básicamente por las propiedades mencionadas, además al transmitir el video utilizando el protocolo GEV se obtiene la libertad de poder desarrollar aplicaciones con la SDK, sin tener que utilizar necesariamente el software proporcionado por el fabricante.

El campo de visión de la cámara es de 48° (H) x 39° (V), entrega el video a través del puerto ethernet en formato MPEG-4 y pude recibir radiación de fuentes de calor en el intervalo de -40 a 160 °C y también de -40 a 550 °C.

3.6.2 Cámara web logitech HD Pro C920

Modelo con mayor definición de imágenes y video (15 y 2.1 megapíxeles, respectivamente), maneja 3 diferentes formatos, incluido el H.264, tiene un micrófono estéreo con reducción de ruido ambiental, un balance de blancos automático y auto ajuste de la intensidad de luz.

3.6.3 Sistema de adquisición de información

La selección del sistema de adquisición dependerá de la forma de implementación del sistema, por lo que se contemplan las siguientes opciones:

Opción 1
Pensando en compactar el módulo de búsqueda lo más que se pueda, los sistemas embebidos Banana Pi Pro y Up board tienen los recursos necesarios para adquirir la información de ambas cámaras, además de contar con pines de propósito general que podrían utilizarse, por ejemplo en el manejo de una lámpara para la cámara web, controlar motores, etc. Sin embargo la plataforma Up board se mantiene en preventa para fechas fuera de la realización de este trabajo.

La cámara FLIR A35 que se eligió, requiere de un puerto Giga Ethernet para poder realizar la transmisión del video, por ello el sistema de adquisición deberá de tener por lo menos un puerto de este tipo, nuevamente dejando como único candidato a la tarjeta Banana Pi Pro. Sin embargo, como se mencionó en el subcapítulo previo, se podrían realizar pruebas con una tarjeta de red USB - Giga Ethernet a las plataformas que no cuenten con este tipo de puerto [ver anexo B] y determinar si podrían soportar la extracción y transmisión del video al módulo de procesamiento, cuyo ancho de banda fue estimado en el anexo A.

Opción 2
Las computadoras miniatura ofrecen una mayor cantidad de recursos, que permiten hacer diferentes etapas de los algoritmos de detección o en dado caso, etapas de preprocesamiento, además de contar con un mayor grado de libertad para conectarse y comunicarse con otras plataformas, como con microcontroladores, si las necesidades de las aplicaciones de búsqueda de víctimas lo requieren.

Las dimensiones y el peso de estas computadoras son medianamente superiores a los sistemas embebidos citados, además de no cuentan con pines de propósito general punto que se compensa con su mayor capacidad de procesamiento. Al implementar alguna de los dos modelos mencionados en la Tabla 3.3 o algún otro con mejores características, se pueden integrar al sistema base, otros dispositivos de entrada o salida, como altavoces para que en un momento dado se pueda establecer comunicación con la víctima.

En ambos casos, las unidades son portables y pueden funcionar con baterías en determinados plazos. Dependiendo del móvil que transporte y contenga al sistema, se puede elegir alguna de las opciones citadas.

3.6.4 Sistema de procesamiento

Para el módulo de procesamiento no se realizó un estudio, ya que primero se contempla observar en la implementación del sistema, la demanda de recursos computacionales al tener trabajando las aplicaciones que se abordarán en el capítulo 5. Sin embargo, se partirá de determinadas características mínimas para esta unidad, que para efectos prácticos será una computadora portátil que tenga por lo menos;

- Tarjeta de red inalámbrica compatible con el estándar 802.11 b/g/n y opcionalmente ac
- Contar con un puerto ethernet y más de uno USB.
- Memoria RAM mínima de 2 GB.
- Capacidad para funcionar con alguna distribución de Linux para manejar la SDK.
- Independencia energética (bateria funcional).

Pensando en las pruebas y en la implementación en tiempo real, lo mejor sería que la computadora contará con unidad de disco duro de estado sólido. Sin olvidar que de utilizar una mini computadora con un buen procesador, esta podrá realizar el análisis de la información y únicamente enviar los resultados a otros dispositivos, o bien utilizar otra como módulo de acceso remoto y de procesamiento.

3.7 Implementación

La implementación del sistema de detección se enfocará en poner en operación el sistema base para realizar pruebas de los diferentes algoritmos para la búsqueda de víctimas, construyendo el concepto de diseño descrito en el subcapítulo 3.2, además de atender en materia de software la expansión del sistema a la arquitectura MIMO que se planteó.

El sistema base se implementará utilizando dos computadoras portátiles; una será el módulo de búsqueda que tendrá conectadas ambas cámaras, representando el sistema embebido que se integrará al robot, para que exploré la zona de emergencia y la otra computadora funcionará como el módulo de procesamiento el cual estará conectado a la primera laptop, la cual estará recibiendo las transmisiones de video e implementando las aplicaciones para la detección de víctimas.

Funcionando el sistema base, se estimarán los recursos que consume la implementación de los algoritmos de detección lo que permitirá conocer el rendimiento que deberán tener ambos módulos que conforman el sistema.

Se trabajará con el sistema operativo Ubuntu 14.04 LTS todo lo referente al sistema de detección, principalmente por qué todos los sistemas embebidos propuestos pueden utilizarlo.

3.8 Resumen del capítulo

A partir del objetivo y de la solución planteada en el capítulo anterior, se especificaron los parámetros de diseño del sistema de detección propuesto, lo que dió paso a la documentación y análisis de los diferentes equipos que podrían integrar el sistema.

Para seleccionar la cámara térmica FLIR A35 se abordaron los principios físicos de su funcionamiento, para poder acotar la búsqueda de equipos sensibles al intervalo de longitud de onda, en donde el cuerpo humano tiene una mayor potencia de radiación infrarroja, esto con el fin de contar con la posibilidad de poder detectar el pulso cardíaco de la víctima como trabajo a futuro. Respecto a la cámara web, su elección fue un poco más simple, ya que el modelo C920 de logitech tiene cualidades que la han llevado a considerarse como la mejor en el mercado a la fecha de este trabajo.

Ambas cámaras se conectarán a uno de los sistemas embebidos propuestos, los cuales cumplen con los requerimientos mínimos para transmitir la información al módulo de procesamiento, cuidando todos los aspectos de diseño. La implementación que se realizará permitirá construir y poner en operación el sistema de detección, utilizando dos laptops para representar ambos módulos y probar su desempeño.

Capítulo 4

Software de interfaz del sistema de detección

En el capítulo anterior se definió el hardware del sistema para la búsqueda de víctimas, seleccionando la cámara térmica y la cámara de video convencional, además de los sistemas embebidos que se encargarán del flujo y control de la información. Durante el desarrollo de este capítulo se expondrán las herramientas, bibliotecas y paquetes de software que realizarán la interconexión entre los elementos del sistema para adquirir la información, transportarla y presentarla para poder procesarla con procedimientos que se expondrán en el capítulo 5.

Durante la selección de los diferentes dispositivos, se consideraron sus requerimientos mínimos en materia de software, para lograr integrarlos sin limitarse o atenerse a una sola forma de hacerlo. A continuación se comprenderán los controladores necesarios para ocupar los dispositivos, protocolos de comunicación, estructuras de red, entre otros aspectos para poner en operación el sistema. Todo esto se hará utilizando como sistema operativo Ubuntu 14.04 LTS disponible para cualquiera de los sistemas embebidos citados en el capítulo 3.

4.1 Comunicación de las cámaras y transmisión de video

A continuación se explican los requisitos de software y configuraciones para comunicarse con cada una de las cámaras para adquirir el video de manera individual.

4.1.1 Cámara termográfica FLIR A35

El control y salida de video de la cámara es a través de un puerto Gigabit Ethernet, bajo el estándar IEEE 802.3af, el cual regula la tecnología de alimentación e intercambio de información sobre el puerto ethernet (PoE por sus siglas en inglés, Power over Ethernet). La comunicación del equipo se hace por la interfaz estándar GigE Vision (GEV), característica que concede la libertad de utilizar o incorporar otro modelo de cámara termográfica con este protocolo de comunicación, que permite desarrollar diferentes aplicaciones utilizando el kit de desarrollo de software (SDK por sus siglas en inglés) eBUS de la compañía Pleora Technologies.

La versión más reciente disponible (a la fecha de implementación de este trabajo) de la SDK, es la versión 4.1.5 que se implementa para poder comunicarse con la cámara y es compatible con arquitecturas ARM. También incluye el controlador eBUSd necesario para realizar la transmisión del video entre la computadora con el kit de desarrollo y la cámara, el procedimiento de instalación de estas herramientas se explica paso por paso en el anexo C, bajo el sistema operativo elegido para implementar el sistema, Ubuntu 14.04 LTS.

La cámara térmica se conecta directamente al puerto Giga Ethernet de la unidad de adquisición de datos, por medio de un cable ethernet categoría 6 con conectores RJ45. La alimentación del equipo se hace a través de su conector macho M12 y un cable con su respectiva contraparte, ocupando los pines 1 y 2 conectados a la fuente regulada de 12V que se especifica en el anexo A.

Para iniciar la comunicación con el equipo y transmitir el video, se configura una red local punto a punto, de tal forma que cada uno de los puertos ethernet tenga activa una dirección IP estática dentro de una misma red, cuyos parámetros son libres siempre y cuando se logren identificar ambos nodos de conexión, por eso se propuso trabajar con la configuración mostrada en la tabla 4.1.

Parámetros	Puerto Ethernet	
	Unidad de adquisición	Cámara térmica
Dirección IP estática	192.168.2.1	192.168.2.2
Mascara de red	255.255.255.0	
Red	192.168.2.0	
Broadcast	192.168.2.255	

Tabla 4.1 Parámetros de la configuración de la red punto a punto

Para asignar la dirección estática IP a la cámara, se utilizó la herramienta de software proporcionada por el fabricante, llamada FLIR IP Configure, el procedimiento para hacerlo se desglosa en el anexo D, junto con la configuración del puerto ethernet a la que se conecta para extraer la información.

Descripción del protocolo de comunicación de la cámara FLIR A35

El estándar GEV controla la comunicación tanto del cliente (computadora que ejecuta una aplicación) como de la cámara, dividiendo el proceso en cuatro fases como se observa en la figura 4.1 y que se describen brevemente a continuación.

Fase 1. Conexión cámara - aplicación

En primer lugar, la cámara asume una dirección IP, ya sea asignada por el usuario, por medio de un servidor DHCP o automáticamente por su configuración. Posteriormente la aplicación cliente, envía un mensaje para localizar todos los dispositivos GEV que están a su alcance (conectados a la misma red), el cual se encuentra esperando dicho llamado y por lo que al recibirlo responde con otro mensaje, para iniciar la comunicación con la aplicación.

Fase 2. Configuración por el usuario - protocolo de control de la cámara GEV (GVCP)

Posteriormente, mediante el protocolo UDP la aplicación desarrollada por el usuario, configura y controla los parámetros necesarios de la cámara para establecer la comunicación, cada uno de los datos que se envía en este proceso es confirmado, teniendo un canal bidireccional.

Para la transmisión de video, la aplicación debe realizar configuraciones básicas como el tamaño de imagen, el formato de los píxeles, entre otras, sin embargo como no todos los dispositivos cuentan con todas las configuraciones de la interfaz GEV, éstas son previamente revisadas por el estándar GenICam.

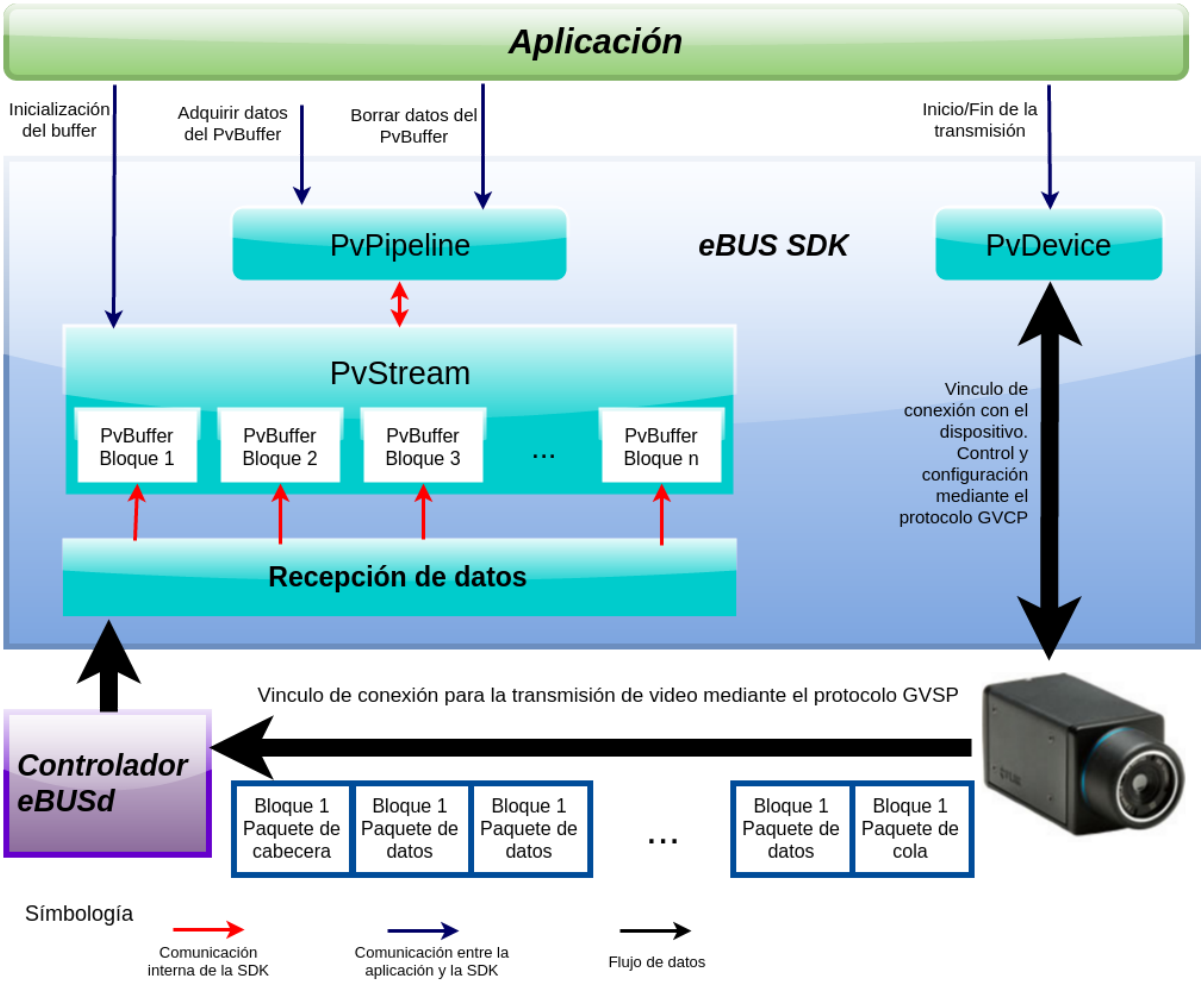


Figura 4.1 Comunicación entre la cámara térmica y la aplicación

Fase 3. Estándar GenICam - lectura de parámetros configurados de fábrica

Proporciona una interfaz unificada de programación, que contiene los atributos y diferentes configuraciones de la cámara en un archivo XML, el cual es definido por el fabricante del dispositivo ya que enlista todas sus características configurables, acorde a su nombre.

Gracias a ello, cualquier controlador GEV puede leer las configuraciones y características contenidas en el archivo XML de la cámara, prescindiendo de realizar configuraciones iniciales, por lo que leído dicho archivo por la aplicación, se procede a la transmisión de video.

Fase 4. Protocolo de transmisión de video GEV (GVSP)

La transmisión del video utiliza paquetes UDP para transportar la información entre la cámara y la aplicación. Dichos paquetes consisten de la estructura mostrada en la figura 4.2 la cual es característica del estándar GEV.

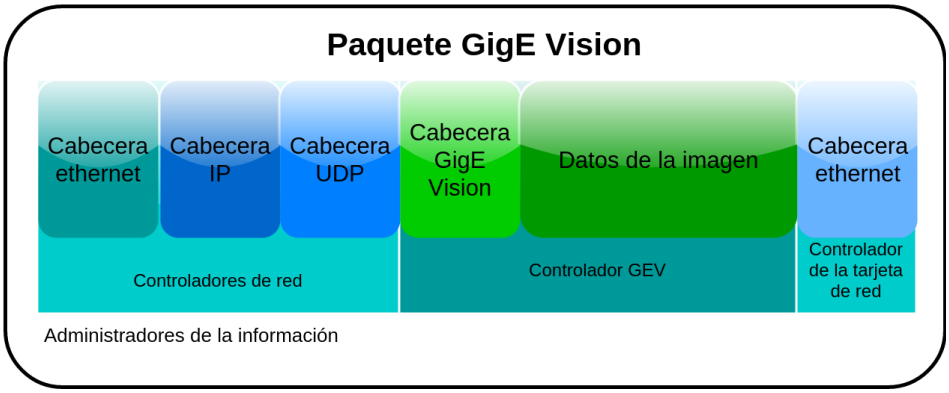


Figura 4.2 Estructura de un paquete de información para la transmisión de video bajo el estándar GigE Vision

El protocolo de transmisión de datos UDP no garantiza la entrega de los paquetes de información, sin embargo muchos de los dispositivos que utilizan el estándar GEV implementan procesos de recuperación que asegura que las imágenes no tengan pérdidas de información [8]. En resumen, el procedimiento para realizar la transmisión de video de la cámara térmica se muestra en el siguiente pseudocódigo.

//Pseudocódigo para realizar la configuración de parámetros para realizar la transmisión de video de
//la cámara térmica

Inicio

Ejecutar el controlador eBUSd

Agregar bibliotecas

PvSampleUtils, PvDevice, PvDeviceGEV, PvStream, PvStreamGEV, PvBuffer,

PvSystem

//Búsqueda y conexión de la cámara

Crear un objeto de la clase PvSystem (representa la NIC de la computadora)

Extraer y guardar la información de la NIC de la computadora en PvDeviceInfo

Realizar búsqueda de la cámara térmica en la red local y conectarla a la aplicación

//Configuración para el control de la cámara

Si la comunicación es establecida

Obtener la información de la cámara

Crear y abrir un objeto para construir la transmisión de video

//Configuración para la transmisión de video

Realizar configuraciones para la transmisión

Obtener el tamaño de los paquetes a transmitir

Indicar la dirección IP destino y el puerto destino de la transmisión

Crear buffers para la recepción de paquetes

Obtener el tamaño de la carga de la transmisión, leyendo el parámetro de la cámara

Obtener el máximo número de buffers o definirlo fijo (16 recomendados)

Alojar buffers en la memoria

Ordenar los buffers para la transmisión

Adquirir imágen

Obtener los parámetros necesarios para el control de la transmisión

Leer comandos GenICAM

Obtener parámetros para la transmisión

//El usuario ejecuta la instrucción de inicio

Habilitar e iniciar la transmisión

Adquirir imágenes hasta el envió del fin de la transmisión

Extraer imagen del buffer

Liberar buffers

Leer siguiente buffer

//El usuario ejecuta la instrucción de fin

Fin de la transmisión

Deshabilitar la transmisión

Liberar y eliminar buffers

Terminar conexión con la cámara

Fin

El código que implementa la configuración para mantener la comunicación con la cámara térmica y transmite el video se encuentra en el anexo E.

4.1.2 Cámara web logitech C920

La configuración y transmisión de información de este dispositivo se realiza por el estándar USB 2.0, soportado en linux por un conjunto de controladores y herramientas que permiten utilizar dispositivos de video usb, llamado USB Video Class Linux (UVC), el cual describe las características y configuraciones de la cámara para la transmisión de video, similar a GenICam y que ésta integrado al núcleo de Ubuntu. Para modificar parámetros de funcionamiento de la cámara, se instalaron unos repositorios del paquete V4L2.

Todos los sistemas embebidos seleccionados para la implementación, cuentan con puertos USB 2.0, simplificando la conexión entre la cámara web y la unidad de adquisición, ya que incluso por el mismo puerto, el dispositivo de video es alimentado con 5V.

A diferencia de la cámara térmica, es suficiente con instalar el paquete mencionado para utilizar el equipo en alguna interfaz, además podemos realizar diferentes configuraciones al dispositivo de video, por ejemplo, el formato de los píxeles (con soporte en formatos MJPEG, YUYV y H.264), el espacio de colores, tamaño de imagen, entre otros permitiendo incluso realizar directamente la transmisión, almacenamiento y reproducción de vídeo directamente desde la terminal.

Independientemente del controlador de la cámara, la mayoría de las configuraciones que se pueden hacer al equipo por medio de V4L2 también pueden hacerse con un conjunto de operaciones para procesamiento de imágenes y video en una sola biblioteca, llamada OpenCV, utilizada en determinados lenguajes de programación, la cual se describe de forma general en el siguiente subcapítulo.

4.2 Manipulación de imágenes y video

Implementar los algoritmos de procesamiento para la detección de víctimas, requiere de utilizar diferentes herramientas para manipular, transformar y operar los datos de las imágenes o cuadros acorde a su formato. Existen diferentes programas y paquetes de software que permiten realizar operaciones sobre imágenes y video, uno de estos medios es la biblioteca Open source Computer Vision (OpenCV). Funciona en varios sistemas operativos (incluyendo Ubuntu), soporta diferentes formatos, está diseñada para optimizar el uso de recursos computacionales, está enfocado hacia aplicaciones en tiempo real y se puede manejar en

diferentes lenguajes de programación, como C/C++, Ruby y Python, entre otras cualidades, que cumplen con los criterios de diseño del proyecto, razones por las que se decidió utilizarla. OpenCV trabaja en conjunto con los diferentes codificadores y decodificadores instalados en el sistema operativo como jpg, bmp, png en imágenes o mp4 y avi en videos. A su vez también soporta formatos de video compreso, como el H.264, siempre y cuando se tengan instalados los codificadores correspondientes y se habiliten para ser utilizados por la biblioteca.

Permite aplicar diferentes tipos de operaciones y transformaciones a imágenes o a cada uno de los cuadros de un archivo de vídeo o transmisión en vivo, así como guardar cuadros del video. Además cuenta con diferentes métodos configurables que aplican algoritmos de procesamiento de imágenes, por ejemplo, para detectar bordes, cálculo del flujo óptico, cálculo de áreas, perímetros, centroides, contornos, detección de rostros, entre otros. En el anexo E se muestra un código ejemplo de algunas herramientas de la biblioteca utilizando la cámara web, que posteriormente se adecuará con el sistema de comunicación que se verá en la siguiente sección.

Tanto al módulo de procesamiento como al de búsqueda se le instaló la versión 4.10 de la biblioteca, ya que esto permitirá realizar procedimientos previos a los algoritmos de detección, dividir tareas o codificar la información antes de ser enviada a su análisis y procesamiento.

4.3 Comunicación entre el módulo de búsqueda y el de procesamiento del sistema de detección

La comunicación entre ambos módulos se puede realizar utilizando un par de sockets con el protocolo UDP o montando un servidor, sin embargo se cuenta con una alternativa que tiene herramientas adicionales que pueden ocuparse en la implementación del sistema base así como en la arquitectura MIMO. Por ello se eligió utilizar el sistema de comunicación Robot Operating System (ROS), el cual funciona como una red tipo estrella con un nodo central o maestro que administra y controla el flujo de información entre las diferentes aplicaciones y dispositivos conectados a la red en forma de nodos, la figura 4.3 ilustra la descripción realizada de ROS.

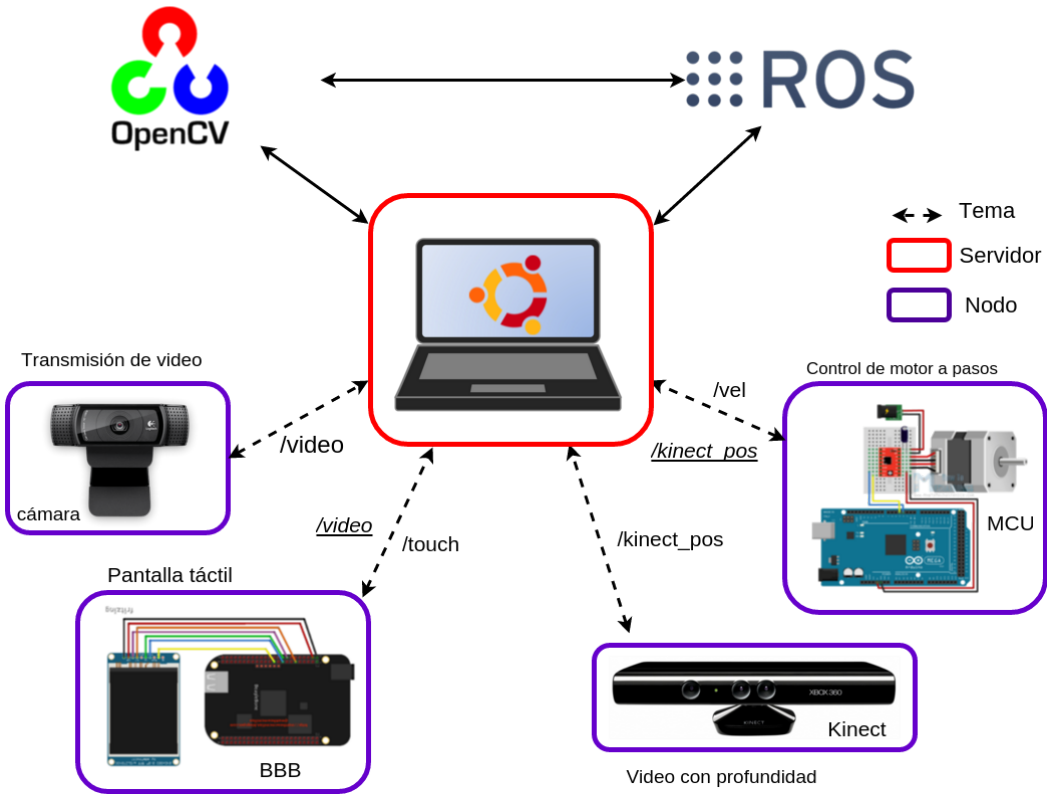


Figura 4.3 Diagrama del funcionamiento general de ROS

Todos los nodos se conectan al nodo maestro para conectarse e intercambiar información entre ellos, dichos datos son enviados por mensajes, transportados en canales de comunicación llamados “temas” (topic en inglés) implementados por conexiones alámbricas como por puertos seriales, USB y ethernet así como de forma inalámbrica. Los nodos se clasifican esencialmente en dos tipos; los publicadores emiten los mensajes de datos a través de los temas, al publicarse la información en un tema, el nodo maestro la pone a disposición de todo aquella aplicación que la requiera, es decir por el segundo tipo de nodos, los suscriptores que generalmente procesan la información.

Al ser moderador de toda la información que se intercambia en toda la red, es recomendable que el nodo maestro sea una computadora o sistema embebido funcionando preferentemente con Ubuntu, específicamente en el sistema de detección, este rol será ejecutado por el módulo de procesamiento. En cambio, los publicadores son programas dedicados comúnmente a extraer datos de diferentes dispositivos que tienen soporte en ROS, como cámaras, joysticks, sensores conectados a microcontroladores etc. y los suscriptores son códigos que trabajan con la información de los canales, ejecutados por sistemas embebidos, microcontroladores o computadoras para ejecutar procesos o acciones, como activar elementos electromecánicos o realizar cálculos para odometría.

En ambos módulos del sistema, se tiene instalada la versión ROS Indigo, la cual tiene soporte con paquetes dedicados al transporte de imágenes y video en diferentes formatos así como de audio proveniente de micrófonos, entre otros. Cabe señalar que el funcionamiento del sistema no es en tiempo real, debido a que el nodo maestro tiene que administrar el flujo de información, sin embargo, los suscriptores se sincronizan acorde a la tasa de transmisión de los publicadores, por lo que este hecho no representa un problema en la implementación, además de que el desfase de tiempo es mínimo.

Algunas de las herramientas que se están utilizando de ROS, es el soporte para trabajar junto con OpenCV tanto en C++ como en python, lo que permite utilizar la biblioteca en nodos programados en cualquiera de estos dos lenguajes. También permite intercambiar datos mediante servicios e iniciar múltiples nodos con un iniciador automático.

El funcionamiento de ROS está soportado en un sistema de archivos, en la cual los programas ejecutables deben de estar alojados en un paquete, el cual es un directorio que puede contener varios nodos y está configurado acorde a las herramientas, dependencias y bibliotecas que se declaren al momento de crear dicho paquete, que a la vez es parte de un espacio de trabajo necesario para poder trabajar con los paquetes. Este espacio de trabajo puede contener tantos paquetes como la aplicación los requiera o el usuario lo desee y de la misma manera se puede tener cualquier cantidad de espacios de trabajo en una misma computadora o sistema embebido con soporte en ROS [27]. De forma general en la figura 4.4 se puede observar la estructura del sistema de archivos descrita.

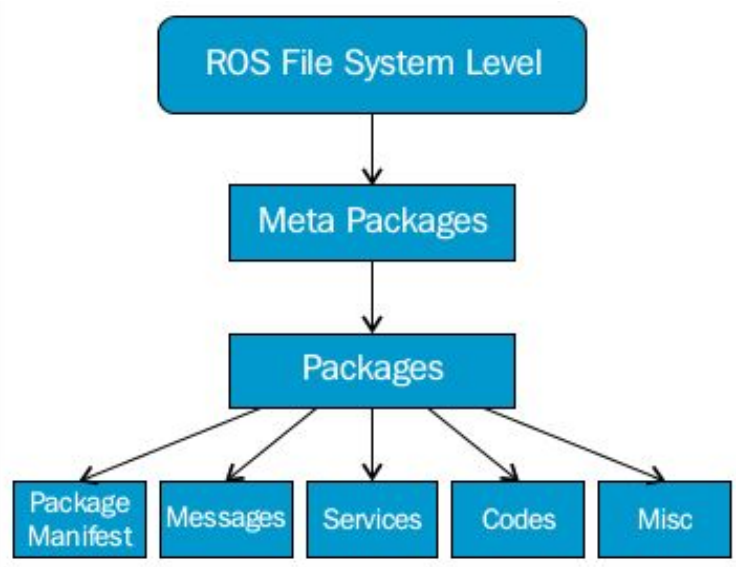


Figura 4.4 Estructura del sistema de archivos ROS

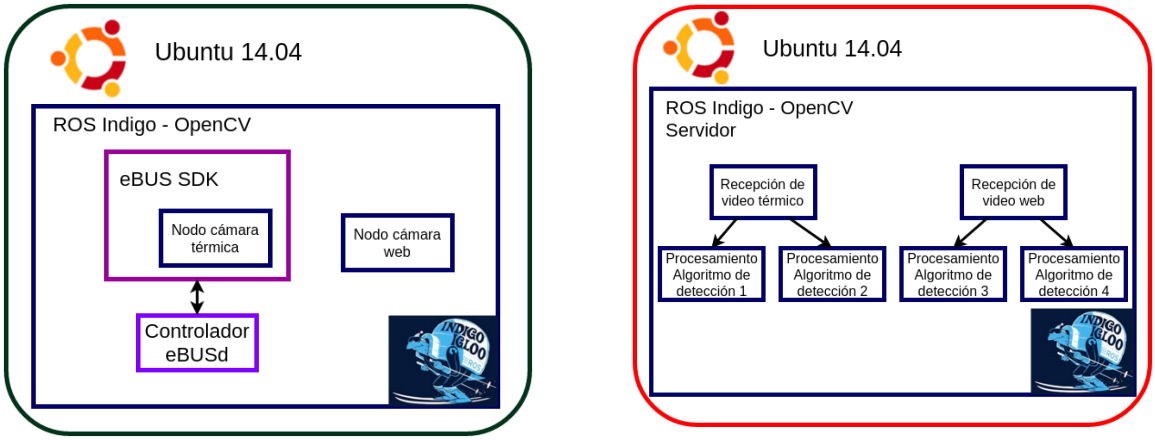
Para la adquisición de datos en el módulo de búsqueda, para cada cámara se escribió un nodo publicador, de esta manera las configuraciones para transmitir el video se hacen una sola vez, además permite utilizar los datos publicados en diferentes nodos suscriptores, por lo que

acorde a los recursos de la computadora para procesamiento se puede tener ejecutando las aplicaciones desarrolladas en el capítulo 5 de manera paralela.

Para alojar las aplicaciones del proyecto se creó un paquete llamado *find_victims* que será trabajado en ambos módulos, en contraste con el paquete *flir_a35* que estará únicamente en el módulo de búsqueda, puesto que es el encargado de configurar e iniciar la transmisión del video de la cámara térmica. Este último paquete fue modificado para la implementación a partir del realizado por Chao Qu, disponible en el repositorio de GitHub del laboratorio Kumar Robotics, de la Universidad de Pensilvania.

4.4 Arquitectura del sistema de detección

Para comenzar a abordar la arquitectura implementada del sistema de detección, en la figura 4.5 se describe de forma gráfica la todos los controladores, bibliotecas y paquetes que se instalaron y pre configuraron en cada uno de los módulos, para iniciar y detener las aplicaciones desde la terminal de Ubuntu.



A B

Figura 4.5 Configuración del software del módulo de búsqueda (A) y del módulo de procesamiento (B).

Antes de comenzar a trabajar, es necesario cargar las variables de inicio para utilizar la SDK junto con el controlador eBUSd, ya sea de forma manual o automática como se especifica en el anexo C. Ambos módulos van a estar conectados en primera instancia mediante una red punto a punto para que ambas tengan acceso remoto a través de Secure Shell (SSH) ocupando los puertos ethernet de las computadoras por lo que la cámara térmica estará conectada al módulo de búsqueda utilizando la tarjeta de red usb, mencionada en el capítulo 3 en la selección del sistema embebido.

Hecho lo anterior, ocupando la herramienta roslaunch se inicia el nodo maestro desde la computadora del módulo de procesamiento junto con los nodos que transmiten el video de ambas cámaras, el código del archivo launch se encuentra en el anexo F, el cual se puede modificar para que alguna de las aplicaciones del capítulo 5 también se inicie al arrancar el sistema.

Los archivos launch tienen un conjunto de instrucciones para iniciar todos los nodos que sean declarados, independientemente de que estos se encuentren en la computadora del nodo maestro o fuera de él, como en otra computadora o sistema embebido que pueda ejecutarlo. Además permite configurar en forma de global, variables que vayan a ser utilizadas en el código del nodo.

Para ejecutar el archivo launch, previamente se cargan las variables del ambiente de nuestro espacio de trabajo, para que se identifique el archivo launch y se inicie, al ingresar la siguiente instrucción:

roslaunch start_sys_nsv start_up.launch

Donde la roslaunch le indica al sistema que se va a ejecutar un archivo launch del paquete *start_sys_nsv* cuyo nombre es *start_up* con terminación *.launch*. Al tener los canales activos que transportan el video, el operador puede iniciar desde la terminal cada aplicación del paquete *find_victims*, de forma individual o simultánea. De manera alterna, mientras se tenga la conexión asegurada entre ambos nodos, se puede iniciar únicamente el nodo maestro en la terminal del módulo de procesamiento y posteriormente ejecutar algún nodo, a nivel de instrucción esto es;

//Para iniciar el nodo maestro
roscore

En otra terminal o pestaña se ingresa

rosrun find_victims application_name

Los nodos que se desarrollaron para el sistema de detección de víctimas no superficiales son:

1. video_web_publisher
2. video_web_monitor.py
3. thermal_publisher
4. thermal_video_monitor.py
5. color_thermal_monitor_#.py
6. get_angle.py
7. haar_detector.py
8. roi_extractor_set_wnd.py

9. canny_edge_detector.py

En el siguiente capítulo se describe cada una de las aplicaciones que corresponden a las aplicaciones disponibles en el paquete *find_victims* junto con su marco teórico. Únicamente los nodos que transmiten video directamente de las cámaras, están programados en C++, todos los demás están en python.

4.5 Resumen

Cada uno de los dispositivos de detección conectados al módulo de búsqueda, proporcionan los datos en diferentes formas, la cámara térmica al funcionar bajo la interfaz estándar GEV requiere de un procedimiento específico para poder acceder a la información del video, por ello se explicó el funcionamiento del protocolo de comunicación GEV y posteriormente los pasos para realizar la adquisición de video. El código implementado en ROS se abordará en el siguiente capítulo.

En contraste, con la cámara web no fue necesario realizar instalaciones o configuraciones bases, ya que Ubuntu tiene incluido en su núcleo los controladores para trabajar con dispositivos UVC, característica que fue contemplada para seleccionar el modelo C920 de logitech, además de poder configurar ciertas características de la cámara, instalando el repositorio V4L2.

Ambas fuentes de video serán procesadas utilizando algunos de los métodos, operaciones y transformaciones de la biblioteca OpenCV, la cual puede trabajar en conjunto con el sistema ROS, brindando la ventaja de poder utilizar los datos de las cámaras por más de un programa o aplicación además de facilitar la transmisión del video entre ambos módulos sin profundizar en materia de redes. A su vez el sistema de archivos sobre el que trabaja ROS, permite incorporar otros nodos de forma independiente, razón que convierte en portables tanto los espacios de trabajo como los paquetes.

Las bibliotecas, paquetes y controladores instalados en ambos módulos se mostraron en la figura 4.5 y se explicó la forma en la que se inicia la extracción y transmisión del video de ambas cámaras utilizando la herramienta roslaunch y también de forma individual, con la finalidad de tener funcionando los nodos base e ir ejecutando la mejor aplicación según sea el caso. Obteniendo la integración de todos los elementos del hardware del sistema, estableciendo la comunicación entre ellos listo para implementar los diferentes nodos de procesamiento que se presentan en el siguiente capítulo.

Capítulo 5

Búsqueda de víctimas no superficiales

Para terminar la implementación propuesta del sistema de detección, a lo largo de este capítulo se abordarán cada una de las aplicaciones desarrolladas para la detección de víctimas no superficiales, orientadas a la detección del pulso cardíaco.

En el capítulo anterior se mostraron ejemplos para transmitir el video de cada una de las cámaras, sin embargo, no se explicó su integración a ROS, por lo que se iniciará con la descripción de los nodos que extraen el video de cada equipo, el transporte de información entre los módulos a través de los canales de comunicación, es decir de los “temas” creados, la recepción y visualización en el módulo de procesamiento.

A partir de los dos nodos que transmiten el video de ambas cámaras, se presenta el desarrollo de nodos que aplican algoritmos propuestos para la detección de víctimas, utilizando principalmente la cámara térmica debido a que la mayor parte de los métodos,

transformaciones y operaciones utilizan cámaras de video convencional, razón por la que se propuso prestar mayor atención al procesamiento de imágenes térmicas. También se desarrollaron algunas aplicaciones con la cámara web, como la extracción del audio de los micrófonos estéreo con los que cuenta la logitech C920, que se abordará al finalizar el capítulo.

Como último punto, para intentar detectar el pulso cardíaco, una vez que se haya detectado una víctima con la metodología propuesta de reconocimiento de formas antropomórficas, se describen dos aplicaciones, para resaltar la compresión y expansión de las paredes arteriales que pueden llegar a distinguirse una vez que se enfoque alguna parte del cuerpo donde sea perceptible el pulso por medio del tacto.

5.1 Nodos básicos de transmisión y recepción de video

Para realizar transmisiones de video, ROS dispone de paquetes que interactúan con las propiedades de las cámaras para manejar adecuadamente la información de cada uno de los frames y ponerlo a disposición en la red local mediante un canal de comunicación, es decir publicando mensajes de tipo imagen en un tema.

Se instaló el paquete *image_transport* en el módulo de búsqueda, porque contiene las herramientas para programar publicadores que transmiten imágenes o video a través de ROS, permitiendo que múltiples nodos puedan utilizar la información que reciben las cámaras, en RGB y MONO en 8 o 16 bits, además permite que la información se transmita en su formato original, en algún formato de compresión y en otro tipo de codificaciones, siempre y cuando el usuario incorpore el soporte al sistema. Este paquete únicamente puede ser utilizado (a la fecha de redacción), programando los nodos en el lenguaje C++.

Otro de los paquetes que se están ocupando en ambos módulos del sistema, es el paquete *cv_bridge,* porque proporciona el soporte de la biblioteca OpenCV dentro de ROS, con lo cual es posible manipular el tamaño de la imagen o video antes y después de transportarse por el paquete citado en el párrafo anterior, modificar su espacio de colores, entre otras cosas, además de proveer los métodos para visualizar la transmisión de video en el nodo suscriptor. A continuación se describen cada uno de los nodos básicos, tanto publicadores como suscriptores de video.

5.1.1 Transmisión y visualización de video de la cámara web

Nodo video_web_publisher

La estructura del programa del nodo de ROS para transmitir el video de la cámara web consiste en lo siguiente; en primer lugar se inicializa el nodo con las herramientas que provee ROS, en esta parte se asigna un nombre al nodo, el cual debe de ser único en toda la red local. Posteriormente se define el tipo de nodo, que en este caso corresponde a un publicador, para

ello mediante las instrucciones que provee el sistema se notifica al nodo maestro que nuestro nodo creará un canal de comunicación denominado (“topic o tema”) en este caso *camera/image*.

Creado y configurado el nodo, se define la fuente de video ocupando la asignación que define Ubuntu para todos los dispositivos de video conectados al módulo de búsqueda, en este caso como la cámara térmica no es común, no es reconocida por el sistema como una cámara de video, por lo que la cámara C920 será reconocida como el dispositivo de video0, esta información se puede revisar ingresando en la terminal de Ubuntu la siguiente instrucción *ls /dev/video*,* con la cual nos desplegará una lista de todos los dispositivos de video que estén conectados. Como la definición de la fuente de video a ocupar se realiza utilizando métodos de OpenCV, también se puede configurar determinadas propiedades del equipo, como el formato o el tamaño de la imagen, previo a consultar con el sistema operativo si el dispositivo no está ocupado por alguna otra aplicación.

La cámara proporciona cuadros a la velocidad con la que haya sido configurada y fabricada, sin embargo al utilizar ROS nosotros podemos adquirir todos esos cuadros a la velocidad definida, pero a una frecuencia menor con la finalidad de no ocupar en gran proporción el procesador únicamente para esta tarea, punto importante para migrar el módulo de búsqueda a un sistema embebido compacto. Entonces se define la frecuencia a la que se publicará cada cuadro del siguiente ciclo.

Para transmitir el video en el canal de comunicación definido, previamente a la publicación se revisa que el cuadro actual a punto de ser publicado, no esté vacío es decir, que su tamaño sea diferente de cero, de ser así se convierte el cuadro a un mensaje tipo imagen para que sea transportado en el tema. La publicación de mensajes se realiza cada vez que arriba un cuadro de la cámara hasta que el usuario detenga el proceso, presionando al mismo tiempo las teclas *ctrl+c* en la ventana que se haya iniciado el nodo o si este fue iniciado por un launch, se puede detener escribiendo en otra terminal *rosnode kill video_web_publisher.*

En la figura 5.1 se muestra el diagrama de flujo de la descripción del código del nodo publicador del video de la cámara web.

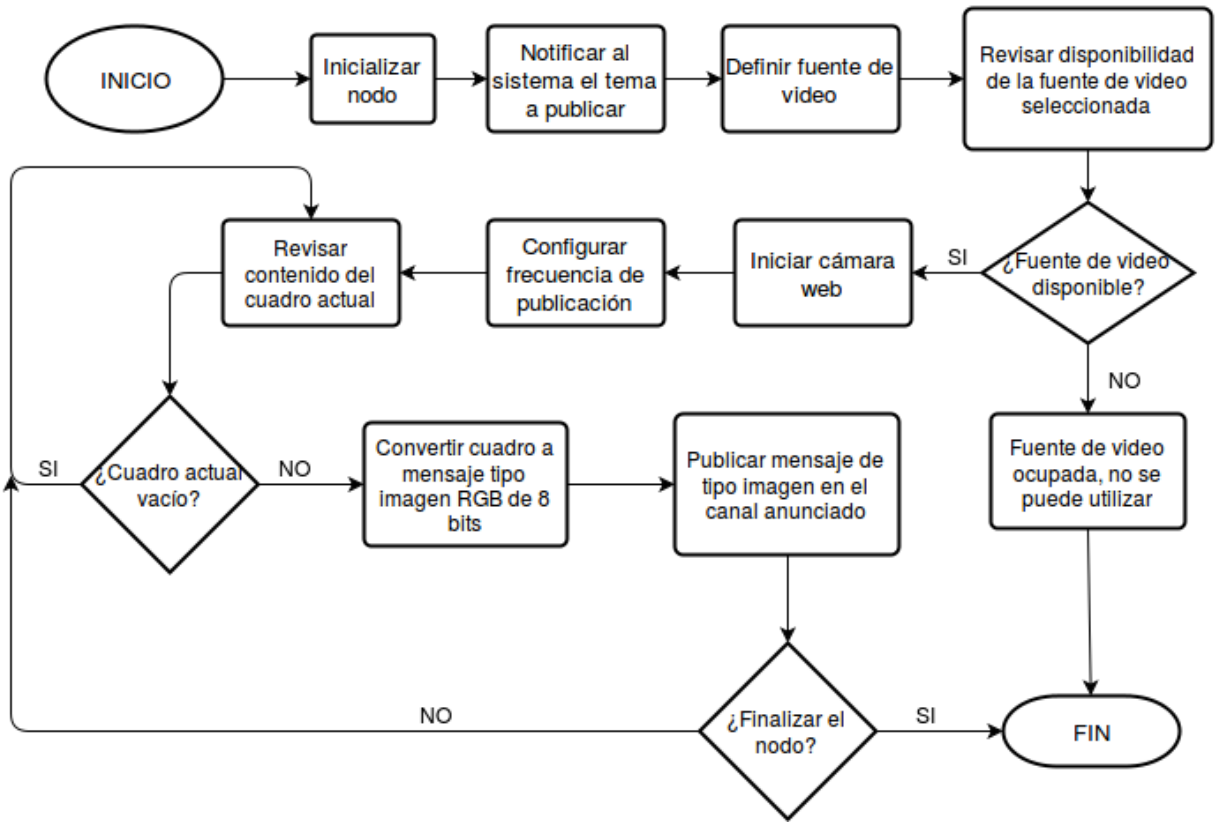


Figura 5.1 Diagrama de flujo del nodo publicador de video de la cámara web

El programa completo se encuentra en el anexo G. Una forma de revisar si el nodo está ejecutándose y además esté publicando los mensajes a la frecuencia definida es requiriendo la lista de nodos en ejecución ingresando en otra terminal *rosnode list* o en dado caso, solicitar la lista de los temas que se están publicando escribiendo en la terminal *rostopic list.*

Por último para revisar la frecuencia de publicación del tema de nuestro nodo, también en la terminal se ingresa la instrucción *rostopic hz /camera/image* y nos regresará una tasa promedio de la publicación del tema.

Nodo video_web_monitor

Un nodo suscriptor se configura indicando el tema que transporta los datos que ocupa para procesar, después de que el nodo sea identificado con un nombre ocupando las herramientas de ROS, en este caso *video_web_monitor*. Al publicarse uno o varios temas, el nodo maestro mantiene disponibles las conexiones a dicho canal de comunicación para que cualquier nodo que requiera de la información publicada, sea conectado al canal y pueda recibir los datos.

A diferencia de un nodo publicador, el suscriptor funcionará únicamente cuando llegue por el canal de comunicación un nuevo mensaje, en caso contrario se mantiene estático hasta que

arribe un mensaje. Sin embargo, como toda la información pasa por el nodo maestro, este crea buffers con un tamaño configurable, para que los nodos no se queden sin recibir información y se mantengan inactivos, a no ser que el tema al que se requiere suscribir no se encuentre activo. Al arribar un mensaje se decodifica el mensaje a una imagen manipulable para OpenCV, en espacios de colores de RGB o MONO de 8 o 16 bits y se almacena como una variable (matriz) en el programa para poder ser utilizada.

Para comprobar el funcionamiento del nodo *video_web_publisher*, en este nodo mediante los métodos de OpenCV, se despliega una ventana donde muestra el video transmitido desde el módulo de búsqueda en RGB. El código de este nodo suscriptor se encuentra en el anexo G y en la figura 5.2 se observa de forma gráfica el funcionamiento del nodo junto con el suscriptor.

A B

Figura 5.2 Imágenes obtenidas del video de la cámara web a través de ambos módulos del sistema. El diagrama corresponde a los nodos en ejecución vinculados por el canal de comunicación. La imagen (A) corresponde a una fotografía de la tarjeta Beaglebone Black y la imagen (B) de la tarjeta Raspberry Pi 2 B+.

La implementación de este nodo se realizó programando en el lenguaje Python, debido a que prescinde utilizar el paquete *image_transport* y por tener mayor experiencia en el manejo de este lenguaje. También puede ser implementado en el lenguaje C++, como se muestra en el programa del nodo en el anexo G. El tamaño de la imagen del video se puede configurar como mínimo de 320 x 256 píxeles hasta 1920 x 1080.

5.1.2 Transmisión y visualización de video de la cámara térmica

Nodo thermal_publisher

La estructura básica para transmitir video a través de ROS desde cualquier cámara es la presentada en el nodo *video_web_publisher*. Sin embargo, para la cámara FLIR A35 se tiene que hacer una serie de configuraciones para poder publicar y transportar el video. El procedimiento para transmitir el video de la cámara se desarrolló en el capítulo 4, explicando cada una de las partes del proceso, cuyo diagrama de flujo se muestra en la figura 5.3.

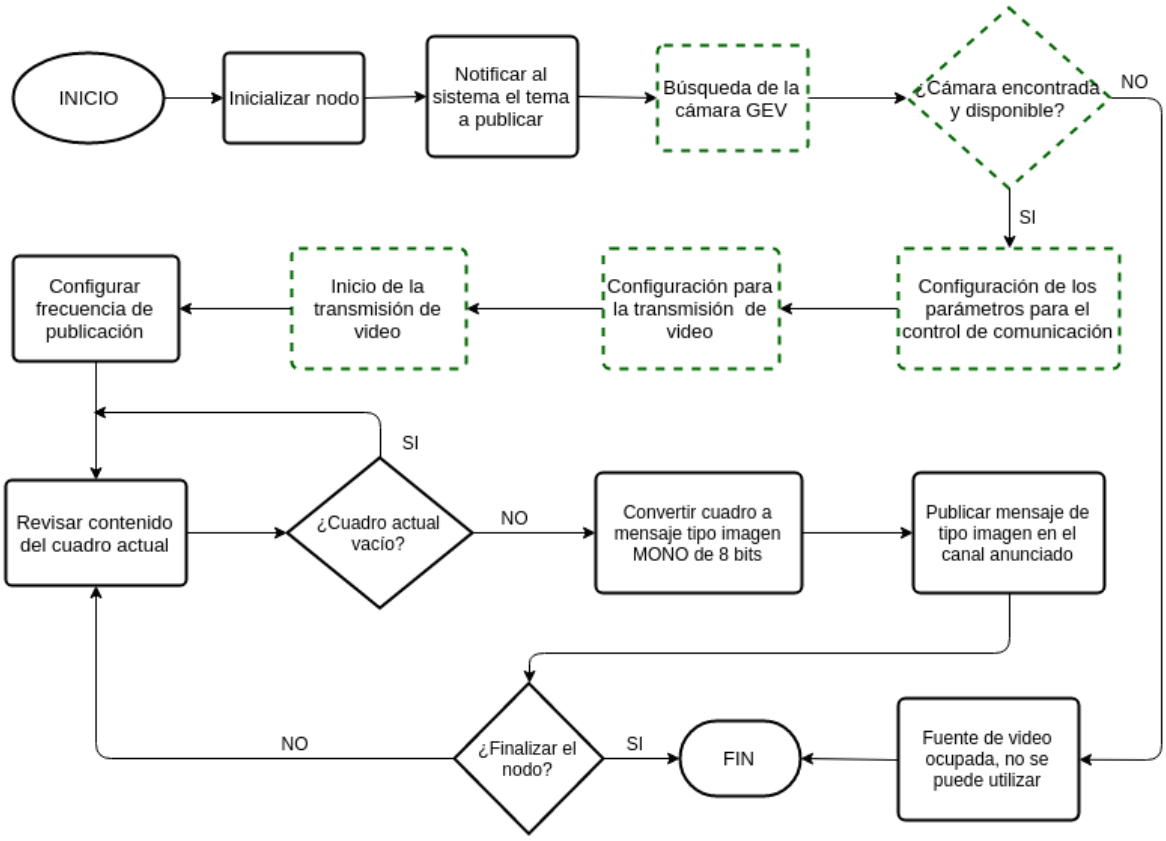


Figura 5.3 Diagrama de flujo del nodo publicador de video de la cámara FLIR A35

Los símbolos en el diagrama cuyo contorno es de color verde y con un estilo de línea discontinuo, corresponden a las etapas de configuración que se llevan a cabo utilizando la SDK y el controlador eBUSd. Antes de ejecutar este nodo se deben cargar las variables ambiente, ingresando en la terminal la instrucción

source /opt/pleora/ebus_sdk/Ubuntu-14.04-x86_64/bin/set_puregev_env

La dirección y el nombre del directorio puede variar acorde a la versión y sistema operativo. Además se debe iniciar el controlador eBUSd para trabajar con el equipo, como se menciona

en el anexo C y para detener el nodo se sigue el mismo procedimiento explicado en el nodo *video_web_publisher*.

Al estar la tarjeta de red de la cámara configurada como la red local creada, permite que la aplicación busque el equipo para establecer comunicación a través de la dirección estática IP, asignada a la cámara que es manejada como una variable de tipo cadena que está declarada en el archivo *start_up.launch* junto con otros parámetros como la frecuencia de publicación y los bits ocupados en cada píxel. Una vez que se extraen los frames son revisados antes de ser publicados como mensajes de tipo imagen en formato MONO 8, porque el paquete *cv_bridge* no tiene soporte para MONO 14.

El canal de comunicación que inicia este nodo es */thermal* a través del cual se transportan los mensajes */image_raw* que corresponden al video de la cámara sin modificación alguna. Para revisar si el nodo está activo se solicita la lista de nodos con *rostopic list /*, también se puede buscar el tema con *rostopic list* ó revisar la frecuencia con la que se está publicando el tema */thermal/image_raw*. El programa de este nodo se encuentra en el anexo G.

La cámara térmica proporciona 60 cuadros por segundo (FPS), sin embargo, al ejecutar el nodo y requerir la frecuencia promedio se observó que a pesar de declarar esta frecuencia de publicación, los procedimientos de la SDK para realizar la transmisión junto con los procedimientos de ROS provocan un retardo de aproximadamente de 8 Hertz (Hz), esto nos indica que cierta cantidad de frames se no son considerados a pesar de declarar 60 Hz, debido a los procesos que conlleva el transporte de la imagen, sin embargo, el nodo publicador permite configurar dicha frecuencia a una menor, esto se verá más adelante en este capítulo, por ejemplo a 30 Hz o 15 Hz frecuencias que no afectan a las aplicaciones posteriores, por la frecuencia promedio del pulso cardíaco de los humanos.

Nodo thermal_video_monitor

Para visualizar el video de la cámara térmica, al programa del nodo *video_web_monitor* se le realizaron unas modificaciones, las cuales fueron; asignar un nombre diferente al nodo, cambiar el tema al que debe suscribirse y especificar que se estarán recibiendo mensajes de tipo imagen MONO 8. El programa de este nodo se encuentra en el anexo G.

Nuevamente al ser un suscriptor, el nodo funciona únicamente cuando recibe un mensaje a través del canal de comunicación */thermal/image_raw*, por ello al realizar algún procesamiento se debe de considerar la frecuencia de publicación de los cuadros, para procurar procesar en el tiempo en el que arriba otro mensaje y no dejar en espera el nodo, o como se implementó en este trabajo, configurar un buffer al suscriptor para que aloje cierta cantidad de cuadros y los vaya proporcionando conforme el nodo vuelva a la fase de recepción de mensaje. En la figura 5.4 se observan los resultados de la ejecución del nodo publicador y del nodo suscriptor.

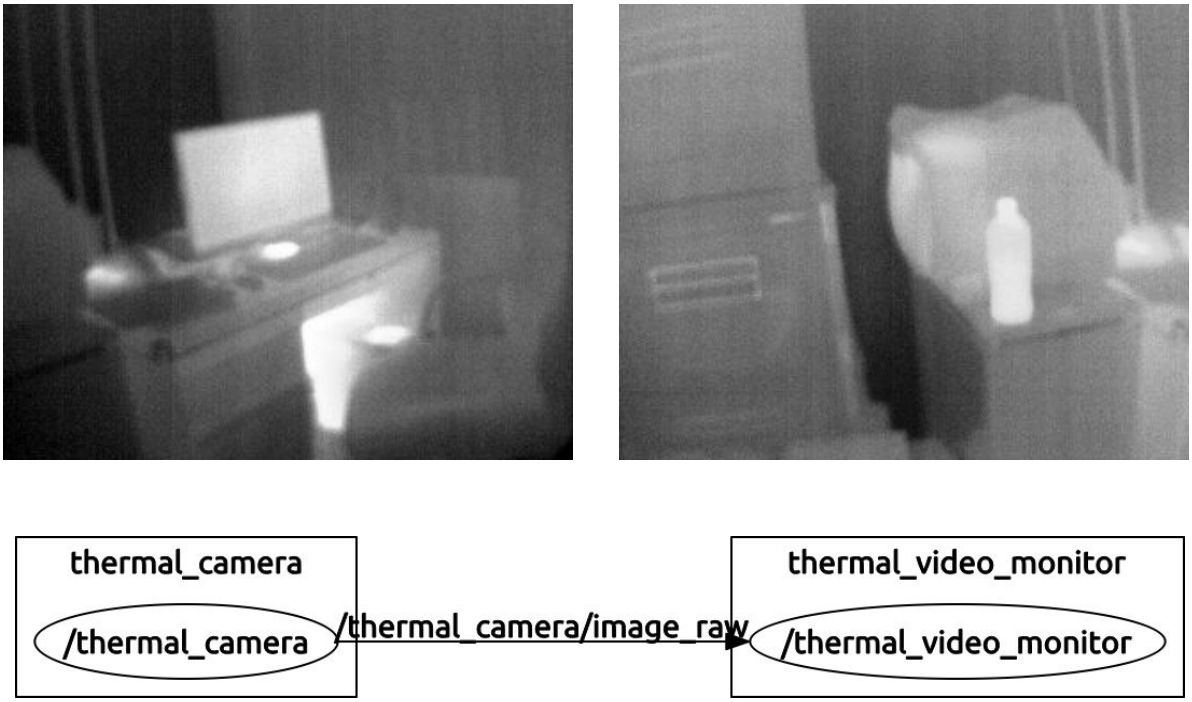


Figura 5.4 Imágenes de muestra de la cámara térmica a través de los módulos del sistema de detección de víctimas. El diagrama corresponde a los nodos en ejecución conectados por el canal de comunicación.

El video es visualizado en su formato original, escala de grises de 8 bits en formato MPEG-4, donde los valores más bajos representan fuentes de radiación IR de magnitud baja y conforme se acerca al límite superior se van haciendo más distinguibles los objetos que emiten más radiación IR. Para este trabajo se configuró la cámara para que ajustará automáticamente los niveles de la escala de gris acorde a los objetos que estén dentro de su campo de visión (FOV por sus siglas en inglés), porque la temperatura ambiente bajo los escombros normalmente es más fría que el temperatura promedio de un ser humano, información proporcionada por el equipo de perros de rescate de la UNAM, por ello es más conveniente tener la cámara con auto ajuste para resaltar los gradientes de temperatura que al mantener valores fijos.

5.2 Detección de gradientes de temperatura

La escala de grises es comúnmente utilizada en el procesamiento de imágenes porque reduce la sensibilidad a las variaciones de luminosidad, simplifica el uso de recursos computacionales y define mejor los detalles, sin embargo, para nuestros ojos dichos detalles son más perceptibles transformando los píxeles a espacios de colores, como es el caso de las imágenes térmicas.

Por esta razón, se programaron un conjunto de nodos que procesan cada uno de los cuadros recibidos de la cámara térmica, para representarlos en el espacio de colores RGB con diferentes paletas de colores que resalten los gradientes de temperatura que nos proporciona el equipo. En la figura 5.5 se muestra el algoritmo para representar los cuadros del video en escala de grises en una representación a color.

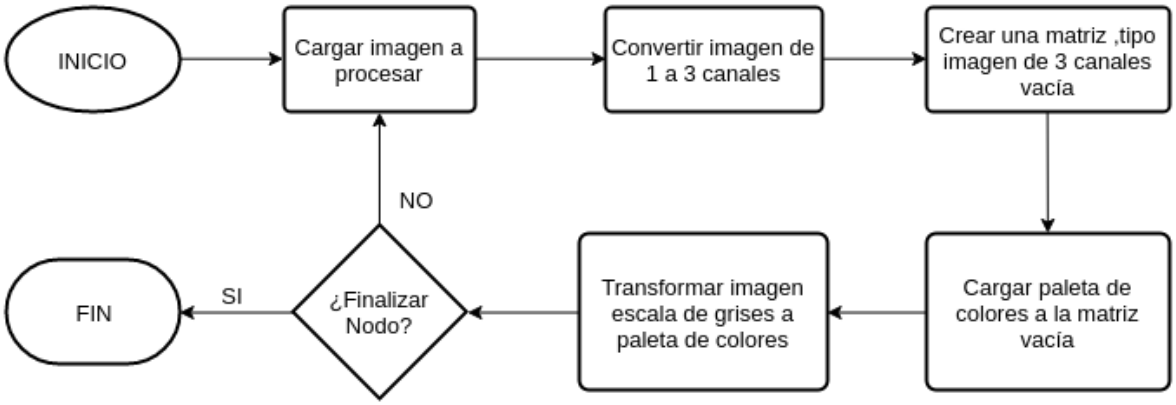


Figura 5.5 Diagrama de flujo del algoritmo que transforma imágenes monocromáticas al espacio de colores RGB con paletas de colores personalizadas

El mensaje de imagen monocromático de 8 bits recibido, que llega a través del canal de comunicación */thermal/image_raw,* se convierte al espacio de colores RGB, transformando la matriz que representa el espacio de colores de acorde a la expresión 5.1.

$$Gray\ to\ RGB[A]:\ R \leftarrow Y,\ G \leftarrow Y,\ B \leftarrow Y \tag{5.1}$$

Donde Y es el valor correspondiente del píxel en al imagen de trabajo, este valor se asigna a los canales R, G y B. Con esta transformación, la matriz del espacio de colores de la imagen adquiere los tres canales correspondientes con valores que en conjunto equivalen al mismo valor del píxel que tenía en la escala de grises.

Posteriormente se crea una matriz cuyos elementos son ceros, de la misma dimensión que el espacio de colores RGB creado es decir, de 256 x 1 x 3, para cargar los valores de la paleta de colores que se va a aplicar. OpenCV provee diferentes opciones de paletas que se pueden aplicar directamente con uno de sus métodos, sin embargo, no se ocuparon porque no resaltan los gradientes de temperatura como lo hacen los equipos comerciales, por lo que se hicieron paletas de colores personalizadas con la herramienta en línea ColorMap, cuya interfaz se muestra en la figura 5.6.

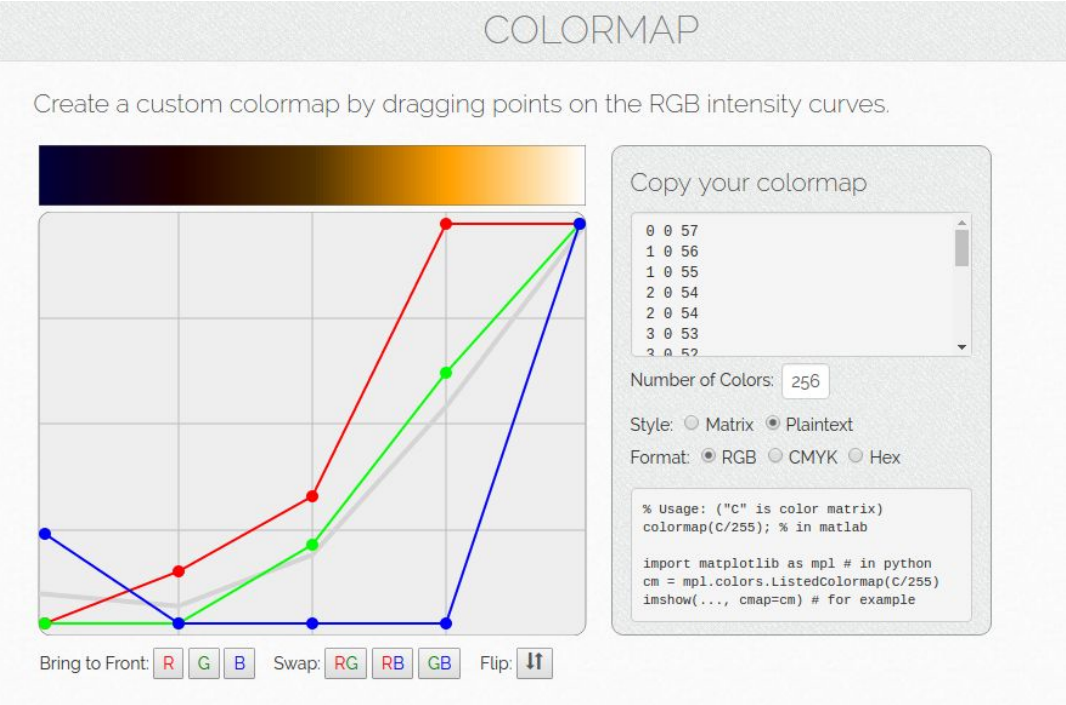


Figura 5.6 Interfaz de la herramienta en línea ColorMap

Ajustando los puntos de cada curva correspondiente a cada canal del espacio de colores RGB, se crearon paletas de colores personalizadas para imágenes de 8 bits, obteniendo 256 diferentes colores en un matriz RGB con un tamaño de 256 x 3, la cual se descargo como un archivo de texto plano .txt.

Para aplicar la paleta de colores, se abre el archivo de texto y se lee línea por línea para cargar la matriz de colores para que reemplace los elementos del arreglo de ceros que se había creado y posteriormente se ordenan los canales, porque el espacio de colores RGB utilizado en OpenCV tiene intercambiados los canales R y B, además de que es aplicado a toda la imagen mediante una convolución espacial considerando como núcleo un arreglo de 256 x 1 x 3, es decir 1 columna de 256 filas con una profundidad de 3 elementos que corresponden a cada uno de los canales.

Finalmente los valores de cada canal Por último se mapea del espacio de colores transformado por la expresión 5.1 a la matriz de colores cargada y se obtiene la representación final del video, resaltando los gradientes de temperatura capturados por la cámara. En la figura 5.7 se muestran las representaciones gráficas de los 256 tonos de las paletas de colores personalizadas que se realizaron para visualizar mejor las imágenes térmicas.

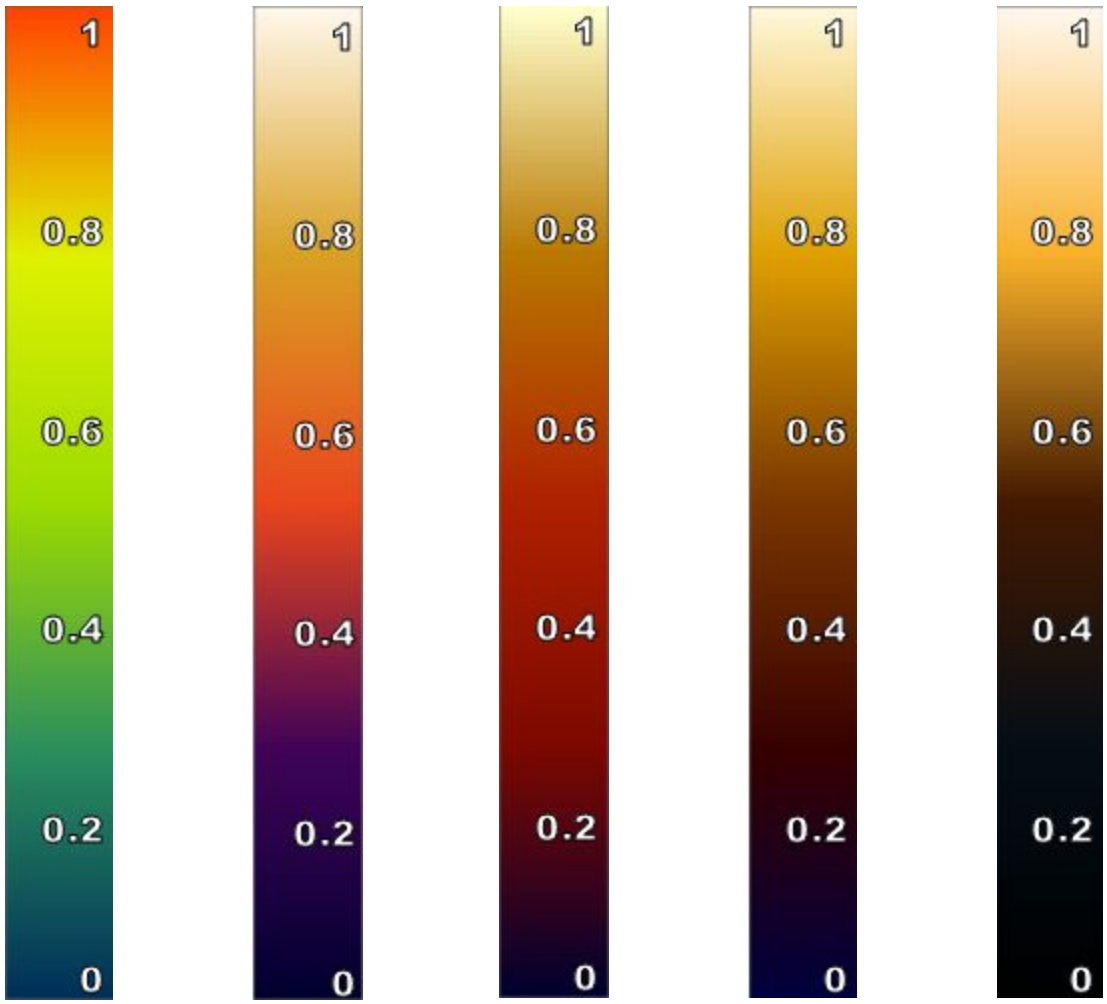


Figura 5.7 Paletas de colores aplicadas al video de la cámara térmica

Cada una de las paletas fueron graduadas para tomarse como referencia para estimar las variaciones de temperatura (gradientes) que la cámara éste detectando. También se pueden aplicar para estimar la temperatura de los objetos que se encuentran dentro del campo de visión del equipo siempre y cuando se consideren los parámetros necesarios para realizar la medición aproximada, es decir; la temperatura ambiente, la emisividad de los cuerpos y su radiación térmica tanto del objeto de interés como de los que los rodean así el valor de los píxeles, para segmentar adecuadamente la escala de colores.

Como se mencionó anteriormente, se programó un nodo por cada una de la paletas, de tal manera que la escala de colores está empalmada con la transmisión del video, en la parte derecha de cada cuadro recibido. En la figura 5.8 se muestra la imagen original en escala de grises comparada con otras imágenes a las que se les aplicó las diferentes paletas de colores.

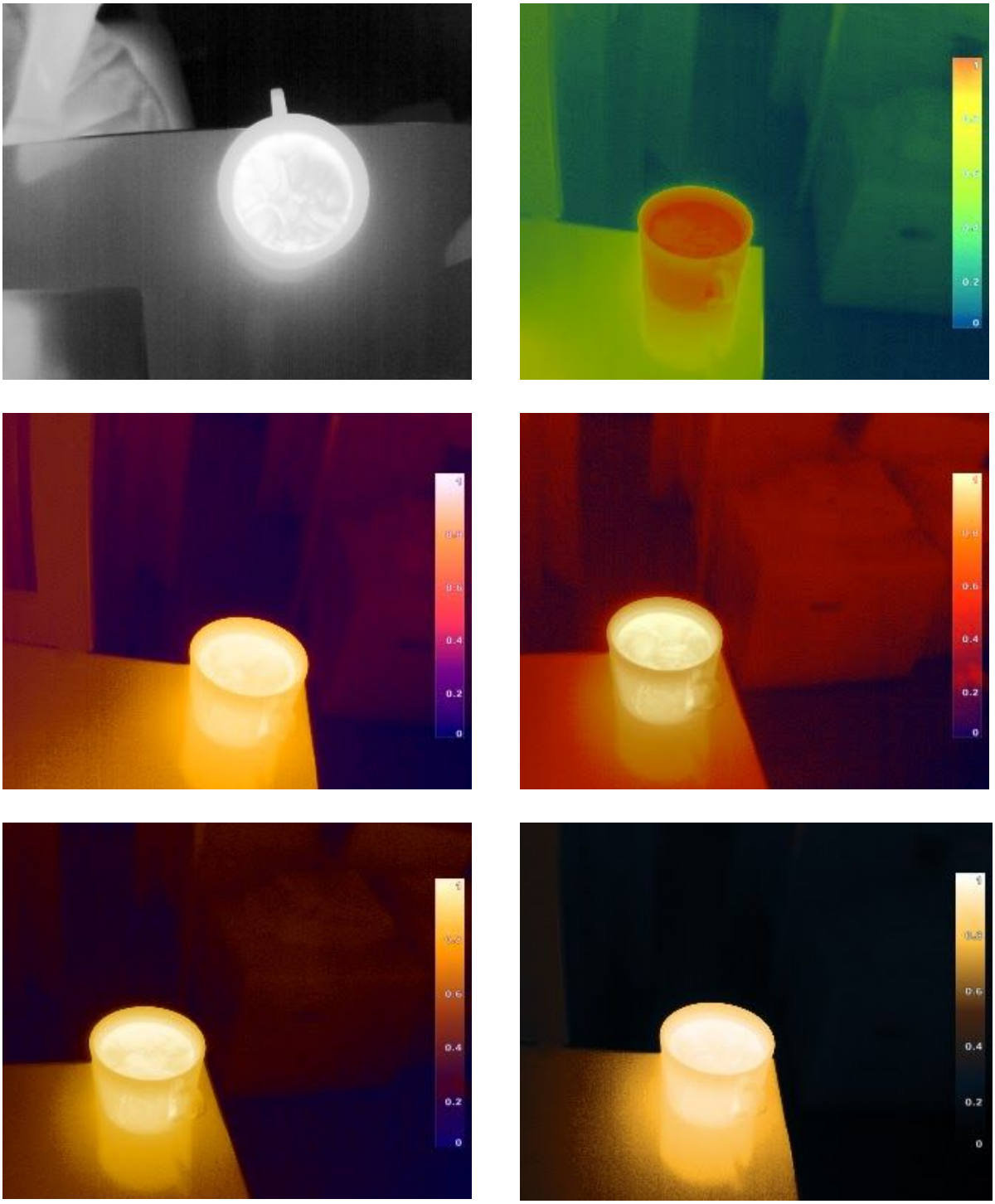


Figura 5.8 Imágenes térmicas con diferentes representaciones a color para resaltar los gradientes de temperatura.

Acorde a los gradientes que se quieran resaltar en el video de la cámara térmica, se elige la representación de colores que mejor convenga de las expuestas en la figura 5.8, puesto que

cada una tiene características útiles para distinguir diferentes elementos. El código base para implementar las paletas de colores se encuentra en el anexo H.

5.3 Segmentación de imagen por gradientes de temperatura

Antes de hacer algún procesamiento de imágenes, normalmente se llevan a cabo procedimientos previos, uno de los más frecuentes es la aplicación de filtros para eliminar el ruido aleatorio provocado por las variaciones de brillo, las representaciones de color de las imágenes o por el propio sistema electrónico de las cámaras.

La información proporcionada por las cámaras infrarrojas, no se exenta el tener ruido a pesar de ser inmune a las variaciones del brillo, por lo ello en todos las aplicaciones que se describirán en este y en capítulos posteriores, se aplicará un filtro gaussiano y una ecualización adaptativa del histograma por contraste limitado (CLAHE por sus siglas en inglés), ambos procedimientos se explican a continuación para proceder con la explicación de las aplicaciones programadas.

Suavizado de imagen aplicando un filtro gaussiano

El suavizado de datos (como de una imagen), mejor conocido como smoothing, es crear una función que aproxima el comportamiento de los patrones más importantes de los datos, reduciendo o eliminando el ruido y pequeñas variaciones que no corresponden a la información de interés de la imagen. En esta operación se modifican los píxeles originales, de tal manera que los puntos individuales (considerados como ruido) se reducen y los píxeles vecinos cuyo valor es inferior, se incrementan obteniendo la imagen filtrada o suavizada.

El tipo más común de filtro para hacer smoothing, son los lineales en los cuales el valor de píxel resultante, por ejemplo $a\ (i,j)$ es determinado como una suma ponderada de los valores de los píxeles de entrada $f(i+k,\ j+l)$ como:

$$p\,(i,j)\ \ =\ \ \sum_{k,l} f\,(i+k,\ j+l)\,h\,(k,l) \tag{5.2}$$

Donde $h(k,l)$ es denominado como el núcleo del filtro, que corresponden a sus coeficientes que operan sobre cada uno de los elementos de la matriz original de la imagen, definidos por $k\,y\,l$ cuyo límite está en función del tamaño de la imagen. Un filtro lineal de imagen puede entenderse como una convolución espacial de los valores del filtro y el de los píxeles de la imagen [18].

Acorde al tipo de núcleo, los filtros lineales se clasifican de diferentes formas, siendo uno de ellos el filtro gaussiano que se realiza mediante la convolución de cada punto de la matriz de entrada con un núcleo gaussiano definido por la expresión 5.3, y posteriormente se suman todos ellos para producir la matriz de salida.

$$G_0(x,y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(x-\mu_y)^2}{2\sigma_y^2}} \tag{5.3}$$

Donde σ es la desviación estándar, A es la amplitud del filtro, x e y representan las columnas y filas de la matriz del núcleo y μes la media (pico de la campana) [19]. OpenCV tiene predefinido un método para aplicar este filtro, configurando los parámetros correspondientes al núcleo que se describieron previamente de la ecuación 5.3, la instrucción de dicho método se puede apreciar en los programas de las aplicaciones que se mostrarán en los siguientes subtemas.

Ecualización Adaptativa de Histograma por Contraste Limitado (CLAHE)

Para comenzar, el histograma de una imagen es la representación gráfica de la distribución de tonos de una imagen, donde se visualiza la cantidad de píxeles que tienen un mismo valor de tonalidad [20]. Esta gráfica trabaja únicamente con un canal de tonos, por lo que de trabajar con imágenes de 3 canales, se tendría un histograma para cada uno de ellos.

La ecualización del histograma se utiliza para ajustar el contraste total de una imagen, generalmente tiende a incrementarlo cuando los valores de los píxeles son muy parecidos entre sus vecinos, teniendo como resultado una distribución más uniforme del histograma. El problema de esta operación es que también incrementa el contraste del ruido en la imagen y decrementa los detalles de los datos de la imagen. Sin embargo, esta técnica es ampliamente utilizada para imágenes con fondos o planos de imagen que sean oscuros o brillosos, como en el caso de las imágenes de rayos X, pero al ajustar el contraste de forma general en las imágenes térmicas, para los objetivos del proyecto sería contraproducente por la pérdida de detalles, una de las modificaciones de este método contrarrestan este y otros efectos, como la ecualización adaptativa del histograma (AHE por sus siglas en inglés).

La operación AHE calcula varios histogramas, de las diferentes partes en las que secciona la imagen, redistribuyendo los contrastes de la imagen de forma local y no global como lo hace la ecualización normal. Sin embargo, al operar la imagen seccionada tiende a amplificar ruido en regiones relativamente homogéneas en las que localice, por lo ello para realizar el procesamiento en este trabajo, se aplicó esta técnica con la variación denominada por contraste limitado.

La ecualización de histograma por contraste limitado (CLAHE) trabaja nuevamente con secciones de imagen, de las cuales calcula su AHE, pero si algún segmento de la sección está por encima del valor límite de contraste especificado, las bandas del histograma se recortan y se distribuyen uniformemente a otras secciones antes de que se aplique la ecualización del histograma total y posteriormente se aplica una interpolación bilineal, para eliminar los efectos de los bordes de los intervalos re distribuidos [19]. En la figura 5.9 se observa una imagen

adquirida por la cámara térmica y esta misma aplicando las operaciones AHE y CLAHE utilizando los métodos de la biblioteca OpenCV.

A B

C

Figura 5.9 Imagen original (A) e imágenes con ajuste de contraste utilizando ecualización adaptiva de histograma (B) y ecualización adaptiva de histograma por contraste limitado (C).

Finalizada la reseña de estos filtros para imágenes, a continuación se describe la segmentación de imágenes por gradientes de temperatura.

5.3.1 Detección de regiones de imagen acorde a su área

Las imágenes que proporciona la cámara FLIR A35 están en un formato monocromático (gris) de 8 bits, donde cada tonalidad de gris representa un gradiente de temperatura proporcional a la radiación IR que incide en la lente, de los cuerpos que estén dentro del campo de visión del equipo.

Al trabajar con la cámara en modo autoajustable, la asignación de tonos en la escala de grises para todos los píxeles se ajusta acorde a la radiación que percibe el detector FPA, por lo que considerando determinados parámetros como la temperatura ambiente del lugar en donde se adquiere la imagen, la emisividad del objeto con gradientes mayores de contraste, entre otros (como se vieron en el capítulo 3), se puede estimar su temperatura sin contacto alguno.

El espacio de colores gris es comúnmente utilizado en el procesamiento de imágenes, por la simplificación que representa trabajar con un solo canal y por ser menos sensible a los cambios de luz, entre otras razones por las que se aplica para segmentar imágenes o detectar objetos de interés, conocido como detección de blobs.

Un blob es un conjunto de datos binarios agrupados en una sola entidad seleccionados por ciertas características [21], es decir, las regiones que se llegan a formar al binarizar una imagen aplicando un determinado criterio.

Al definir regiones de imagen acorde al valor de sus píxeles, se pueden identificar fuentes de calor, esto puede detectar parcialmente personas en movimiento o víctimas, al segmentar la silueta de los cuerpos, por esta razón se implementó un nodo en ROS que identifica regiones de imagen acorde a sus gradientes de contraste (representaciones de las diferencias de temperatura), el diagrama de flujo el algoritmo se muestra en la figura 5.10.

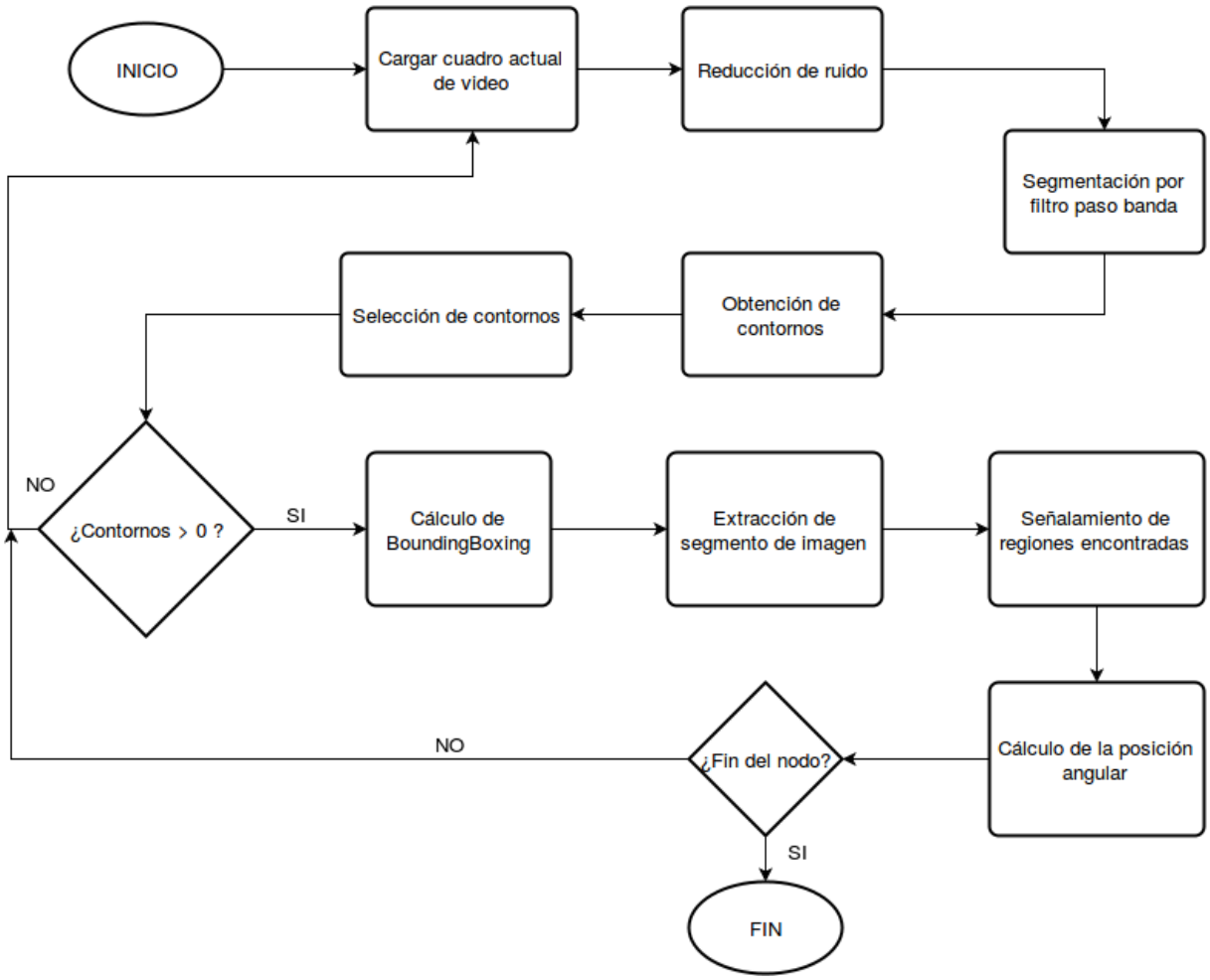


Figura 5.10 Diagrama de flujo para la detección de secciones en un video con un área determinada

Las etapas del ***algoritmo de detección de regiones con un área determinada*** consiste en las siguientes etapas.

Reducción de ruido*;* en esta etapa se aplica el suavizado gaussiano y la ecualización adaptativa del histograma por contraste limitado que se describieron previamente a la presentación del algoritmo.

Segmentación por filtro paso banda*;* es un filtro que selecciona todos aquellos píxeles que se encuentren dentro de un intervalo establecido, les asigna un valor y los aísla del resto de la imagen, definiendo segmentos de imagen, esto puede ser expresado como en la ecuación 5.4.

$$out\,(i,\,j) = \begin{cases} x & Si\ min\ th > src\,(i,j) > max\ th \\ 0 & Cualquier\ otro\ caso \end{cases} \tag{5.4}$$

Donde x es el valor determinado que se asigna a todos los píxeles que tengan valor dentro del intervalo establecido, $src(i,j)$ es la imagen original de entrada y $out(i,j)$ es el resultado del filtro. Con esto se definen todos los BLOB's de forma general, puesto que en la imagen de salida nos mostrará todas las regiones que cumplan con el criterio de umbral.

***Obtención de contornos**;* definidas las regiones en la imagen binaria que resultó del filtro anterior, se calculan los contornos de todos los BLOB's y se almacenan para su posterior utilización.

***Selección de contornos**;* definiendo un intervalo de área en pixeles, se calculan las áreas de todos los contornos y se verifica si dicho valor está dentro de los límites establecidos.

***Cálculo del cuadro delimitador**;* se calcula y obtiene, para todas las regiones que hayan logrado superar el filtro de área, un cuadro que contenga toda la región (conocido como *bounding box* en inglés). Se analiza la ubicación espacial en la imagen de los cuadros para formar uno solo en caso de presentarse regiones cuya distancia entre ellas sea tal que los cuadros delimitadores se lleguen a empalmar, por lo que se fija un porcentaje permitido de empalme para mantener los cuadros delimitadores separados o en dado caso unificarlos.

***Extracción de la imagen**;* en esta etapa se extrae la sección de imagen que delimitan los cuadros calculados anteriormente y se muestran por separado. La región definida por el bounding box puede ser variable en el video siempre y cuando cumpla con el criterio de selección.

***Señalamiento de las regiones encontradas**;* los cuadros delimitadores finales son marcados en la imagen original para observar la sección de la imagen original corresponden dichas áreas.

Cálculo de la posición angular; a partir del blob que se seleccionó, se obtiene su centroide C(x, y) y se definen dos vectores considerando como centro el punto O(165,0). El primer vector se define como $\overline{OC}$ y el segundo como $\overline{OCI}$, donde CI es el punto (165,128) y con estos se calcula el ángulo entre ellos para obtener la posición angular del segmento de imagen detectado.

Con este algoritmo se pretende detectar fuentes de calor estáticas o en movimiento, sin importar la distancia a la que se encuentren de la cámara infrarroja, aislar un segmento de imagen y analizarlo con diferentes procesamientos, considerando que el tamaño del segmento puede ser variable. Sin embargo, para garantizar que las regiones detectadas corresponden a personas, se desarrolló un conjunto de clasificadores que se describen en el siguiente subcapítulo.

5.4 Detección de formas antropomórficas o personas

El estado en el que puedan encontrarse las víctimas no superficiales se desconoce completamente, pero con base en la documentación de hechos ocurridos [1][2] se pueden estimar nociones de él, por ejemplo el tipo de heridas que pueden presentar, la postura física en la que pueden encontrarse así como de los espacios en los que pueden estar atrapadas bajo los escombros.

Considerando las situaciones límite, por un lado las víctimas pueden estar en triángulos de vida resguardados de lesiones graves, y por el otro pueden haber quedado total o parcialmente sepultadas en espacios estrechos, suficientes para mantenerlos a salvo, dentro de todo el intervalo es probable que se encuentren víctimas inconscientes por el derrumbe, personas con alguna parte de su cuerpo atrapada por escombros, entre otras, por ello al introducir el módulo de búsqueda a zonas de difícil acceso con un robot capaz de desplazarse sobre y entre los escombros, es muy probable que no se pueda visualizar con la cámara completamente a la persona, sino únicamente una parte de ella, hecho por el cual se propuso detectar formas antropomórficas sin la necesidad de un observador.

La detección de personas y partes de su cuerpo es un tema que ha sido estudiado y trabajado en el área de procesamiento de imágenes para diferentes aplicaciones, teniendo como resultado diversos algoritmos detectores, acorde a las condiciones en las que se requiera buscar. Para nuestro caso de estudio, nos interesa aplicar alguna de estas técnicas de procesamiento con la cámara infrarroja, debido a que la mayor parte de ellos han sido implementados con cámaras de vídeo convencional, con el fin de probar su desempeño en la detección de víctimas no superficiales y dar pie a la localización de regiones susceptibles para detectar el pulso cardíaco.

Por consiguiente, para detectar víctimas no superficiales se propuso e implementó un conjunto de clasificadores Haar en cascada, para detectar manos, cabeza y cuerpo completo. A continuación se presenta un breve marco teórico de estos clasificadores, el entrenamiento de las cascadas y el programa que las implementa.

5.4.1 Clasificadores Haar en cascada

Esta técnica de reconocimiento aprende a identificar objetos basado en un número de muestras positivas y negativas diferenciandolas por características simples de contraste, para decidir si una subimagen contiene el objeto de interés a detectar.

Este algoritmo fue propuesto por propuesta por Viola y Jones en 2001 [22], introduciendo el concepto de característica Haar-like como; la diferencia de la suma del contraste de los píxeles de dos conjuntos de secciones (representando con color negro al conjunto de menor intensidad y con blanco al de mayor), dichas divisiones cuadriláteras son producto de la transformada

discreta wavelet de Haar bidimensional no estándar [23]. Por ejemplo, el valor de una *característica de dos rectángulos* es la diferencia entre la suma de los píxeles de las dos regiones rectangulares, en la figura 5.11 se muestra de forma gráfica la característica mencionada así como algunas otras aplicadas, un conjunto grande de estas características forman un clasificador [21].

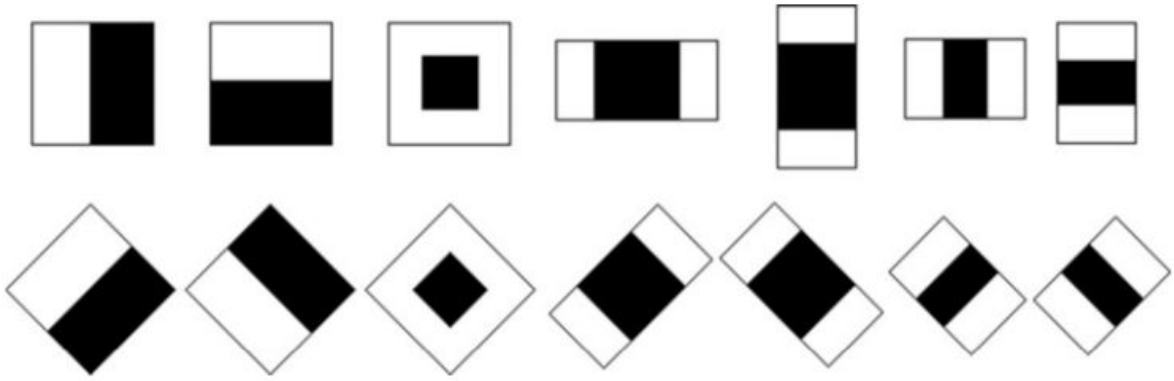


Figura 5.11 Algunas características Haar-like utilizadas en los clasificadores en cascada

En la fase de detección, la imagen de entrada es escaneada por una ventana que contiene una parte de la imagen, donde se aplica cada una de las características del clasificador (las cuales pueden llegar a ser más de 6000), las diferencias de contraste son comparadas para aprender a distinguir los objetos que se desean detectar y los que no.

Sin embargo, no todas las posibles características son aplicadas al banco de imágenes positivas que se utilizan para el entrenamiento, porque esto consumiría muchos recursos, por ello se utiliza el algoritmo AdaBoost, el cual se describe a continuación.

Al inicio a todas las imágenes se les asigna un mismo peso, además se establece un contraste deseado para las características y un error mínimo para el clasificador. Al calcular la primera característica en todas las imágenes de entrenamiento, se disminuye el peso inicial a todas aquellas imágenes que cumplan con el contraste establecido y se aumenta el peso a las que no cumplan con este factor, después se verifica el número de muestras negativas que se detectaron como positivas con la primer característica, si este error es superior al establecido para el clasificador, se repite el procedimiento para la segunda característica. Se aplicarán *N* cantidad de características a las muestras de entrenamiento hasta que se alcance el error mínimo para el clasificador [22].

Acorde al tipo de objeto que se desee identificar mediante este método, se necesitarán *N* cantidad de clasificadores, sobretodo si se trata de imágenes de cuerpos dinámicos, para obtener buenos resultados. Razón por la que los clasificadores suelen organizarse en grupos

para aplicarse de forma secuencial, a este conjunto de clasificadores se les conoce como cascadas cuyo entrenamiento se muestra en el diagrama de flujo de la figura 5.12.

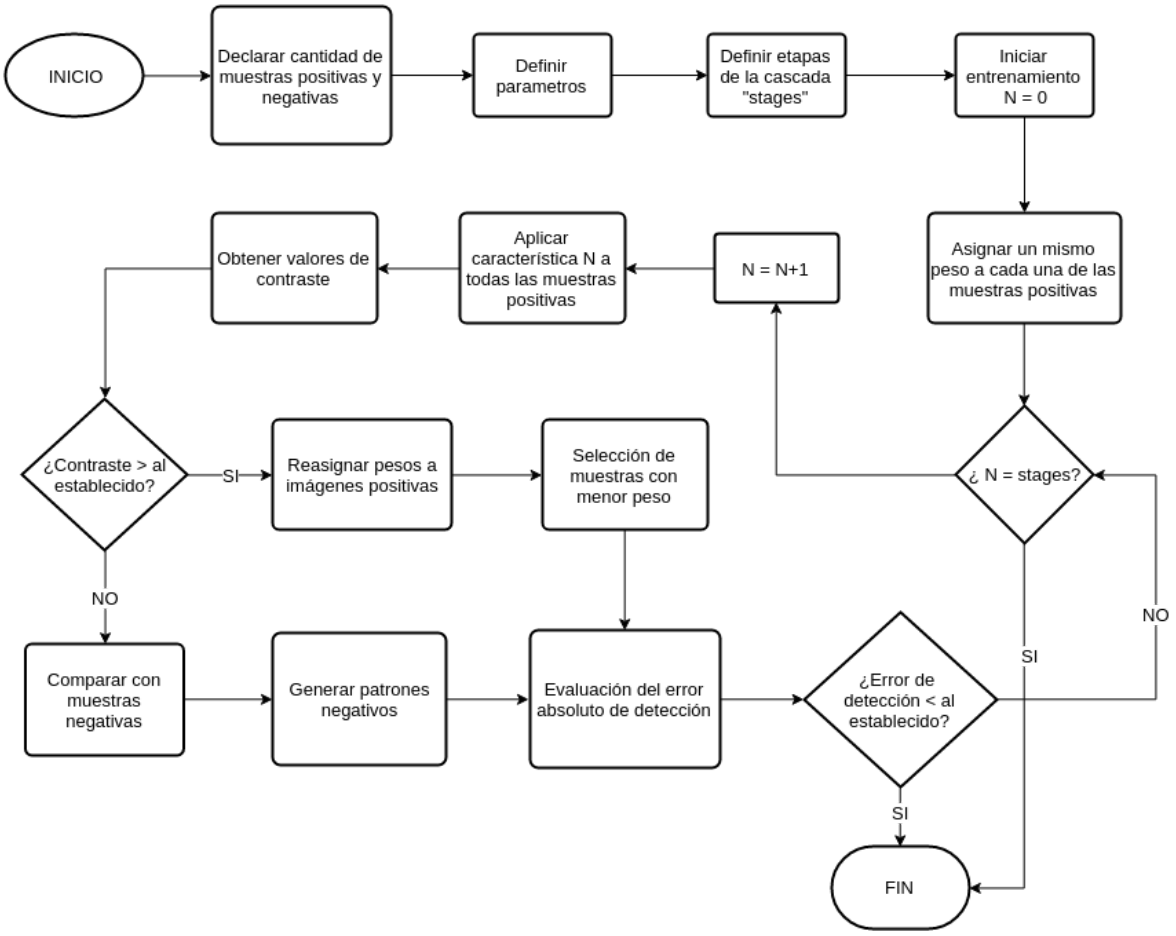


Figura 5.12 Diagrama de flujo del algoritmo implementado para el entrenamiento de un clasificador en cascada Haar

Para obtener una cascada robusta se deben de aplicar varios clasificadores sin excederse, porque la colección puede llegar a ser tan discriminante al grado de no detectar las muestras positivas proporcionadas para el entrenamiento. Por eso es recomendable utilizar una cantidad de muestras grande considerando diferentes perspectivas, distancias y posiciones del objeto, entre otros factores para poder obtener cascadas con que proporcionen buenos resultados.

La implementación de este tipo de detector, consiste en aplicar las características obtenidas por la cascada de clasificadores a una imagen o secuencia de ellas (vídeo), para comparar los parámetros e indicar si en el cuadro analizado se encuentra el objeto para el que se fue entrenado. Generalmente el resultado de la aplicación se realiza encerrado en un rectángulo el objeto detectado. Con las herramientas de entrenamiento de OpenCV se pueden obtener cascadas que trabajen a diferentes escalas y los métodos permiten obtener directamente las

coordenadas en píxeles del cuadro que contiene el objeto detectado. El nodo de ROS que se programó para implementar los clasificadores en cascada entrenados, se encuentra en el anexo J junto con los programas para seleccionar y obtener el conjunto de muestras así como el procedimiento para obtener las cascadas, mostrando los resultados obtenidos en el siguiente capítulo.

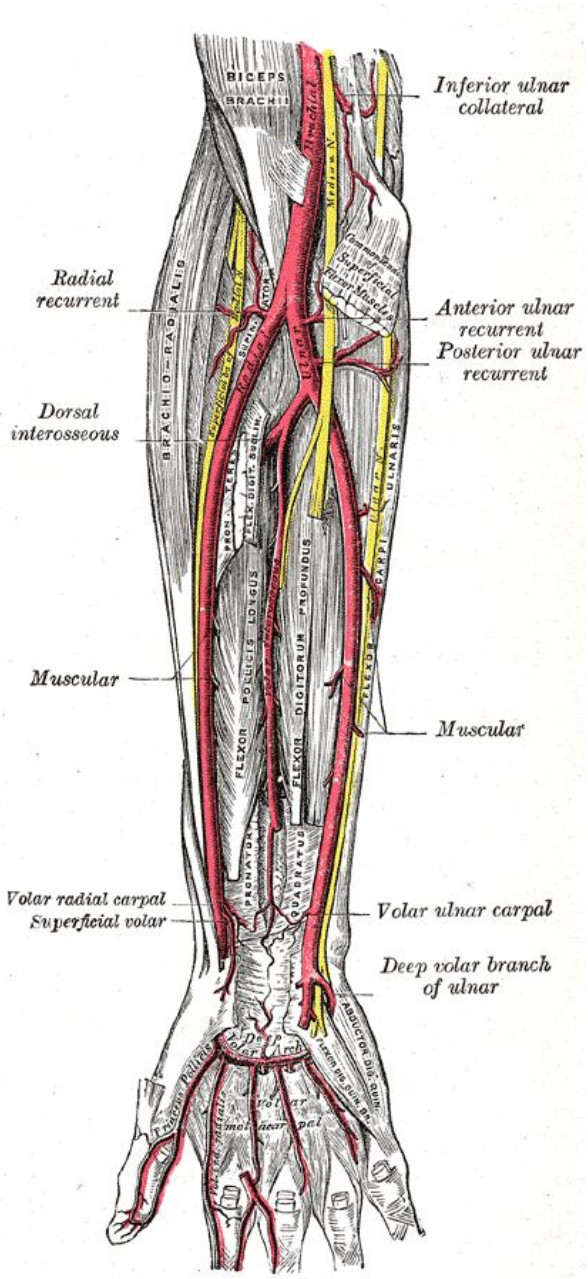
5.5 Búsqueda y extracción de región de interés

Si alguna de las cascadas logra identificar a una víctima, de ser posible la cámara se trataría de direccionar y/o acercar en la medida de lo posible, hacia alguna de las partes del cuerpo en donde se pueda ajustar la escala de grises, para resaltar los gradientes de temperatura debidos al torrente sanguíneo.

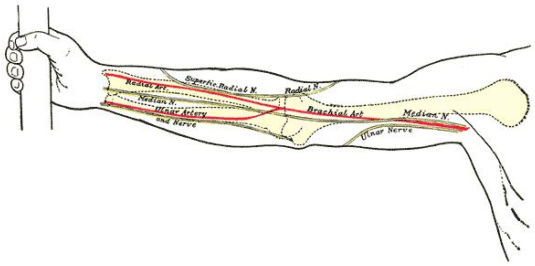
El pulso cardíaco es la frecuencia de expansión de las arterias al circular la sangre a través de las arterias, hecho por el cual bajo determinadas condiciones, provoca una ligera diferencia de temperatura, alrededor de la arteria y la piel. Este signo vital se puede medir al tacto en las partes del cuerpo donde ciertas arterias están cercanas a la superficie de la piel, razón por la que se eligió hacer clasificadores en cascada Haar para detectar manos y cabezas, porque en las muñecas se tiene parte de la arteria ulnar, en la sien la arteria temporal y en el cuello parte de la arteria carótida externa, entre otras que son probables de poder tener a cuadro.

La temperatura promedio del cuerpo humano (en condiciones normales) oscila en el intervalo de 36.5 a 37.5 °C, este valor ha sido promediado a partir mediciones realizadas en el recto, que es nuestro núcleo de calor, promediando una temperatura de 37°C, sin embargo, la temperatura promedio del cuerpo es absoluta para todas las partes del cuerpo, debido al funcionamiento de los diferentes sistemas y órganos que lo componen [24].

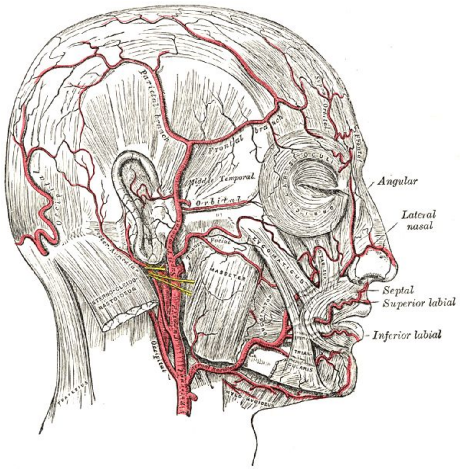
El flujo de sangre que bombea el corazón a través de las arterias, presenta diferencias del orden de 0.2 a 0.5 °C con respecto a la temperatura oral que en promedio se encuentra a 36.6 °C, valor superior al intervalo de la temperatura de la piel que es de 32 a 35 °C. Todos estos datos se mantienen bajo ciertas condiciones, la principal considerando que la persona se encuentra en un clima templado (cuya temperatura sea menor al promedio del cuerpo), además de no presentar exaltaciones y las diferencias mencionadas son más visibles a temperaturas ambientales entre los 15 y 20 °C [25]. En la figura 5.13 se muestra las partes del cuerpo donde algunas arterias están más cercanas a la superficie de la piel.



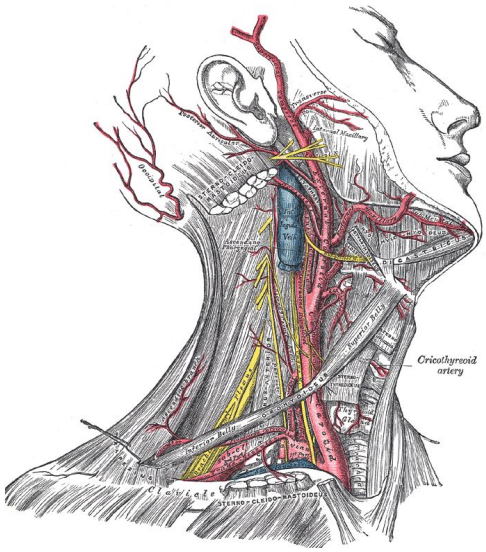
Arteria radial



Arteria lunar



Arteria temporal



Arteria carotida externa

Figura 5.13 Ubicación de arterias superficiales en las cuales es perceptible el pulso cardíaco, por lo que serán buscadas para extraer la región de interés.

La cámara infrarroja seleccionada, acorde a sus especificaciones puede detectar variaciones de temperatura del orden de 0.05 °C, diferencial que es considerablemente menor a las tasas promedio de temperatura presentadas por las arterias en comparación con la piel. Partiendo de la idea de que la temperatura bajo los escombros es de templada a fría, se mantendrán las correspondientes diferencias de calor en el cuerpo de la víctima, por ello se implementó un nodo de ROS para buscar estos puntos en la piel para aislar una región de la imagen original y proceder a ser analizada con el objetivo de detectar el pulso. En la figura 5.14 se muestra el diagrama de flujo de esta aplicación.

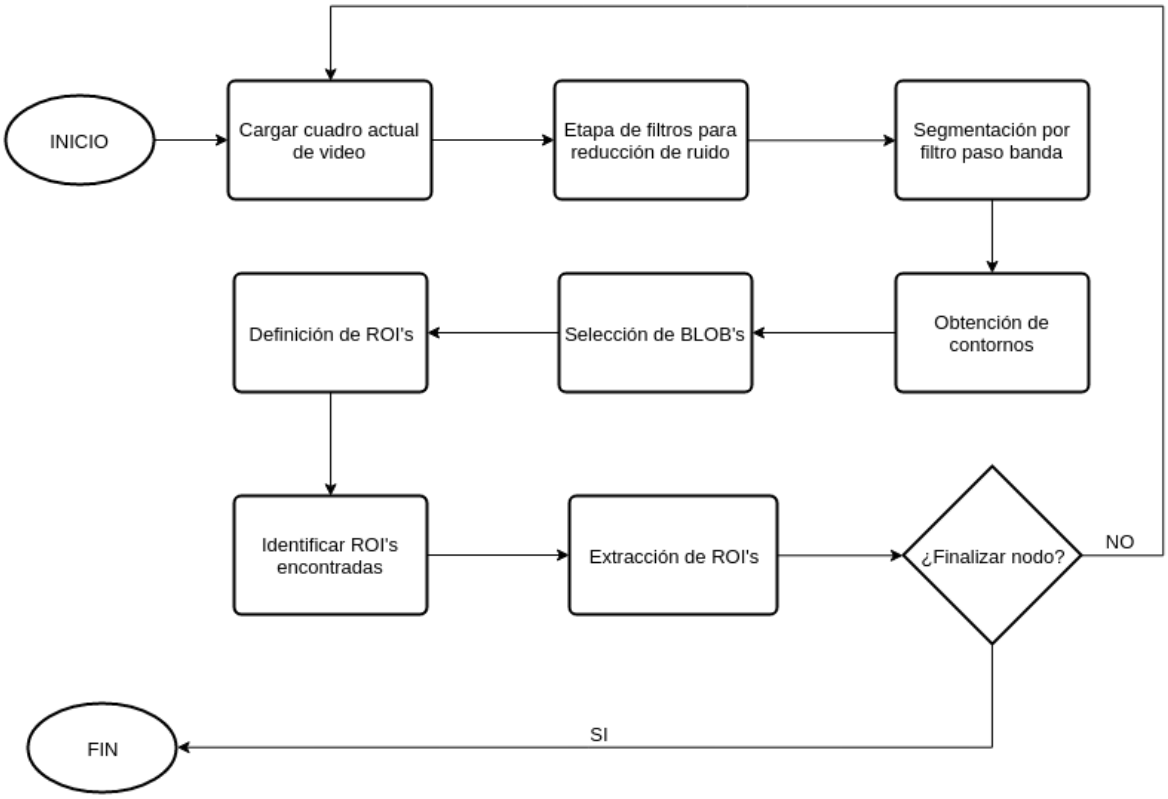


Figura 5.14 Diagrama de flujo del algoritmo para detectar y extraer las regiones de interés

Una de las principales diferencias con el programa de segmentación de imagen por gradientes de temperatura, es que se mantiene un tamaño fijo de la región a extraer, esto es porque al hacer dinámico el tamaño se puede ver afectadas todas las operaciones que puedan llegarse a aplicar.

Algoritmo para la búsqueda y extracción de regiones de interés

El algoritmo tiene una estructura inicial igual que la aplicación del subcapítulo 5.2 hasta la obtención de los blobs, en adelante se realizan las siguientes etapas:

Selección de blob's*;* una vez que se definen los blob's se seleccionan ordenan y seleccionan los 3 más grandes, considerando que en el campo de visión de la cámara se tiene alguna de las partes del cuerpo en la que es posible detectar el pulso.

Definición de ROI*;* a partir de los 3 blob's detectados se definen las regiones de interés, definiendo una ventana de tamaño fijo a partir del cálculo de los centroides de dichas regiones.

Identificar ROI*;* definidas las regiones, en la imagen original se señalan mediante un rectángulo, identificando cada una de ellas para su posterior procesamiento.

Extracción de ROI*;* utilizando las herramientas de ROS, se extraen las regiones de imágen detectada para ser publicadas como mensajes de tipo imagen a través de canales de comunicación identificados como $/roi\_detected/roi\_n,$ donde n es el número que identifica a cada una de las regiones.

El programa de implementación de esta aplicación se encuentra en el anexo K, para cumplir el propósito de este nodo es necesario tener dentro del campo de visión alguna de las partes objetivo mencionadas.

5.6 Detección de paredes arteriales

La presión más elevada que se presenta en el sistema circulatorio tiene lugar en las arterias, varía entre el pico producido por la contracción cardíaca (presión sistólica) y el mínimo que se tiene entre dos contracciones (presión diastólica), estas variaciones son perceptibles en las arterias al expandirse y regresar a su tamaño normal debido a las propiedades elásticas que tienen sus paredes [25].

Existen trabajos de procesamiento de imágenes donde tratan de percibir la expansión de las arterias más cercanas a la superficie de la piel, para detectar y medir el pulso cardíaco, como la aplicación en un teléfono inteligente desarrollada por Sungjun Kwon, et. al. [26].

Al tener detectadas varias regiones de interés que puedan contener parte de alguna arteria, el cambio de tamaño de sus paredes podría ser detectado al resaltar los bordes de la región de imagen que comprenden, hipótesis que se sustenta por la sensibilidad de la cámara infrarroja y que teóricamente podría detectar el pulso cardíaco de una víctima, por ello se implementó un detector de bordes Canny en un nodo de ROS. A continuación se describe brevemente el algoritmo de este detector.

5.6.1 Detector de bordes Canny

Los bordes de una imagen se definen por aquellos píxeles que se encuentran en el borde de las estructuras de los cuerpos y detalles en una imagen. Para su detección en 1986 John F. Canny desarrolló un algoritmo que logra definir en un intervalo varios bordes, para ello comprende de múltiples etapas para reducir la cantidad de información a procesar [19]. Las etapas de las que consiste su algoritmo son [21]:

***Reducción de ruido**;* primero se aplica un filtro gaussiano para reducir la cantidad de ruido que pueda tener la imagen fuente a trabajar, un tamaño de núcleo comúnmente utilizado al aplicar este filtro en el detector es de 5 x 5.

***Búsqueda de gradientes**;* un borde en una imagen puede tener múltiples direcciones, conscientes de ello se aplica cuatro filtros para detectar bordes horizontales, verticales y diagonales en la imagen suavizada. Para esto se aplica un filtro Sobel de dimensión 3 x 3 en las direcciones citadas, expresado matemáticamente como [21]:

$$G = \sqrt{G_x^2 + G_y^2} \tag{5.5}$$

$$\theta = tg^{-1}\left(\frac{G_y}{G_x}\right) \tag{5.6}$$

Donde G es la magnitud del gradiente de borde, también conocido como el operador de detección de bordes, G_x es la magnitud del gradiente horizontal, G_y la del gradiente vertical y θ el ángulo de dirección. Los gradientes son resultado de un filtro Sobel, que proporciona la primera derivada, orientando los bordes de forma perpendicular.

***Supresión de máximos**;* después de obtener la magnitud y dirección del gradiente de borde, se comienzan a suprimir los valores no máximos que se hace buscando los cruces por cero en la segunda derivada direccional de la imagen filtrada por la convolución realizada a partir del núcleo gaussiano, esta segunda etapa de derivación es tomada con respecto a la orientación del borde que fue calculado en la anterior etapa.

***Umbralizado de bordes por histéresis**;* en esta etapa se determinan los bordes potenciales detectados, a partir de los puntos calculados. Para hacerlo utiliza dos intensidades de píxel para conectar los puntos calculados y formar los contornos, el valor de mayor intensidad define todos los puntos internos de la estructura de un cuerpo, es decir forman los bordes internos y el valor de menor intensidad se utiliza para conectar todos los puntos que definen el perímetro de la estructura.

***Supresión de bordes débiles**;* finalmente se suprimen todos aquellos bordes que no se lograron conectar con los definidos por la umbralización con histéresis.

En la figura 5.15 se observa el diagrama de flujo del programa realizado para la detectar la expansión de la pared arterial, aplicando el algoritmo de detección de bordes Canny al cual se le aplicó un método estadístico para definir los valores límite del umbral de bordes por histéresis además de hacer una modificación que se explicará a continuación, para eliminar todos aquellos bordes que no son de nuestro interés.

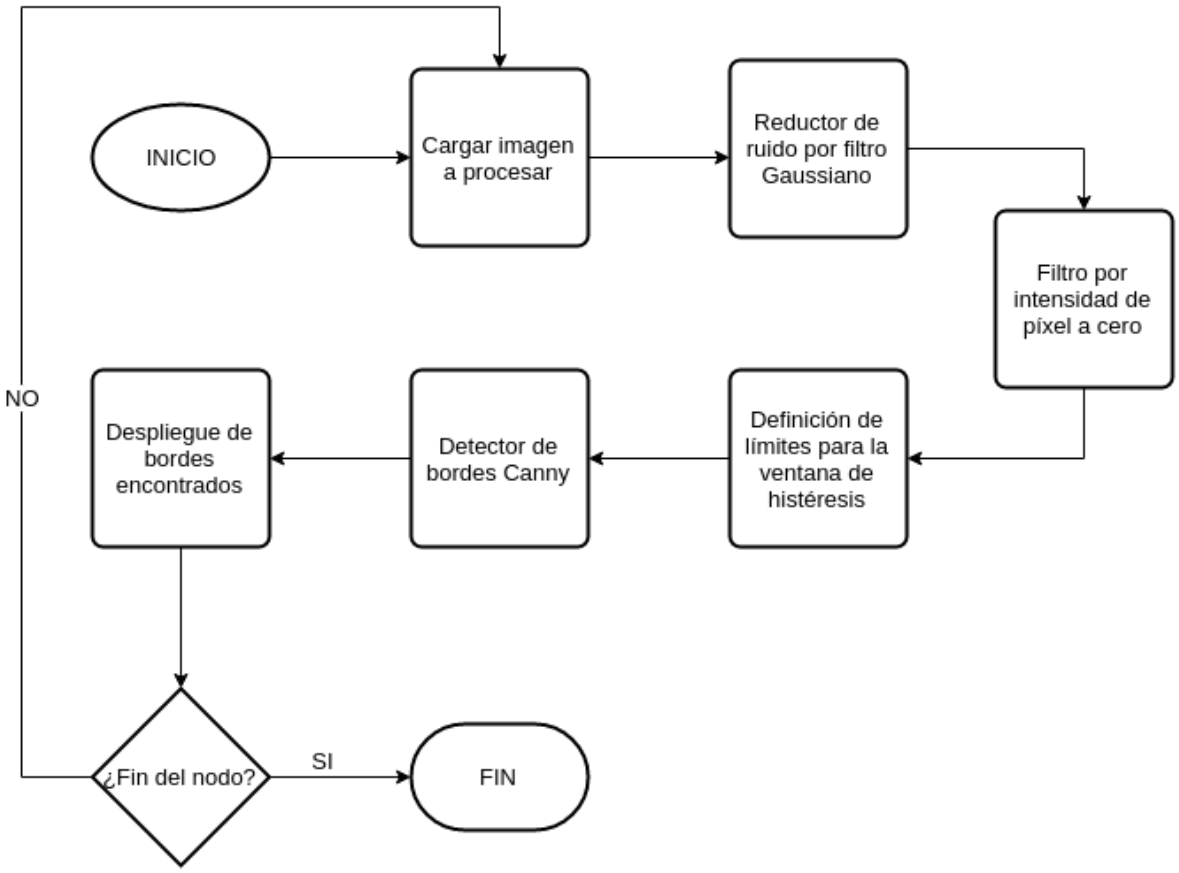


Figura 5.15 Diagrama de flujo del algoritmo de detección del borde de las arterias

A diferencia de las demás aplicaciones presentadas, este nodo se suscribe a los canales de comunicación (topics) que contienen las regiones de interés publicadas, con la finalidad de analizar específicamente las regiones de imagen que pueden contener parte de alguna arteria y llegar a detectar el cambio de tamaño debido a los cambios de presión. Las etapas del algoritmo implementado en el nodo se describen a continuación:

Cargar imagen*;* las imágenes que se procesarán en este nodo son las regiones de interés publicadas por la aplicación de búsqueda y extracción de ROI's. Se recibirán los cuadros del video siempre y cuando el canal de comunicación este activo, es decir que se esté ejecutando la aplicación que publica los mensajes de imagen de las regiones de inteŕes.

***Reductor de ruido por filtro gaussiano**;* a pesar de que el OpenCV comprende todas las fases del algoritmo de detección de bordes Canny, en diferentes fuentes consultadas [10] [19] se recomienda aplicar previamente un suavizado gaussiano para obtener mejores resultados, por ello se aplicó este filtro para comprobar la recomendación utilizando un núcleo de 5 x 5.

***Filtro por intensidad de píxel a cero**;* en esta etapa a la imagen filtrada se le aplica un selector de píxeles para trabajar únicamente con aquellos que tengan un valor superior al valor establecido, descrito matemáticamente como la ecuación 5.7.

$$thresh\ (x,\ y)\ =\ \begin{cases} value & if\ src\ (x, y) > intensity\ limit \\ 0 & otherwise \end{cases} \tag{5.7}$$

***Definición de límites de la ventana de histéresis**;* se calcula la mediana del valor de intensidad de los píxeles del cuadro en proceso, para definir los límites del umbral para definir los bordes por medio de porcentajes proporcionales a un valor establecido, el propósito de definir de esta manera los límites es que se ajusten acorde al cuadro de video que se esté trabajando.

***Detector de bordes Canny**;* con los límites de la ventana de histéresis obtenidos, se aplica el algoritmo de detección de Canny utilizando el método de la biblioteca, aplicado no a la imagen original, sino a la que se obtuvo al aplicar el filtro por intensidad de píxel a cero.

***Desplegar bordes encontrados**;* los bordes definidos se muestran en una ventana sobre la imagen original, es decir, en alguna de las regiones de interés extraída para visualizar los cambios del tamaño de contorno resaltados.

El programa que se realizó para implementar esta aplicación se encuentra en el anexo L, y los resultados de su aplicación se despliegan en el siguiente capítulo.

5.7 Extracción y transmisión de audio

La cámara web logitech C920 cuenta con un micrófono estéreo con un filtro reductor de ruido ambiente. ROS cuenta con paquetes que permiten extraer el audio de micrófonos compatibles con el sistema operativo bajo el que esté corriendo, por lo que se propuso implementar nodos para adquirir y transmitir el audio que proporciona el micrófono estéreo de la cámara web, con el objetivo de poder percibir algún llamado de auxilio o indicio para detectar a una víctima.

El paquete de ROS para trabajar con dispositivos de audio de entrada y salida se llama *audio_common*, el cual interactua con la Arquitectura Avanzada de Sonido para Linux (ALSA por sus siglas en inglés) la cual forma parte del núcleo de la mayoría de las distribuciones de

linux y también con la biblioteca gstreamer que al igual que ALSA, se integra al núcleo del sistema operativo.

Por medio de las diferentes herramientas de ROS, se puede configurar la frecuencia de muestreo del audio, configurar el número de canales de dispositivos de entrada y salida e incluso grabar la señal. A partir de los ejemplos que se proveen en la documentación del paquete, se realizaron modificaciones al nodo publicador que se pone como ejemplo para poder adquirir la señal del micrófono estéreo de la cámara y por supuesto transmitirlo al módulo de búsqueda para ser reproducido en los altavoces del módulo de procesamiento y en dado caso grabarlo. En el anexo M se encuentra el programa que se modificó para publicar la señal de audio, así como el código del nodo receptor y la descripción para grabar en archivos rosbag.

5.8 Resumen

Para la detección de víctimas no superficiales se propuso trabajar con el procesamiento de imágenes obtenidas de una cámara infrarroja y adicionalmente se incorporó al sistema de búsqueda una cámara web para utilizar la información que nos proporciona de audio y video con la finalidad de poder respaldar los resultados obtenidos de las aplicaciones propuestas.

Previamente a la descripción de los métodos de búsqueda, se explicó el funcionamiento de los nodos que inician las cámaras de video, los cuales difieren por el formato en el que se maneja la información y por los protocolos de comunicación, además transmiten los videos al módulo de procesamiento para ser visualizados y analizados por diferentes aplicaciones siempre y cuando se suscriban al canal de comunicación que porta los mensajes.

Se implementó una serie de nodos para representar el video monocromático de la cámara infrarroja con paletas de colores personalizadas en el espacio RGB, con el objetivo de hacer más apreciables a nuestra vista los gradientes de temperatura de los objetos que estén en el campo de visión de la cámara.

Posteriormente como primera aplicación de detección, se presenta un algoritmo para detectar fuentes de calor, segmentado la imagen en función de los gradientes de temperatura, con el cual por inspección visual permite reconocer las siluetas de todos los cuerpos dentro del campo de visión, clasificados por la intensidad de sus píxeles por medio del cual se extraen regiones de la imagen, sin importar que el el cuerpo presente algún tipo de movimiento.

Al poder detectar cualquier fuente de calor, se entrenaron unos clasificadores en cascada Haar para la detección de formas antropomórficas, siendo de nuestro interés la identificación del cuerpo completo de una persona recostada, brazos, manos y cabeza en estado de reposo, debido a que algunas partes de estas extremidades se tienen arterias cercanas a la superficie de la piel, las cuales son resaltadas por la cámara debido a su variación de temperatura debida al flujo de sangre oxigenada.

Para detectar el pulso cardíaco se abordaron dos aplicaciones, la primera de ellas se encarga de extraer las regiones de imagen que potencialmente pueden contener parte de alguna de las arterias superficiales en alguna de las partes del cuerpo mencionadas en el párrafo anterior, con el objetivo de analizar en la segunda aplicación la expansión y contracción de las paredes de estos conductos al circular la sangre.

En el siguiente capítulo se muestra el reporte de los resultados obtenidos de la implementación de las aplicaciones propuestas en diferentes pruebas.

Capítulo 6

Resultados de las pruebas de implementación

En este capítulo se mostrarán los resultados obtenidos de la implementación del sistema de detección de víctimas no superficiales, en una serie de pruebas diseñadas para analizar el funcionamiento de la metodología propuesta de búsqueda.

Se simularon circunstancias y condiciones para probar las cualidades de cada cámara, así como algunos posibles escenarios que se pueden presentar al buscar en condiciones reales una vez que el sistema haya sido incorporado a un robot de búsqueda, de ser esto factible. Esto se hizo con la finalidad de analizar el desempeño de cada aplicación, la viabilidad de la propuesta realizada en este trabajo así como de sus alcances, además de encontrar los puntos de mejora y corrección para que en un futuro pueda ser probado en condiciones más cercanas a la realidad, como en los lugares de entrenamiento de perros de búsqueda.

Todas las pruebas se realizaron con personas de diferentes fisonomías, para representar a las víctimas y probar los clasificadores en cascada Haar que se entrenaron, con el objetivo de buscar detectar el pulso, observando la expansión y contracción de la pared de arterias superficiales. Para hacer las tareas del módulo de búsqueda se utilizó en todas las pruebas una computadora portátil con procesador intel core i3 con cuatro núcleos de tercera generación y 4Gb de memoria RAM, y para el módulo de búsqueda se utilizó otra computadora portátil con procesador intel core i5 de sexta generación, con cuatro núcleos y 8Gb de memoria RAM.

La presentación de los resultados de cada prueba es contextualizada con una descripción del ambiente bajo el cual se realizó, incluyendo en algunos casos información sobre el consumo de recursos computacionales, para poder caracterizar y posteriormente migrar el módulo de búsqueda al hardware más adecuado, para integrarlo a un robot o módulo más compacto como parte del trabajo a futuro que se describe en el siguiente capítulo.

6.1 Detección de personas con un observador

La implementación básica del sistema de detección, consiste en la transmisión de ambas cámaras y de la señal acústica del micrófono estéreo, al módulo de búsqueda donde un operador acorde a su criterio, puede buscar las víctimas al visualizar en los videos y escuchar la señal de audio.

En el arranque del sistema de detección, se inicializan los nodos de extracción de datos de las cámaras y de los micrófonos que están conectados en el módulo de búsqueda y envía la información al módulo de procesamiento, donde a la vez se inician los nodos que permiten visualizar el video original recibido. La iniciación de las aplicaciones citadas, se hace ejecutando el archivo launch $start\_up.launch$ que se encuentra dentro del paquete $start\_sys\_nsv$, creado específicamente para contener diferentes archivos launch.

La primer prueba consistió en probar la detección básica a cargo de un observador, con el objetivo de analizar el funcionamiento de ambos módulos del sistema, poniendo atención específicamente en algunas características especiales de cada equipo, como el autoajuste de escala de grises de la cámara FLIR A35 ante diferentes fuentes de calor que estuviesen dentro de su campo de visión, tanto en su formato original como en las cinco representaciones en el espacio de colores RGB con las tonalidades personalizadas mostradas en la sección 5.2. En lo que respecta a la cámara web logitech C920, se propuso probar su balance automático de blancos, auto enfoque, la calidad de la imagen proporcionada bajo diferentes condiciones de iluminación y el uso de su micrófono.

A continuación se muestran un conjunto de imágenes, capturadas en diferentes condiciones para analizar el funcionamiento de las cámaras, teniendo dentro de su campo de visión a una persona y todos los objetos que están a su alrededor. Todas las imágenes fueron adquiridas por las cámaras en lugares donde se tenían diferentes temperaturas ambiente (en el intervalo

de 20 a 25 °C) y también con distintas condiciones de iluminación. Cada uno de los nodos de visualización de video permite guardar imágenes en diferentes carpetas, organizadas en el sistema, para hacerlo solo se requiere presionar la tecla s siempre y cuando esté seleccionada la ventana de la transmisión.

Se sometieron a prueba las funciones de balance de blancos y auto enfoque de la cámara C920, monitoreando su comportamiento al tener dentro de su campo de visión a una persona a diferentes distancias y condiciones de iluminación. En la figura 6.1 se muestran algunas de las imágenes capturadas directamente de la transmisión del video, al ejecutar el nodo de ROS ***video_web_publisher*** en cuatro lugares diferentes simulando víctimas.

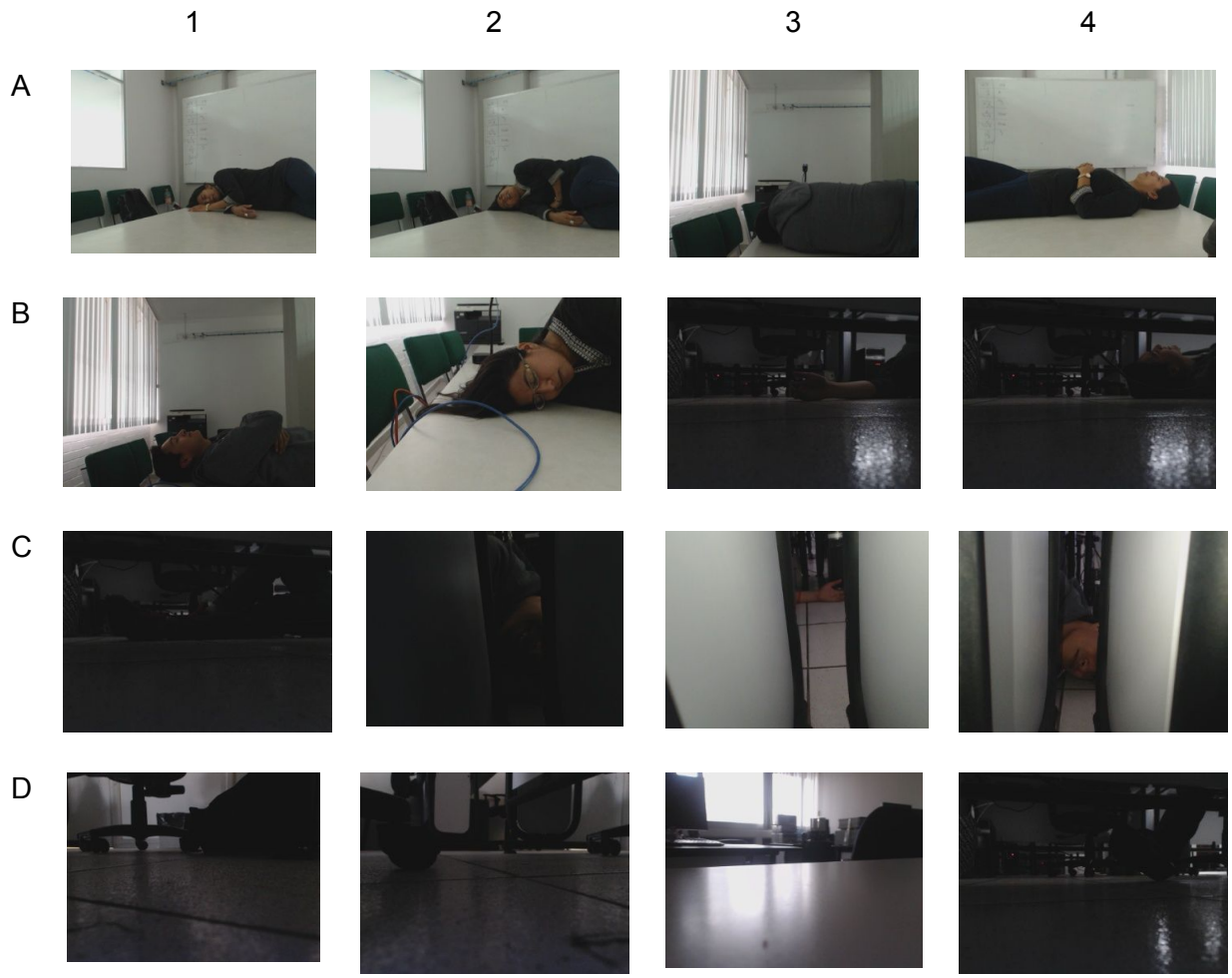


Figura 6.1 Imágenes capturadas por la cámara web C920. Se nomeclatura las filas y las columnas para ubicar y referirse a cada una de ellas.

El autoenfoque y balance automático de blancos en niveles bajos a moderados de iluminación combinado con la resolución de la cámara permiten tener una visibilidad aceptable que permite identificar por observación la mayor parte de las cosas a cuadro (por ejemplo, imágenes de la

fila A, B1 y B2 de la figura 6.1). Sin embargo, en en cierta penumbra no se tiene claridad en la imagen como era de esperarse, pero al incluir una fuente de iluminación artificial (luz blanca proveniente del led ultrabrillante de un smartphone) se resuelve de forma parcial como se puede observar en las imágenes C3 y C4. Bajo condiciones de iluminación normal, la cámara no tiene problemas, sin embargo es poco probable que al buscar en una zona de emergencia se cuente con esas condiciones.

En cuanto respecta a la cámara infrarroja, en la figura 6.2 se muestra un conjunto de imágenes capturadas bajo las mismas condiciones en las que se adquirieron las de la figura 6.1. De la misma forma la mayoría de las imágenes son de personas simulando posiciones en las que se pueden encontrar víctimas.

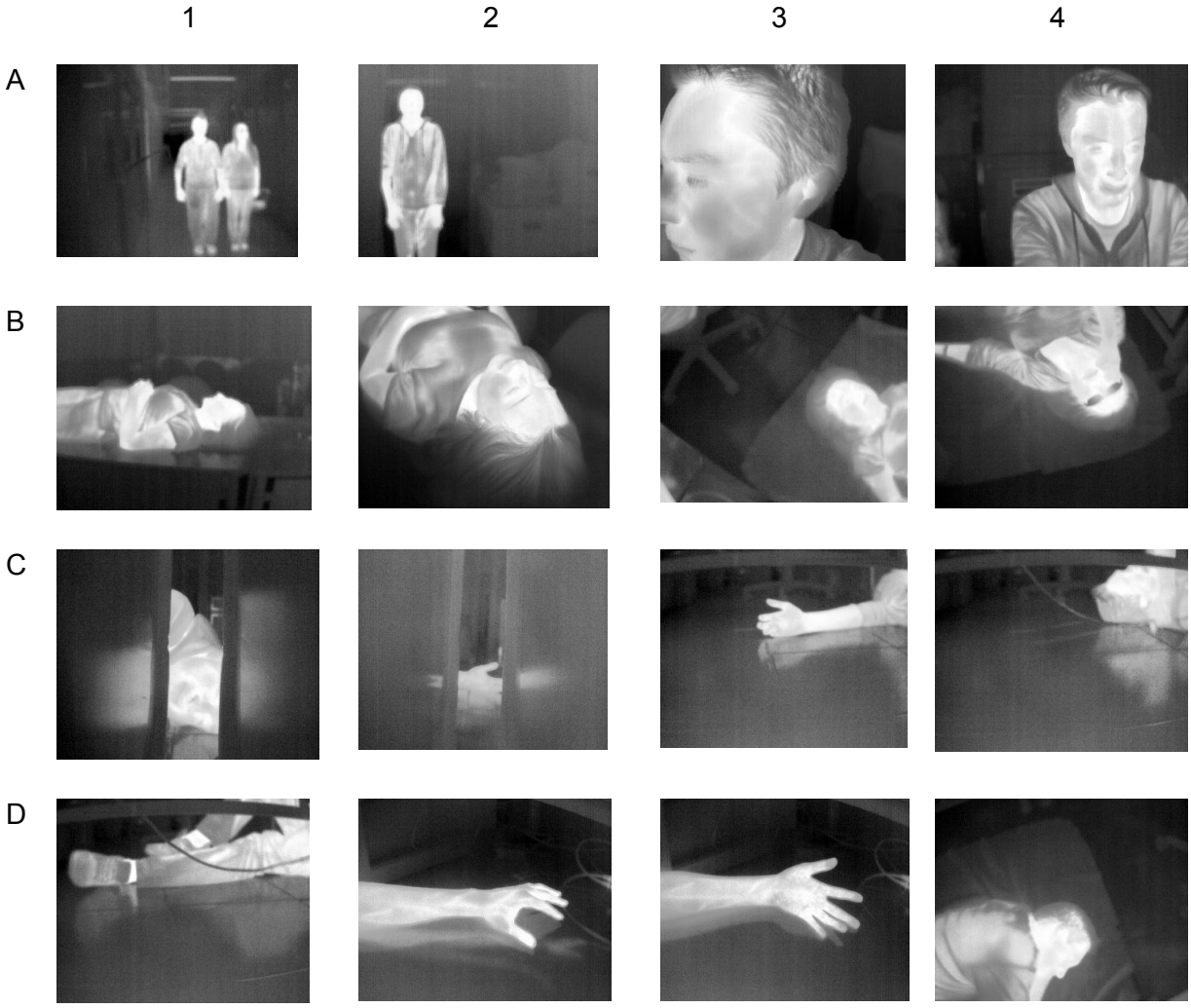


Figura 6.2 Imágenes de víctimas en diferentes ángulos y distancias, obtenidas de la transmisión de video de la cámara FLIR A35

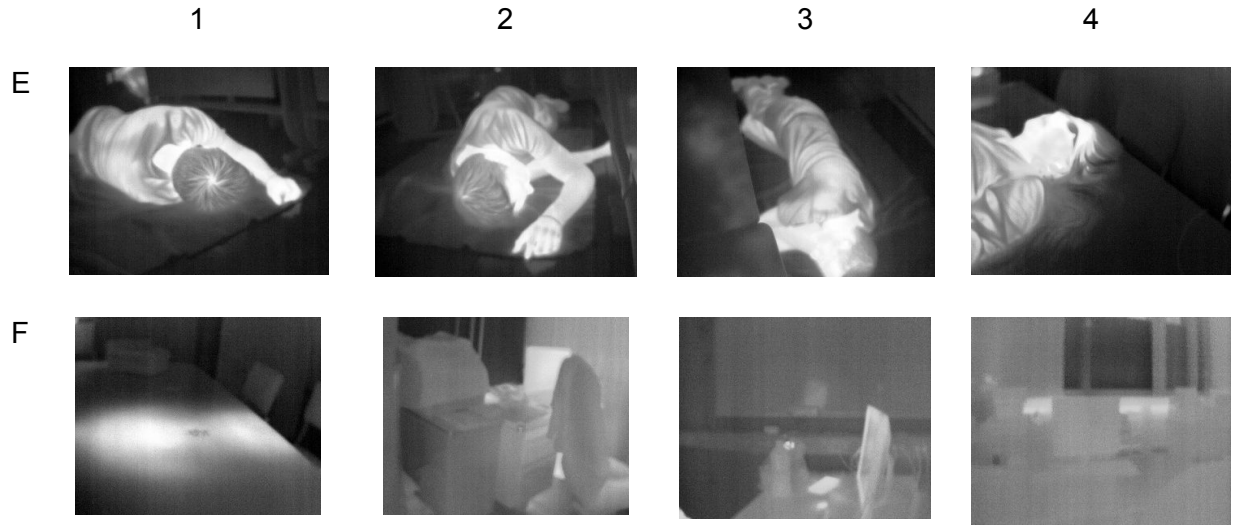


Figura 6.2 (continuación) Imágenes de víctimas en diferentes ángulos y distancias, obtenidas de la transmisión de video de la cámara FLIR A35

Como se puede observar en la secuencia de imágenes presentada, el auto ajuste de tonos hace resaltar la figura de la persona además de proporcionar detalles al encontrarse a un distancia menor a un metro de la persona, esto es posible por la sensibilidad del equipo que permite detectar gradientes térmicos del orden de 0.05 °C.

Notablemente todas las fuentes de calor que están en el cuadro de visión, son definidas por los tonos más altos de gris en la imagen, sin importar las condiciones de iluminación. En las imágenes A3, D2 y D3 de la figura 6.2 se observan a simple vista las arterias superficiales en el brazo, la mano y la sien por la variación de temperatura que tienen con respecto a la piel, la lente se colocó a una distancia promedio de 50 centímetros para adquirir las imágenes mencionadas.

En algunas capturas de la figura 6.2 se observa que la radiación infrarroja que emiten los cuerpos, es reflejada en cierto tipo de superficies representada, con niveles de tonos similares a la propia fuente. También, conforme se incrementa la distancia entre la cámara y la fuente de calor, todos los píxeles que lo representan en la imagen tienden a tener un tono promedio, la imagen A1 de la figura 6.2 fue tomada a tres metros de distancia, se observa que los detalles de la cara se pierden pero se mantiene la diferencia de tonos entre las partes descubiertas del cuerpo y la ropa.

Los dos puntos citados en el párrafo anterior, hacen variable la distinción de personas por simple observación del video de la cámara infrarroja en su formato original, por eso se adquirió el conjunto de imágenes de las figuras 6.3 ejecutando los nodos que transforman al espacio de colores RGB con las diferentes paletas mostradas en el capítulo 5.

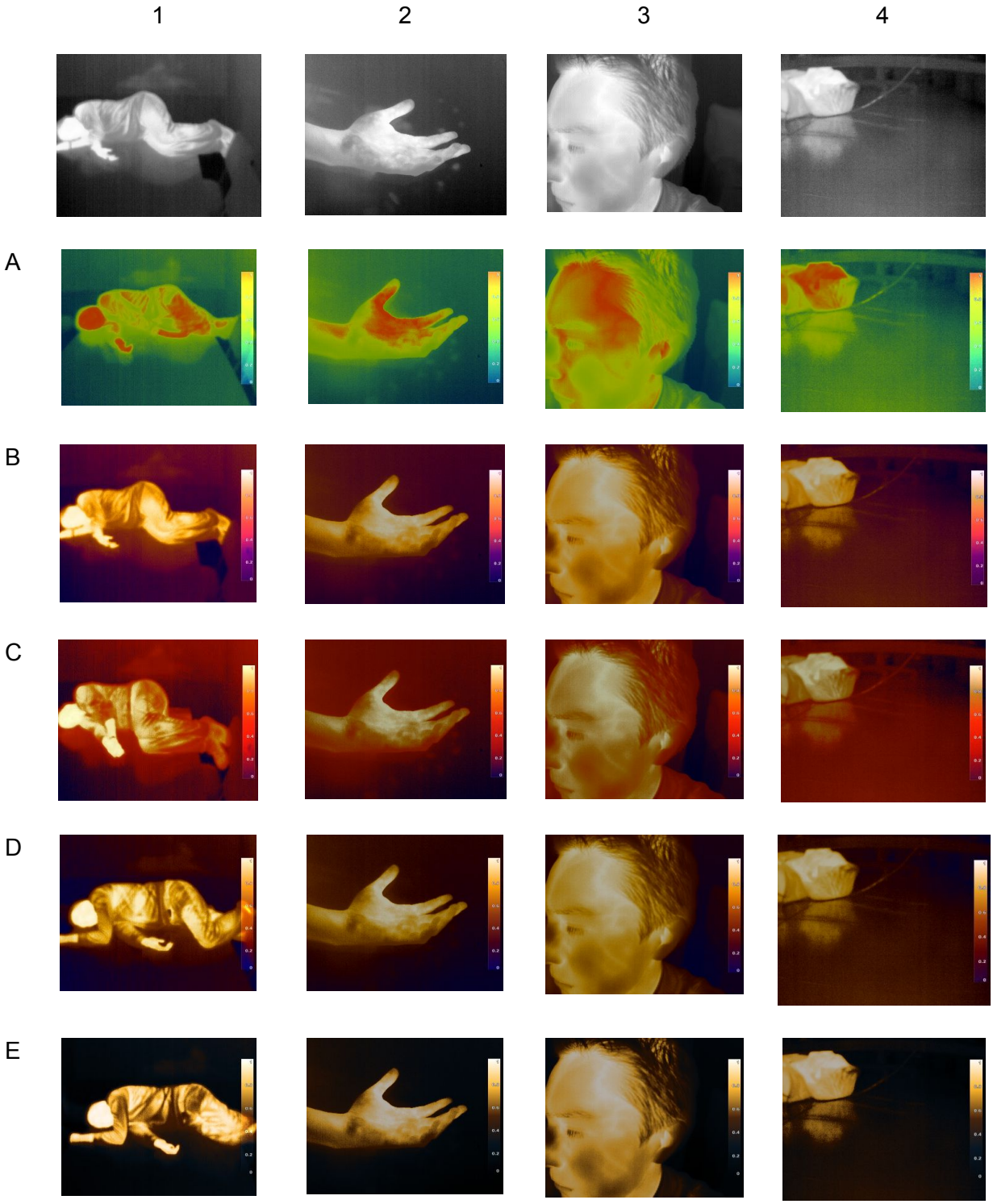


Figura 6.3 Conjunto de imágenes infrarrojas representadas en el espacio de colores RGB con las diferentes escalas de colores

Como se puede observar, algunas paletas de color favorecen más la distinción de todos los objetos y cuerpos que estén en el campo de visión, por lo que acorde los objetivos de búsqueda, se puede seleccionar la mejor opción, por ejemplo las representaciones de colores que corresponden a las paletas D y E (cuyos nodos de aplicación son ***color_thermal_monitor_d.py*** y ***color_thermal_monitor_e.py***) resaltan más que las otras paletas los gradientes de temperatura de una fuente de calor, como el cuerpo humano, e incluso acorde a la matriz de tonos, oscurece todos aquellos elementos en la imagen que son representados por píxeles de baja intensidad. Sin embargo, en la imagen tres de la figura 6.3 en todas sus representaciones se puede distinguir la arteria que pasa por la sien, pero las paletas C, D y E definen su forma de una mejor manera.

Para determinar si el autoajuste de escala de grises de la cámara, afecta en el reconocimiento de personas por observación, se puso a cuadro una fuente de calor, con una temperatura superior a la promedio de los seres humanos junto con una persona y partes de su cuerpo. En la figura 6.4 se muestran imágenes de esta prueba.

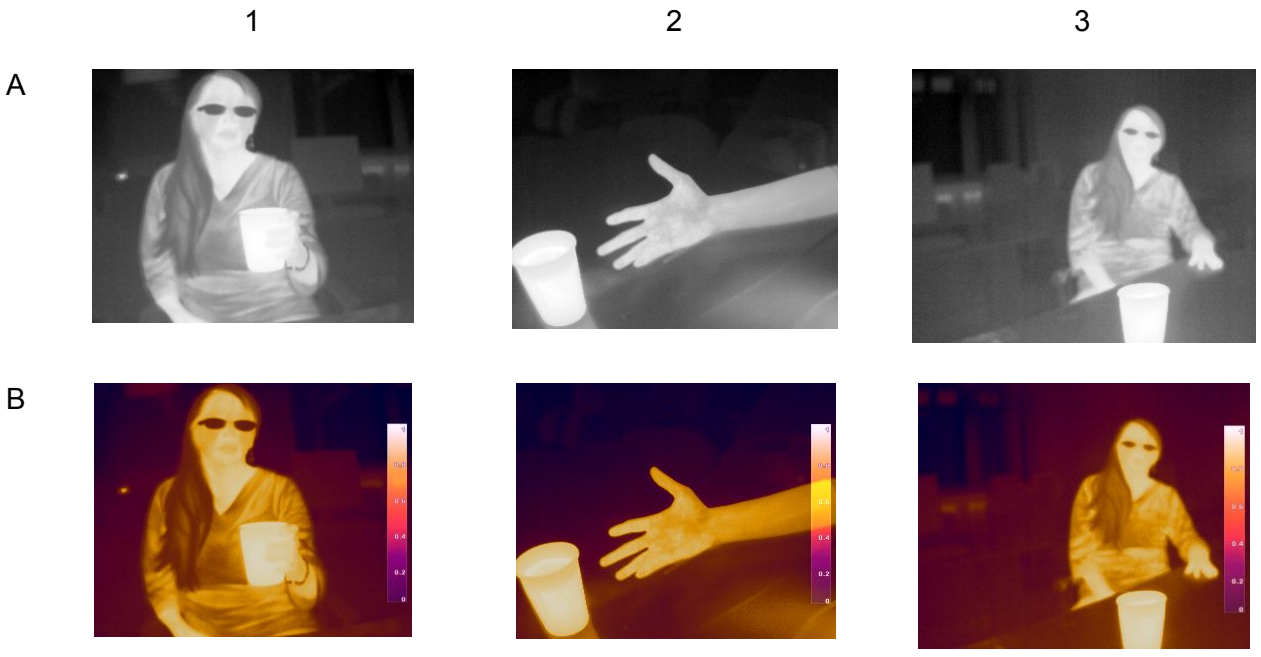


Figura 6.4 Imágenes de una fuente de calor junto con el cuerpo completo o partes de él de una persona

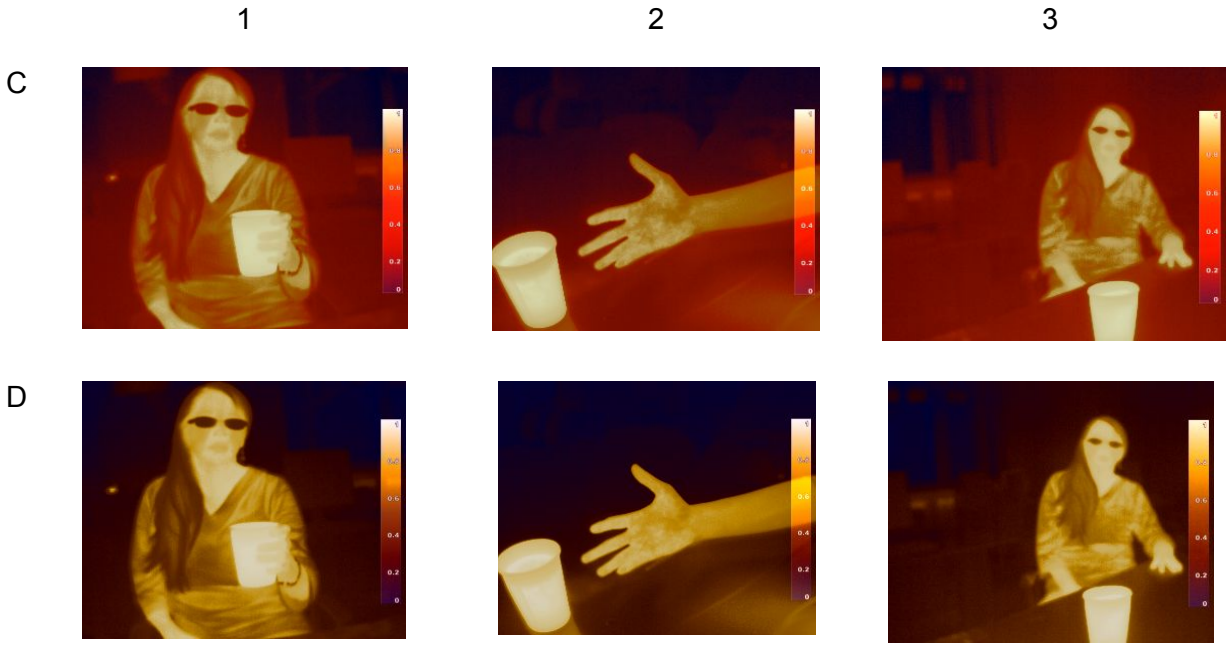


Figura 6.4 (continuación) Imágenes de una fuente de calor junto con el cuerpo completo o partes de él de una persona

Como se puede observar, los tonos más altos son asignados a aquellos píxeles que representan a la fuente de calor de mayor temperatura, que en este caso es un vaso con café caliente, a partir de ella se asignan los demás valores, por ello en las diferentes imágenes de la figura 6.4 se observa que el cuerpo humano o las partes de éste se distinguen ligeramente en menor proporción a simple vista, lo cual ocasiona que ciertos gradientes de temperatura de interés sean menos perceptibles.

Identificar víctimas por observación con estas representaciones de color, en conjunto con la configuración de autoajuste de la cámara es simple, siempre y cuando no se presenten otras fuentes de calor que tengan una mayor temperatura a la promedio del cuerpo humano. Ocupando las paletas de colores D y E con sus respectivos nodos, se filtra parte del fondo en el que se encuentra la fuente de calor, razón por la que se pueden considerar como base para crear otras paletas similares.

6.2 Segmentación de imágenes por gradientes de temperatura

La prueba para esta aplicación consistió en encontrar la forma de las fuentes de calor que la cámara infrarroja tuviese a cuadro y estimar su posición angular tomando como referencia la mitad del ancho de la imagen, esto se hizo con el objetivo de orientar al robot o móvil que transporte la cámara hacia esos puntos de interés y en dado caso el observador pueda determinar si se trata de alguna víctima y pueda acercar la cámara para obtener mayores detalles, siempre y cuando sea posible.

La posición angular es calculada obteniendo el ángulo entre dos vectores, el primero de ellos corresponde a un vector definido entre las coordenadas del centro de la imagen y el origen del plano definido en la mitad del ancho y el segundo se define entre el mismo origen y el centro de la región de imagen que corresponde a la fuente de calor diferenciada por los tonos más altos de la escala de gris. En la figura 6.5 se muestran los resultados de la ejecución del nodo de segmentación.

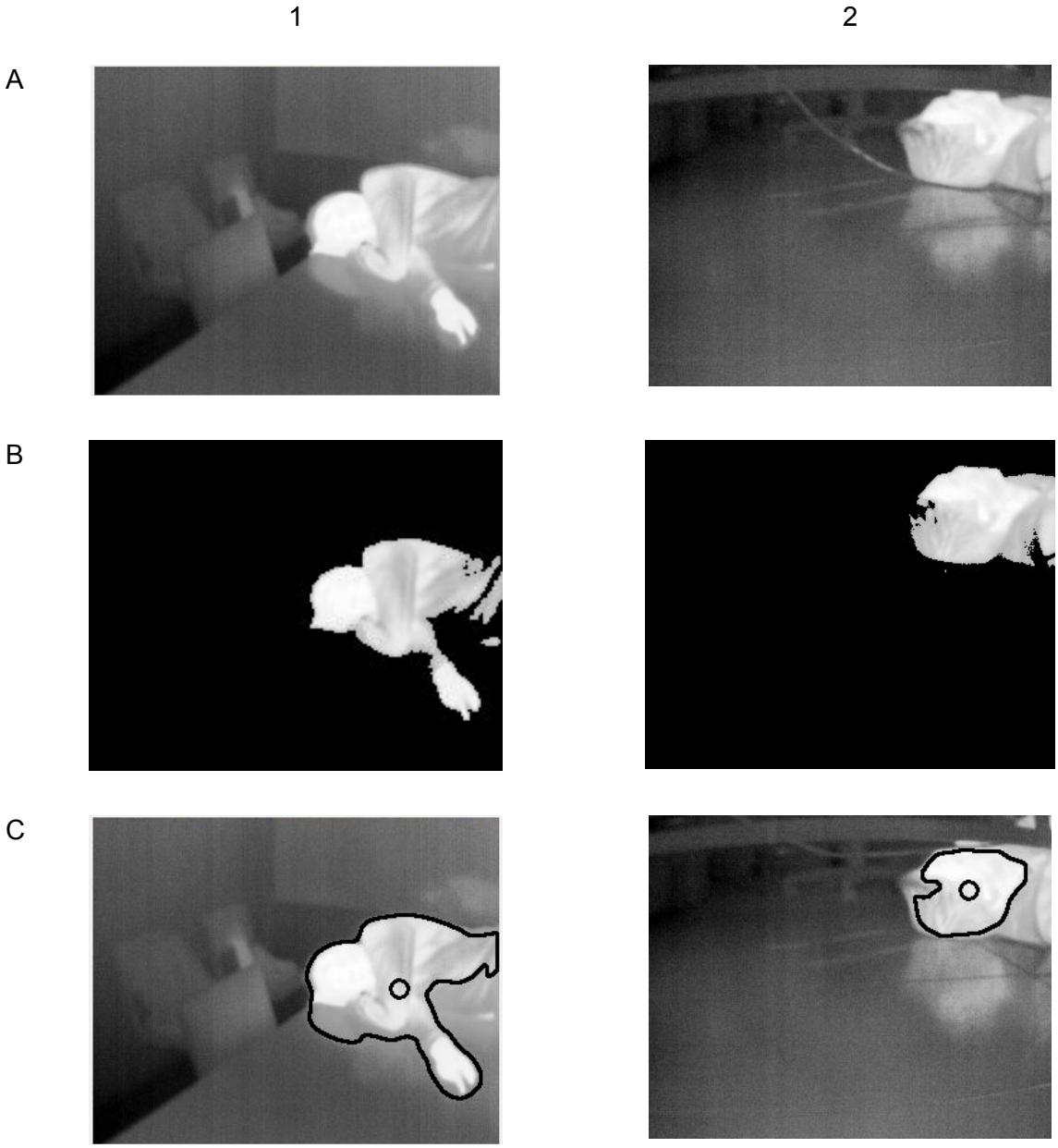


Figura 6.5 Resultados de la segmentación de imágenes térmicas y su ubicación angular

En la fila A se observa la imagen original adquirida por la cámara, sin procesamiento alguno, posteriormente en la fila B se observa el resultado del filtro paso banda que segmenta la imagen en regiones, definidas a partir de la intensidad de los píxeles en un intervalo que identifica los valores superiores a 180. Por último en la fila C se muestra el contorno definido y un círculo cuyo centro corresponde al centroide calculado de la región más grande que se definió por el filtro de umbral.

En la imagen 1 de la figura 6.5, se puede observar que por la posición de la víctima se define un contorno y región que es difícil de distinguir si solo se presentará el resultado del filtro paso banda de tonos de píxeles, por ello se tienen como auxiliar los nodos que transforman el espacio de color original a un RGB con los tonos personalizados.

La detección de víctimas auxiliada por un observador, demanda una cantidad considerable de memoria al estar adquiriendo y visualizando el video de ambas cámaras, además de que es poco confiable ya que depende de la habilidad y experiencia del operador, por eso se complementan con las otras aplicaciones, cuyos resultados se muestran el las siguientes secciones.

6.3 Búsqueda de víctimas por clasificadores Haar

Como se mencionó en el capítulo 5, se entrenaron tres clasificadores en cascada para detectar cuerpo completo, manos y cabeza de una víctima. En la tabla 6.1 se muestran los parámetros que se configuraron para el entrenamiento de cada clasificador.

	Clasificador		
Parámetro	body_detector	hands_detector	head_detector
Cantidad de muestras positivas	580	980	720
Cantidad de muestras negativas	1500	2500	1800
Tamaño de muestras positivas (píxeles)	80 x 60	80 x 60	80 x 60
Presición deseada de detección	99.9 %	99.9 %	99.9 %
Cantidad de etapas definidas	10	10	10
Error mínimo por etapa	25%	25%	25%

Tabla 6.1 Parámetros de entrenamiento de los tres clasificadores Haar

Las muestras positivas fueron adquiridas considerando diferentes posturas y ángulos en los que se podría llegar a encontrar alguno de los objetivos de detección. El algoritmo de entrenamiento fue implementado utilizando las herramientas y métodos de la biblioteca OpenCV. Todos los bancos de imágenes se obtuvieron de cuatro personas, dos hombres y dos mujeres con rasgos físicos diferentes, no se consideraron más participantes porque otro de los objetivos de esta implementación es analizar la factibilidad de crear estos clasificadores con imágenes de la cámara infrarroja.

En la figura 6.6 se muestran algunas de las imágenes detectadas por los clasificadores entrenados, además de mostrar algunas de las falsas detecciones al implementarse el nodo que utiliza los clasificadores para buscar víctimas que fueron representadas por personas que no participaron en el banco de muestras positivas. Los resultados estadísticos obtenidos de cada clasificador se muestran en la tabla 6.2.

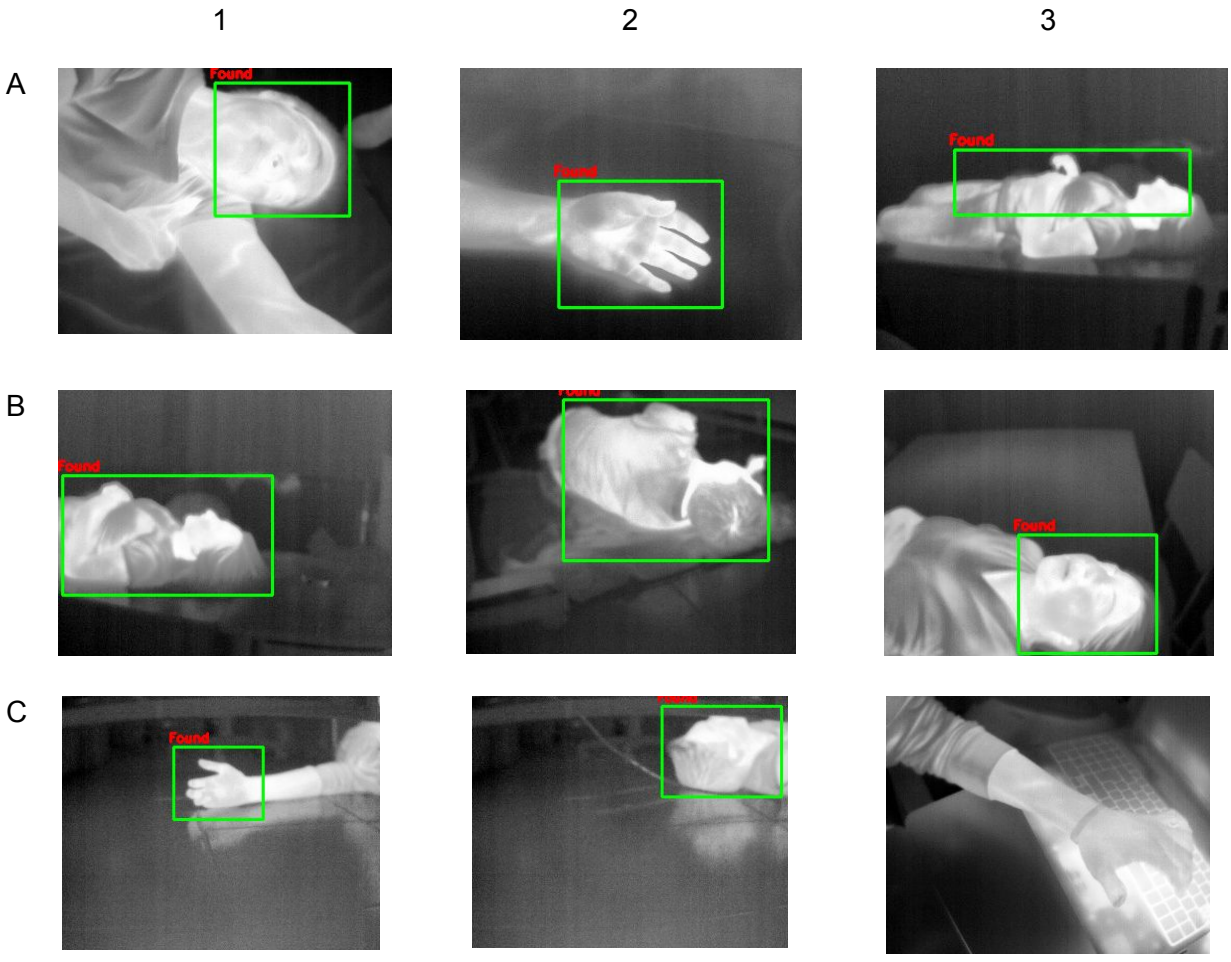


Figura 6.6 Detecciones realizadas por la implementación de los clasificadores Haar entrenados.

D

Figura 6.6 (continuación) Detecciones realizadas por la implementación de los clasificadores Haar entrenados

Clasificador	Tiempo de entrenamiento (horas)	Pruebas	Aciertos	Falsos negativos	Falsos Positivos	% de error
Hand_detector	12.5	50	38	7	5	24
Head_detector	10.8	50	32	6	12	36
Body_detector	9.5	50	27	5	7	46

Tabla 6.2 Resultados estadísticos obtenidos de la implementación de los clasificadores en cascada entrenados para la detección de víctimas no superficiales.

El entrenamiento se realizó en una computadora con procesador Intel core i7 de quinta generación de ocho núcleos y 8Gb de memoria RAM. Durante el entrenamiento todos los núcleos fueron ocupados junto con el 50% de la memoria.

Para su implementación se disminuyó la frecuencia de publicación de los cuadros de la cámara térmica a 15 Hz, puesto que al hacerlo al doble, la aplicación junto con el nodo publicador del video consumía cerca del 80 % del procesador.

6.4 Búsqueda de regiones de Interés

La última prueba realizada, se divide en dos partes implementando dos aplicaciones con el objetivo de poder detectar el pulso de una víctima a distancia. En primera instancia se tiene que buscar y localizar las regiones de interés que pueden contener parte de la representación gráfica de una arteria superficial. Para llevar a cabo esta búsqueda, se considera que se ha detectado la cabeza, manos o brazos y la cámara se encuentra a una distancia máxima de 80 centímetros, este dato fue obtenido por observación al realizar diferentes pruebas. Sin embargo, si no se tienen esas condiciones, aún así el programa funciona mientras en el campo de visión de la cámara exista algún objeto o cuerpo con una temperatura mayor a la del medio.

Los resultados del algoritmo propuesto, descrito en el capítulo anterior, se muestran en la figura 6.7, mostrando la detección de las arterias superficiales y otras partes del cuerpo que tienen diferencias de temperatura significativas con respecto a la piel.

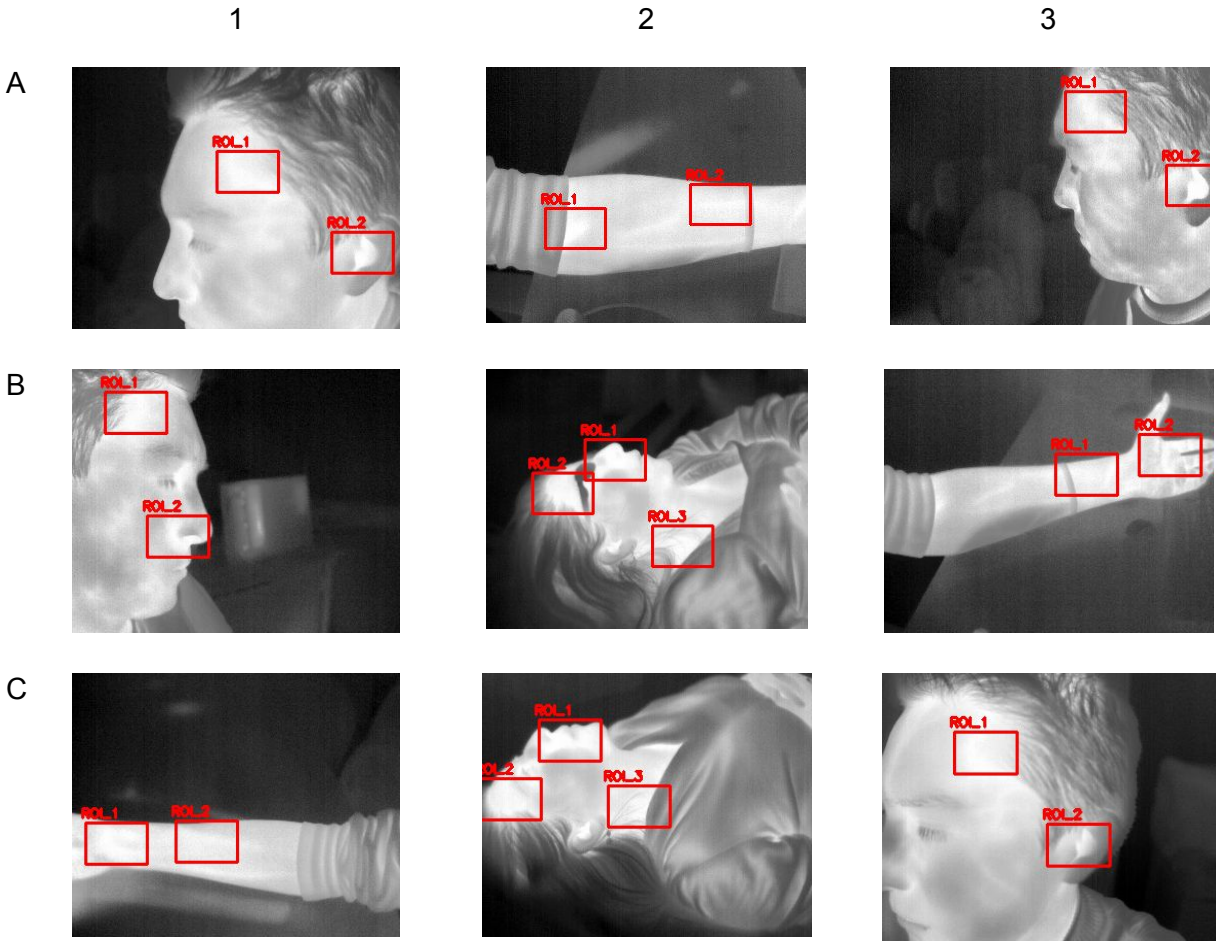


Figura 6.7 Imágenes resultado de la detección de regiones de interés en las zonas del cuerpo en donde se tienen arterias superficiales

Las regiones de interés detectadas coinciden con partes del cuerpo en donde se encuentran partes de arterias superficiales, como se había previsto al desarrollar la aplicación. Cada ROI detectada es publicada en un canal de comunicación, para que pueda ser procesada por otros nodos con diferentes algoritmos, como el que se propuso para detectar la contracción y expansión de la arteria.

En las imágenes B1, B2 y C2 de la figura 6.7, se definen regiones localizadas en la parte de la parte del rostro que comprende la boca y la nariz, y también en cierta parte del cuello. Estos puntos son normalmente utilizados para medir el pulso cardíaco mediante el tacto, lo que los convierte en candidatos de análisis tanto mediante procesamiento como por observación, como se aborda en la segunda fase de la prueba.

6.5 Detección y monitoreo de la pared arterial

Una vez que las regiones de interés hayan sido localizadas por la aplicación de la sección 6.4, se implementa el detector de bordes para encontrar el perímetro, que en principio define la parte de arteria contenida en alguna de las ROI, con el objetivo de presentar de forma visual la contracción y expansión que presentan las paredes arteriales, debidas al flujo sanguíneo. Por ello en conjunto a los detectores de bordes se integró otro nodo para representar en el espacio de colores RGB la ROI, personalizada para tener una mayor certeza de la detección del pulso.

En la figura 6.8 se muestra un par de secuencias de imágenes en donde se observan los resultados obtenidos de la implementación de las aplicaciones mencionadas en el párrafo anterior.

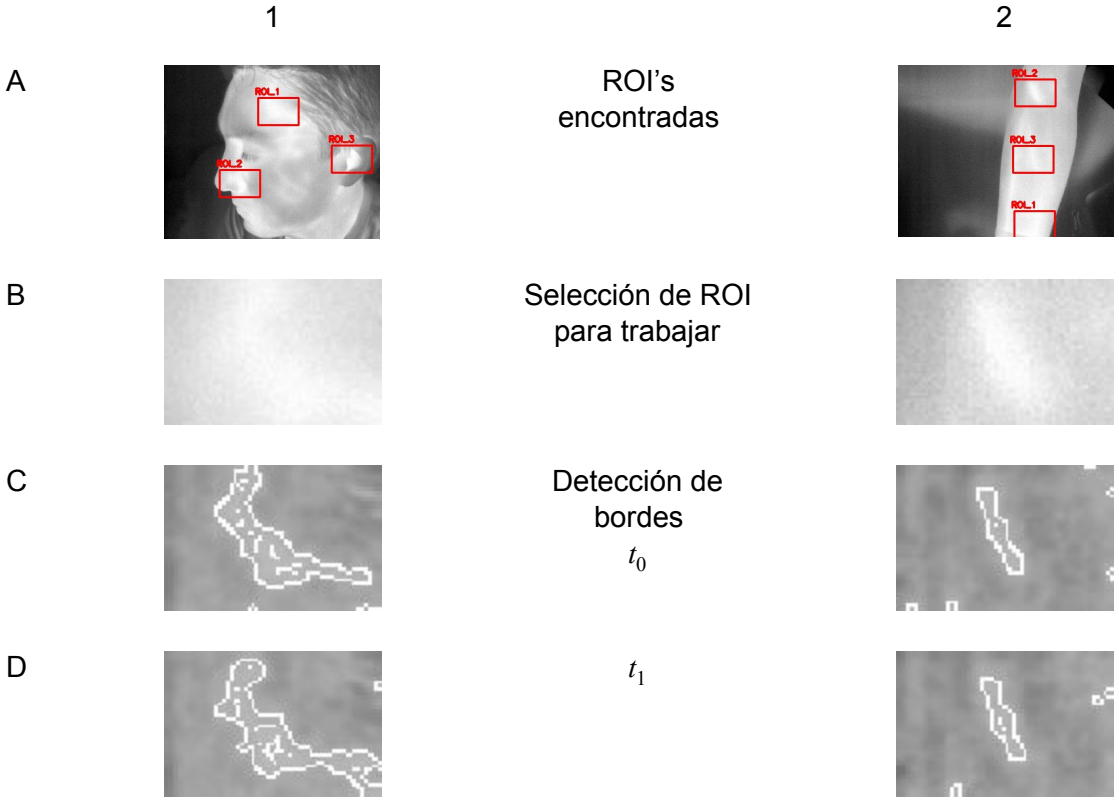


Figura 6.8 Secuencias de imágenes del análisis realizado por el detector de bordes canny para visualizar la expansión y contracción de las paredes arteriales.

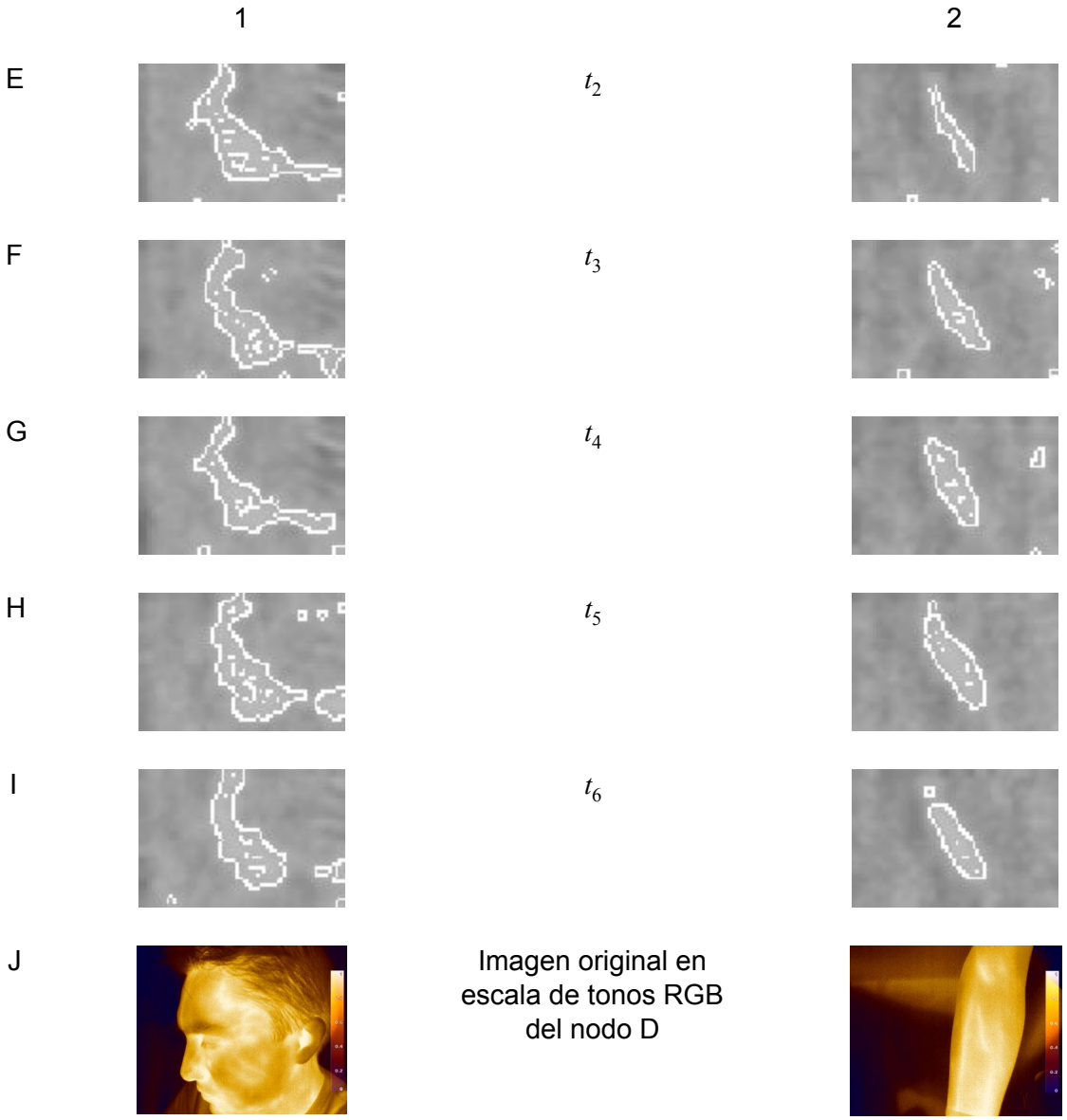


Figura 6.8 (continuación) Secuencias de imágenes del análisis realizado por el detector de bordes canny para visualizar la expansión y contracción de las paredes arteriales. Cada columna es una secuencia de capturas realizadas a la ROI mostrada en la posición B, extraída de la imagen original que se muestra en la fila A y que es representada en RGB por los tonos definidos por la escala D, en la fila J.

En las secuencias de imágenes mostradas en la figura 6.8, se observa que la región de imagen que define el segmento de arteria en cuadro, queda definido por el detector de bordes Canny aplicado. Además en las diferentes capturas se puede distinguir un cambio de tamaño y forma del contorno definido, dando la impresión de que esto corresponde al objetivo perseguido, que es la detección de la contracción y expansión de las paredes arteriales. Este comportamiento al analizarse en un lapso de tiempo de tres minutos, parece ser periódico y constante, por ello de

forma paralela se implementó un nodo con la transformación RGB con la escala de tonos D, para hacer más perceptible a nuestra vista dichos cambios, los cuales fueron contabilizados por un observador al realizarse esta prueba, con la colaboración de dos mujeres y tres hombres, obteniendo los resultados mostrados en la tabla 6.3.

	Pulso promedio (bpm)	Pulsaciones por minuto detectadas	Porcentaje de error %
Persona 1	85	75	11,76
Persona 2	72	65	9,72
Persona 3	80	72	10,00
Persona 4	75	68	9,33
Persona 5	78	70	10,26
		Error promedio	10,22

Tabla 6.3 Resultados estadísticos obtenidos de la observación del comportamiento de los contornos que definen las paredes arteriales

6.6 Metodología de búsqueda de víctimas no superficiales y detección del pulso cardíaco

Por último, después de todas las pruebas realizadas a cada una de las aplicaciones propuestas para la detección de víctimas no superficiales, se propone una metodología de búsqueda que tiene por objetivo aprovechar al máximo los algoritmos desarrollados. En la figura 6.10 se muestra un diagrama que representa de forma gráfica la metodología propuesta.

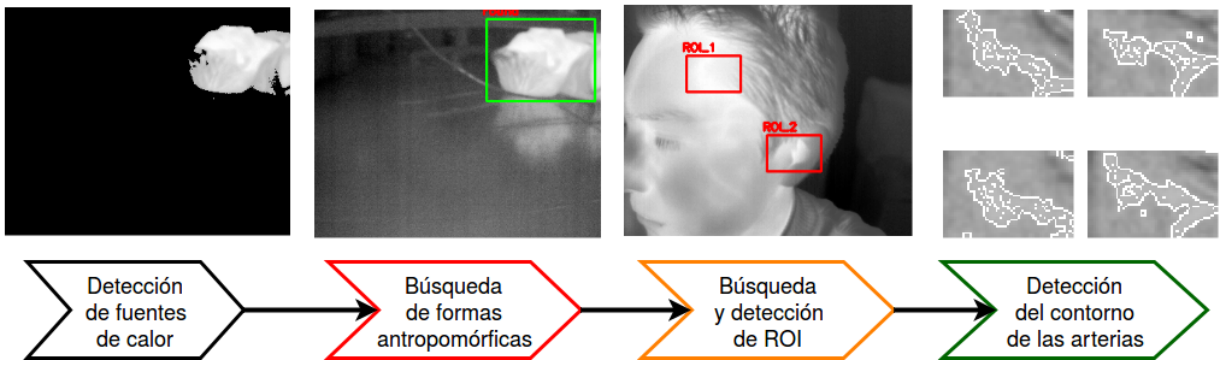


Figura 6.10 Metodología resultante propuesta para la implementación del sistema para detección de víctimas no superficiales

6.6.1 Método para la búsqueda de víctimas

Cada una de las etapas de la metodología de búsqueda propuesta en la figura 6.10, consiste de los siguientes pasos:

1. En primera instancia se trata de localizar todas aquellas fuentes de calor, ya sea por inspección a través de alguno de los nodos de monitoreo de video o por el programa de segmentación de imagen por gradientes de temperatura.

2. Con la posición angular de la fuente de calor que se identificó en el paso anterior, se orienta la cámara para tener el cuerpo u objeto lo más cercano a su centro de su campo de visión y se comienzan a buscar formas antropomórficas por medio de los clasificadores en cascada Haar.

3. Detectadas las partes del cuerpo en donde se podría estimar el pulso, se procede a seccionar la imagen extrayendo únicamente algunas partes rectangulares de esta, para ser analizados cada uno por separado.

4. Con las ROI disponibles, se aplica la detección de bordes para visualizar la presencia del pulso al cambiar de tamaño el contorno que rodea a la parte de arteria enfocada por la detección de regiones de interés que se hizo en el paso anterior.

Además de estos pasos definidos, es posible realizar alguna modificación de ejecución de las aplicaciones que se tienen disponibles, por ejemplo, mientras se sigue la metodología descrita se puede escuchar el audio que proporciona el micrófono estéreo, simultáneamente. El método propuesto es producto de los resultados obtenidos de las pruebas realizadas, teniendo en cuenta que el sistema en un futuro carezca de observadores y sea más inteligente y capaz.

6.7 Resumen

En este capítulo se registraron los resultados obtenidos de la implementación del sistema de detección de víctimas no superficiales, en pruebas simulando circunstancias que representaron ambientes que se pueden presentar en condiciones reales. La búsqueda por inspección con la cámara infrarroja y la cámara web permitió identificar víctimas por simple observación, al representar en RGB con tonos personalizados la escala de gris de la cámara IR e implementar una fuente de luz artificial blanca para los casos de completa oscuridad para la cámara C920.

La metodología de búsqueda propuesta, obtuvo buenos resultados al lograr detectar la posición angular de ciertas fuentes de calor que estén en el campo de visión de la imagen. El reconocimiento de manos, cabeza o cuerpo completo de una víctima permitieron localizar las partes susceptibles a detectar ROI's para buscar las paredes arteriales por medio del detector de bordes.

Capítulo 7

Conclusiones

7.1 Conclusiones generales

El sistema electrónico de detección propuesto basado en el procesamiento de imágenes de una cámara infrarroja, obtuvo buenos resultados en la búsqueda de víctimas no superficiales. La detección por observación permite distinguir las formas de las diferentes fuentes de calor y objetos que estén en el campo de visión de la cámara, reforzado por la imagen que proporciona la cámara web auxiliada con una fuente de luz blanca artificial, ya que en ocasiones la posición o visualización de las víctimas puede ser confusa.

Al obtener la posición angular de fuentes de calor en función del centro de la imagen, se puede orientar el campo de visión de la cámara para enfocar completamente el cuerpo y proceder a buscar alguna forma antropomórfica por observación o aplicando los clasificadores en cascada Haar, que se demostraron ser factibles para aplicarse en la detección de víctimas en vista de los resultados obtenidos con los detectores de manos y cabeza. En el caso de la detección del cuerpo resultó ser más complicado, esto se reflejó en el porcentaje de detección, debido a las múltiples posiciones en las que se puede estar, además de que no es seguro poder llegar a visualizar todo el cuerpo.

La detección del pulso cardíaco en una víctima es posible al poder tener la lente de la cámara a una distancia máxima de un metro, esta conclusión se basa en los resultados obtenidos al aplicar el detector de bordes Canny para medir el pulso cardíaco por observación, los cuales tuvieron un margen de error bajo considerando el tipo de procesamiento aplicado.

El sistema de detección propuesto es factible de ser optimizado para llevarse a pruebas en condiciones más reales, como lo puede ser en los campos de entrenamiento de perros de búsqueda y rescate, integrando el módulo de búsqueda a un robot móvil como se planteó en el

diseño conceptual. Se logró detectar un signo vital y se descubrió que también se podría detectar la respiración al procesar imágenes de las fosas nasales.

7.2 Conclusiones particulares

La demanda de recursos computacionales que requiere la adquisición de datos y su transmisión al otro módulo, permite exportar el hardware y software del módulo de búsqueda a un sistema embebido que tenga integrado un puerto Giga Ethernet, como las Banana Pi Pro, ya que al realizar las pruebas con la tarjeta de red USB a plataformas que no contaban con este puerto ethernet (como las Raspberry y Beaglebone Black), se observó que consumían alrededor del 80% del procesador, lo que no las hace factibles para manejar por lo menos la cámara infrarroja.

El funcionamiento del intercambio de información entre los módulos así como la adquisición de datos de las cámaras y el micrófono no presentó problemas, aún cuando se ejecutan en paralelo diferentes aplicaciones. Sin embargo al aplicar procesos con un alto grado de operaciones, se observó que se debe controlar la publicación de mensajes a través de los buses de comunicación, para que no sobrecarguen tanto la memoria como el procesador.

La detección de las regiones de interés para medir el pulso, se logró de forma satisfactoria al lograr detectar arterias superficiales y zonas del cuerpo que cuentan con una mayor temperatura que la piel, como la nariz o los oídos en los cuales son puntos clave de temperatura que se podría medir al considerar las variables e implicaciones físicas.

El algoritmo de detección del pulso implementado permitió observar la contracción y expansión de las paredes arteriales del segmento visible en la ROI, y medir por visualización la frecuencia del signo vital. Sin embargo, para realizar una medición sin observador se tiene que aplicar mejores procesamientos como el propuesto por Chekmenev [5]. En tanto a la detección de formas antropomórficas aunque se hayan logrado buenos resultados para detectar las manos y la cabeza, se pueden mejorar al considerar otros clasificadores que consuman menos recursos computacionales o entrenando de forma más robusta al considerar más muestras.

7.3 Trabajo a futuro

A continuación se describe algunas de las propuestas de trabajo a futuro en las 3 principales divisiones del sistema de detección.

7.4.1 Hardware

1. Exportar el módulo de búsqueda a uno de los sistemas embebidos propuestos en el capítulo 3, sugiriendo que se trabaje con la plataforma Banana Pi 2, por tener integrado

un puerto Giga Ethernet y porque todo su hardware fue diseñado para ser completamente utilizado con sus diferentes distribuciones de sistema operativo.

2. Hacer un sistema de protección de las lentes de las cámaras, contra polvo y que evite el contacto directo de cualquier cuerpo u objeto, específicamente la lente de la cámara infrarroja deberá de tener un protector que se transparente a la radiación de longitud de onda larga.

3. Diseñar un soporte para ambas cámaras para que puedan situarse a una distancia tal, que permite superponer ambas imágenes (como si se tratará de una cámara con visión estéreo) con la finalidad de complementar las capacidades de cada una de las cámaras.

7.4.2 Software

1. Desarrollar interfaces gráficas que permitan iniciar el sistema y controlarlo fuera de la terminal.

2. Implementar una comunicación inalámbrica entre ambos módulos, en vista de que el consumo de ancho del bus que transporta las señales de audio y video es menor a los 100 Mbps.

3. Optimizar los nodos publicadores de video, para no sobrecargar el canal de comunicación que transmite los datos de un módulo a otro, asignando banderas que inicien y detengan la publicación de los mensajes de video únicamente cuando haya algún suscriptor requiriendo dicha información.

7.4.3 Procesamiento digital de señales

1. Implementar la medición de temperatura a distancia, basado en la teoría descrita en el capítulo 3 que es respaldada por la información proporcionada por el fabricante de la cámara infrarroja en su manual [8].

2. Desarrollar algoritmos con el objetivo de detectar la respiración a través de los gradientes de temperatura que se definen en las fosas nasales, así como en algoritmos más robusto para la detección del pulso.

3. Desarrollar algoritmos para procesar la señal acústica que proporciona el micrófono estéreo de la cámara web C920, para tratar de detectar la dirección de la fuente de sonido, para orientar la exploración.

Bibliografía

1. USAID/OFDA Manual de campo. Búsqueda y rescate en estructuras colapsadas (BREC) Diciembre 2006. Embajada de Estados Unidos de América, San José, Costa Rica.

2. USAID/OFDA. Manual de referencia 2006. Curso de rescate en estructuras colapsadas nivel liviano.

3. Amer Nezirovic, Trappped victim detection in post disaster scenarios usign ultra wideband radar, International Research Center for Telecommunications and Radar, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of technology, Delft, Netherlands, 2010.

4. Satoshi Tadokoro. Rescue Robotics. DDT Project on robots and systems for urban search and rescue. Springer Dordrecht Heidelberg London New York. e-ISBN 978-1-84882-474-4.

5. CHEKMENEV, Sergey Y., et al. Multiresolution approach for noncontact measurements of arterial pulse using thermal imaging. En *Augmented Vision Perception in Infrared*. Springer London, 2009. p. 87-112.

6. Togawa T., “Non-contact skin emissivity: measurement from reflectance using step change in ambient radiation temperature,” Clin. Phys. Physiol. Meas. 1989, Vol. 10 pp. 39–48

7. James Wong, Cassandra Robinson, et al. Urban Search and Rescue Technology Needs: Identification of Needs. Junio 2004

8. FLIR. The ultimate infrared handbook for R&D professionals. 2012 FLIR System Inc

9. OPTRIS. Basic Principles of non-contact temperature measurement.

10. Joseph Howse. OpenCV computer vision with python. Packt publishing open source. Birmingham - Mumbai. Abril 2013

11. Junghoon Lee, Joosung Lee, Sangha Song, et al. Detection of Suspicious Pain Regions on a Digital Infrared Thermal Image using the Multimodal Function Optimization. Vancouver, British Columbia, Canada, August 2008

12. Jan Portmann, Simon Lynen, Margarita Chli and Roland Siegwart. People Detection and Tracking from Aerial Thermal Views. Autonomous Systems Lab, ETH Zurich

13. Yoshiaki Bando, Takuma Otsuka, Takeshi Mizumoto, Katsutoshi Itoyama, Masashi Konyo, Satoshi Tadokoro,Kazuhiro Nakadai & Hiroshi G. Okuno (2015) Posture estimation of hose-shaped robot by using active microphone array, Advanced Robotics, 29:1, 35-49, DOI: 10.1080/01691864.2014.9812

14. Krick, E. V. Introducción a la Ingeniería y al Diseño de la Ingeniería. Edit. Limusa. México, 1986. 60 - 95 pp

15. Il Chan Lee. Rescue system design. Theses. Rochester Institute Technology, 2007. Documento disponible en línea.http://scholarworks.rit.edu/theses/7882/

16. Serway A. Raymond, Jewett W. John Jr. Physics for Scientists and Engineers, volumen 1, editorial CENGAGE Learning, séptima edición, pp. 572 - 577.

17. Serway A. Raymond, Jewett W. John Jr. Physics for Scientists and Engineers, volumen 2, editorial CENGAGE Learning, séptima edición, pp. 1154 - 1158.

18. Simonoff, Jeffrey S. (1998) Smoothing Methods in Statistics, 2nd edition. Springer ISBN 978-0387947167

19. Kennet Dawson-Howe. A practical introduction to computer vision with OpenCV, Edit. Willey

20. Garima Yadav, et. al Contrast limited adaptative histogram equalization based enhancement for real time video system. 2014 International Conference on Advances in Computing,Communications and Informatics (ICACCI)

21. Gary Bradsky & Adrian Kaehler. Computer Vision with the OpenCV Library, editorial O'Reilly ISBN:978-0-596-51613-0.

22. Paul Viola, Michael Jones. Rapid object detection using a boosted cascade of simple features. CVPR, issue 1, 2001, pp. 511–518.

23. C. Papageorgiou, M. Oren, and T. Poggio. Pedestrian detection using wavelets templates, Computer vision and pattern recognition, 1999

24. Irving P. Herman Physics of the human body, second edition, editorial Springer, ISBN 978-3-319-23932-3

25. R.A. Freitas Jr., Nanomedicine, Volume I: Basic Capabilities (Landes Bioscience, Austin, 1999)

26. Sungjun Kwon, et. al. Validation of heart rate extraction using video imaging on a built-in camera system of a smartphone. 34th Annual International Conference of the IEEE EMBS, San Diego, California USA, 28 August - 1 September, 2012

27. Lentin Josep. Mastering ROS for robotics programming, editorial Packt publishing open source, Diciembre 2015

Mesografía

1. Runaround story, reseña del cuento de ciencia ficción escrito por Isaac Asimov, [artículo en línea], Wikipedia the free encyclopedia, última actualización 2/07/2015 [citado 18/07/2015], disponible en internet.
https://en.wikipedia.org/wiki/Runaround_(story)

2. LEADER search and rescue equipment, sitio web de la empresa [citado 18/07/2015] http://www.leader-group.eu/products/search-rescue-equipment/life-detectors-locators-216.html

3. Elizabeth Landau. FINDER Search and Rescue Technology Helped Save Lives in Nepal, artículo sobre el uso del radar UWB desarrollado por la NASA, [artículo electrónico en línea], Noticias, NASA Laboratorio de propulsión jet, Instituto Tecnológico de California, redactado el 7/05/2015 [citado 18/05/2015], disponible en internet.
http://www.jpl.nasa.gov/news/news.php?feature=4578

4. Smartdust, breve descripción del concepto [artículo en línea] , Wikipedia the free encyclopedia, última actualización 16/04/2015 [citado 18/07/2015], disponible en internet.
http://en.wikipedia.org/wiki/Smartdust

5. Capnography, artículo en línea sobre este procedimiento médico. Wikipedia the free encyclopedia, última actualización 15/07/2015 [citado 19/07/2015], disponible en internet
https://en.wikipedia.org/wiki/Capnography

6. Infrared thermography - Physical basic concept, Documentación sobre el funcionamiento de la termografía por infrarrojo, Infratec, [página web] [citado 20/07/2015]
http://www.infratec.eu/thermography/thermography-knowledge/theory.html

7. FEMA National USAR response system. Appendix A. Search Capabilities. Course Notebook, documento del curso
http://www.w1npp.org/ARES/BUILDI~1.PDF

8. Robocup Rescue, Robocup Rescue Wiki, article, descripción de la competencia internacional e información acerca del escenario de desastre simulado, última actualización 16/05/2015 [citado el día 20/07/2015]
http://www.robocuprescue.org/wiki/index.php?title=Main_Page

Anexo A

A.1 Estimación del ancho de banda para la transmisión de video de la cámara infrarroja

El tamaño de la imagen que nos proporciona la cámara FLIR A35 es de 336 x 256, adquiriendo 60 cuadros por segundo. El formato en el que proporciona la transmisión de video es MPEG-4 en el espacio de colores monocromático de 8 o 14 bits, por los representa los diferentes tonos de gris. Por razones de compatibilidad se trabajó con el formato de 8 bits, el cual viene configurado directamente por el fabricante.

Estimar el ancho de banda que se requerirá para la transmisión de video, es útil para la selección del sistema embebido o procesador que se encargará de adquirir la información de la cámara infrarroja y enviarla al módulo de procesamiento para su análisis. Las imágenes adquiridas se dividen en paquetes para ser transportadas al punto en donde sean solicitadas, dichos paquetes tiene la estructura del protocolo GigE Vision, donde la mayor parte del paquete corresponde a la porción de imagen transportada. El ancho de banda para la transmisión de la imagen se calcula con la ecuación A.1

$$Image\ bandwidth\ [MB/s]\ =\ Height\ x\ Width\ x\ FPS\ \ x\ Bytes\ per\ píxel \tag{A.1}$$

Sustituyendo valores, obtenemos:

$$BW_{image} = 336\ x\ 256\ x\ 60\ x\ 1\ \approx 4.91\ MB/s\ \approx 39.3\ Mb/s$$

Este dato representa una estimación del ancho de banda requerido para transmitir el video, ya que a cada uno de los paquetes en los que se dividen los cuadros de vídeo, se anexa otra información característica del transporte de información del protocolo de la cámara, como se observa en la figura 4.2, sin embargo el tamaño de dicha información no es tan significativo.

El estándar IEEE 802.3 establece que todos los puertos Giga Ethernet deben de autonegociar el tamaño de los paquetes que portarán la información, haciéndolos compatibles con los puertos fast ethernet, de menor velocidad de transmisión, sin embargo la SDK requiere específicamente para su funcionamiento, un puerto Giga Ethernet para poder trabajar con equipos GigE Vision, sin importar la cantidad de información que se vaya a transmitir.

Una vez que se reciben los paquetes que contienen la información de los cuadros adquiridos por la cámara térmica (entre 56 y 58 paquetes, considerando que cada uno de 1476 bytes),

dependiendo el modelo de la tarjeta de red con la que se trabaje, se ocupará en diferente proporción el procesador para construir la imagen y poder utilizarla. Por ello en anexo B se realizaron pruebas a sistemas embebidos con una tarjeta de red USB - Giga Ethernet modelo TP-LINK UE300, ya que parte del diseño del sistema se enfocó en compactar el módulo de búsqueda que podrá ser portado por un robot móvil.

A.2 Alimentación de la cámara térmica

Este equipo tiene dos opciones de para su alimentación, acorde a su implementación se deberá de elegir alguna de ellas, a continuación se explican ambas opciones de forma detallada.

Power Over Ethernet (PoE)

Es una derivación (af y at) del estándar IEEE 802.3 que incorpora la alimentación de equipos por medio de un cable de red ethernet (utilizando conectores RJ-45), esta tecnología se implementa de dos diferentes formas que dan lugar a la clasificación de los dispositivos mostrados en la siguiente tabla:

No. de pin	Color acorde al estándar T568B	Configuración del conector tipo B DC & datos		Configuración del conector tipo A DC & datos	
1	Blanco-naranja	TxRx A+		TxRx A+	DC +
2	Naranja	TxRx A-		TxRx A-	DC +
3	Blanco-verde	TxRx B+		TxRx B+	DC -
4	Azul	TxRx B-	DC +	TxRx B-	
5	Blanco-azul	TxRx C+	DC +	TxRx C+	
6	Verde	TxRx C-		TxRx C-	DC -
7	Blanco-café	TxRx D+	DC -	TxRx D+	
8	Café	TxRx D-	DC -	TxRx D-	

Tabla A.1 Tipos de implementación del estándar PoE

Modelo B

A través de los pines 4,5 (DC +) ,7 y 8 (DC -) se lleva la alimentación al dispositivo, estando completamente separada de los pares que transportan los datos. Esta fue la primer implementación de PoE, ya que el estándar no utilizaba dichos pines.

Modelo A

El voltaje de DC se transmite junto con los pares de cables que llevan los datos, es decir los pines 1,2 (DC+) y 3, 6 (DC-). Acorde a la potencia se subdividen los dispositivos, la cámara FLIR A35 es clase 0, requiere de 56V DC/2.5 W máximo para funcionar y de un cable UTP categoría 6.

Alimentar la cámara de esta forma, requiere de tener una fuente que agregue dicho voltaje a los pines citados del conector ethernet, como la fuente PHIHONG POE16R-1AFG incluida en la cámara. Básicamente su función es suministrar el voltaje requerido de DC utilizando AC y llevarlo a la cámara por medio de los pines. Además de separar dicho voltaje para que por medio de otro conector se pueda acceder a los datos. Sin embargo esto hace dependiente la aplicación de tener una fuente de AC, pero se puede diseñar una que implemente una fuente conmutada booster y que todo trabaje con DC.

Pines de alimentación

La segunda opción para alimentar la cámara, es mediante los pines 1 (DC-) y 2 (DC+) del conector M12, con una diferencia de potencial 12 o 24 VDC. De esta forma, se puede conectar directamente la cámara a otra NIC sin tecnología PoE, razón por la que se usaron estos pines en la implementación del sistema de detección, ya que requiere de menos hardware. La distribución de pines del conector M12 se muestra en la figura A.1

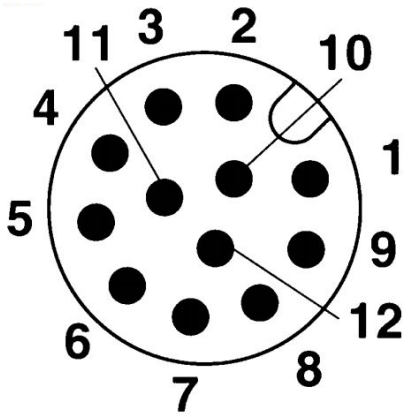
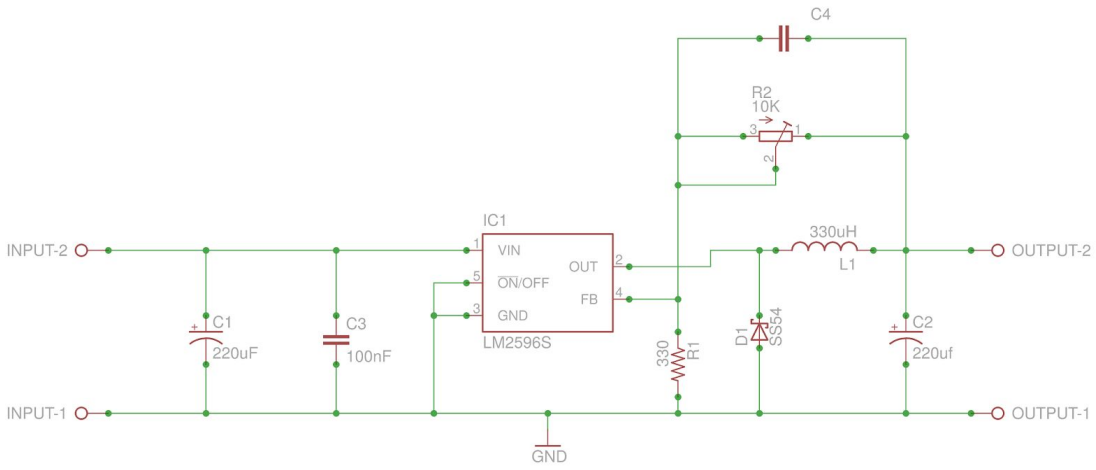


Figura A.1 Distribución de los pines del conector M12 de la cámara FLIR A35

Para independizar la cámara de la corriente alterna, se propuso suministrar a la cámara 12V regulados utilizando el circuito integrado (IC) LM2596, el cual es un regulador de voltaje de la

compañía Texas Instrument en conjunto con una batería de litio polímero de 4 celdas conectadas en serie de 3.7 V cada una. La configuración que se empleó del IC fue la de fuente conmutada paso bajo, para que reduzca el voltaje de entrada y lo regule a 12V, teniendo la ventaja de que al ser menor la diferencia de potencial de entrada que la de salida en un 25%, el voltaje de salida continua regulado. En la figura A.2 se muestra el diagrama de la fuente de voltaje así como una fotografía del circuito ensamblado que se adquirió por practicidad, puesto que cuenta con un voltímetro que permite medir tanto el voltaje de entrada como de salida.



A

B

Figura A.2 (A) Diagrama de la configuración step-down del IC LM2596 y (B) fotografía de la fuente adquirida

Para suministrar la energía por medio de este conector se adquirió la parte hembra del M-12 en la tienda Digi-Key, catalogado con la clave Harting 21348600C79010.

Anexo B

Pruebas de velocidad de transferencia realizadas a los sistemas embebidos sin puerto Giga Ethernet

Para definir la factibilidad del uso de los sistemas embebidos BeagleBone Black y Raspberry Pi B+, se adquirió una tarjeta de red USB 3.0 - Giga Ethernet modelo TP-LINK UE300 para poder conectar la cámara infrarroja a uno de los puertos usb de estas plataformas.

Por principio de funcionamiento, la tasa de transferencia máxima que se podría tener en el puerto es de 480 Mbps, lo cual es superior a lo que ofrecen los puertos Fast ethernet que son los que tienen estos sistemas embebidos de prueba. El ancho de banda requerido para la transmisión del video de la cámara es de 40 Mb/s (acorde a los cálculos realizados en el anexo A), sin embargo específicamente la SDK únicamente puede trabajar con tarjetas de red Giga Ethernet.

Para realizar las pruebas de conexión y desempeño de las dos plataformas citadas en el primer párrafo , se les instaló el sistema operativo Ubuntu 14.04 para procesadores ARM, OpenCV, ROS Indigo y la SDK eBUS. Las dos pruebas que se hicieron son de reconocimiento de la cámara y transmisión del video, ejecutando los programas de ejemplo PvStreamSample y PvDeviceFinder, los cuáles se encuentran en la dirección

/opt/pleora/ebus_sdk/Ubuntu-14.04-x86_64/share/samples/

Prueba de comunicación y conexión con la cámara infrarroja

Se conectó la cámara con la tarjeta de red a uno de los puertos USB de la Raspberry y BeagleBone Black, y se ejecutó el programa el cual busca todos los dispositivos tipo GEV y USB 3 Vision que están conectados a la computadora. En la pantalla de la terminal se despliega una lista de las interfaces de red disponibles en el sistema, y al encontrar la cámara muestra la información referente a la configuración de su puerto de red. En la figura B.1 se muestra la información que se muestra en la pantalla.

Figura B.1 Información desplegada en la pantalla de la terminal al ejecutar el programa DevideFinder

En ambas plataformas (BeagleBone Black y Raspberry pi B+) se logró la comunicación y conexión de la cámara al utilizar este programa, sin tener problema alguno.

Prueba de transmisión de video

A raíz de que se logró comunicar la cámara con los dos sistemas embebidos, se probó la transmisión de video con el ejemplo de PvStreamSample, para ver el rendimiento de las plataformas. En la figura B.2 se muestra el funcionamiento normal de este programa en el módulo de búsqueda.

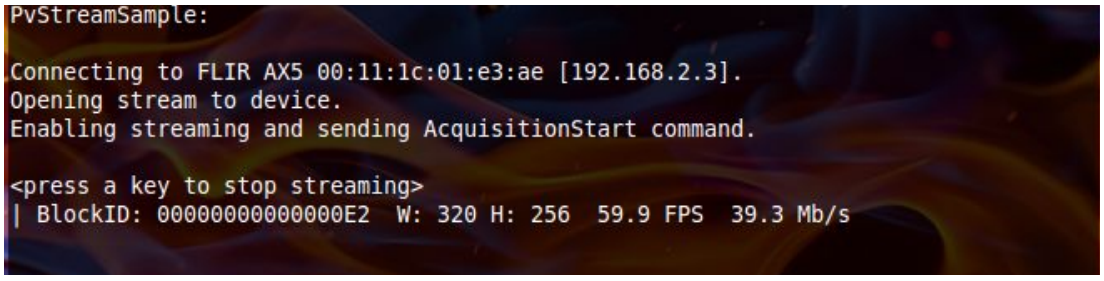


Figura B.2 Captura de pantalla del funcionamiento normal del programa PvStreamSample

Al ejecutar este programa en la BeagleBone Black, la comunicación con la cámara se inicia y al momento de realizar la transmisión de video, durante tres segundo se mantiene el número de frames recibidos (60) pero después comienza a descontrolarse dicho valor hasta que el sistema operativo de la plataforma se pasma completamente y no vuelve a recuperarse hasta reiniciar por completo el sistema, interrumpiendo el consumo de energía. En este suceso se observó en otra terminal el consumo de memoria y del procesador con la instrucción *top*, en este monitor se registró que el procesador de la tarjeta se ocupó al 100 % al momento de tener la transmisión activa, esto concluye que no puede ser utilizada en esta plataforma al no lograr mantener la adquisición de video de la cámara.

En la Raspberry Pi B+, la transmisión de video se mantiene activa pero consume el 90% del procesador, esto complica la ejecución de algún otro programa en esa misma plataforma. Por ello el utilizar esta plataforma podría ser factible si solo se considera que llevará la cámara infrarroja, sin embargo no es recomendable al proponer otras opciones en la tabla 3.2.

Anexo C

C.1 Instalación de la eBUS SDK en Ubuntu

El controlador necesario para utilizar la cámara térmica es instalado junto con el conjunto de bibliotecas que permiten controlar dispositivos GEV.

La SDK se encuentra disponible de forma gratuita en la página web de la empresa desarrolladora, la cual solicita registrarse como usuario para poder acceder a la descarga y a su documentación. La dirección es:

http://www.pleora.com/support-center/documentation-and-downloads/79

Se utilizó la versión 4.1.4 para Ubuntu de 64 bits, la cual se instala extrayendo del archivo .zip el script que realiza todo el procedimiento de instalación, dividido por etapas que se describen a continuación.

1. Abrir una terminal y dirigirse mediante la línea de comandos a la carpeta donde se extrajo el el archivo script.

2. Como super usuario, ejecutar el archivo con terminación .run, esto se hace escribiendo ./ seguido del nombre del archivo, como se muestra a continuación.

   sudo ./eBUS_SDK_4.1.4.3606_Ubuntu-14.04-x86_64.run

   El nombre del archivo puede cambiar acorde a la versión que se haya descargado, sin embargo debe tener en común la terminación .run

3. Este script instala diversas herramientas y hace algunas configuraciones a los puertos que requieren de la autorización e intervención del usuario, por ello se debe estar al tanto de lo que vaya apareciendo en la terminal durante su ejecución, ya que durante la instalación nos requiere dos autorizaciones, las cuales se explican a continuación:

   3.1. Se debe autorizar la inclusión de las bibliotecas eBUS SDK y GenICam a la lista de bibliotecas compartidas disponibles en nuestro sistema operativo. Para hacerlo en cuanto pregunte si deseamos agregarlas, basta con presionar la tecla Enter para que tome la respuesta predefinida, que es si o en su caso escribirlo en letras minúsculas en inglés.

   3.2. La segunda autorización que se debe dar es para compilar el controlador eBUS Universal Pro, el cual optimiza la operación de los dispositivos GEV al modificar

el tamaño de los paquetes que se transportan por los puertos de comunicación. La compilación se llevará a cabo exitosamente si se cuentan con los paquetes básicos de construcción del kernel (*build-essential* y *linux-headers*) y con el conjunto de herramientas *gcc*, de lo contrario fallará.

Si la compilación falla, el script de instalación continuará sin verse afectado, ya que no es necesario para la instalación, a no ser que se maneje más de un cámara con un mismo puerto ethernet utilizando un switch. Sin embargo es recomendable que se compile, por lo que al terminar la instalación, se podrán instalar los paquetes necesarios y ejecutar el manualmente el script que compila este controlador, el cual está en la carpeta de instalación

/opt/pleora/ebus_sdk/Ubuntu-14.04-x86_64/module

Carpeta donde se instaló la SDK

Para volver intentar compilar el controlador, se ejecuta el script build.sh como administrador, escribiendo en la línea de comandos

sudo ./build.sh

NOTA
Antes de utilizar la cámara es recomendable (pero no necesario) iniciar este controlador, siempre y cuando se haya compilado. Para ello se debe de ejecutar como superusuario el script *load.sh*, el cual está dentro del mismo directorio module.

Para desactivarlo se ejecuta el script *unload.sh*. Este controlador no se inicia de forma automática, por lo que cada vez que se apague o reinicie la computadora o sistema embebido, se deberá iniciar de forma manual.

3.3. Por último, tenemos que deshabilitar el strict Reverse Path Forwarding Filtering, el cual nos permitirá encontrar dispositivos GEV con diferente configuración de red a la de nuestro puerto, si no lo autorizamos no podremos encontrar la cámara.

El script realiza los cambios y reinicia la pila de red para aplicar los cambios realizados, los cuáles se pueden cambiar posteriormente ejecutando el script *set_rp_filter.sh* localizado en el directorio donde se instalo la SDK

/opt/pleora/ebus_sdk/Ubuntu-14.04-x86_64/bin

Hecho lo anterior, ya se cuenta con la instalación de la SDK, solo resta instalar el controlador eBUSd.

C.2 Instalación del controlador eBUSd

Al igual que la SDK, solo se debe ejecutar como superusuario otro script, el cual se encuentra en la carpeta bin, dentro del directorio de instalación de la SDK. El archivo es: *install_daemon.sh*

Al preguntar el modo de funcionamiento, seleccionar la opción por default (opción 0, iniciar el controlador de forma manual) y con ello queda instalado.

Para iniciar este controlador, se debe de escribir en la terminal (sin importar la ubicación en el sistema en la que uno se encuentre) la siguiente instrucción

sudo service eBUSd start

La última palabra es el comando que maneja el controlador, el cual que puede detenerlo (stop), reiniciar (restart), iniciarlo (start) o simplemente revisar si está activo o inactivo (status).

Para que se inicie automáticamente este controlador junto con el arranque del sistema, se debe de cargar y habilitar este servicio en la lista *update-rc.d,* para hacerlo basta con ejecutar en la terminal estas dos instrucciones por separado:

sudo update-rc.d eBUSd defaults

y

sudo update-rc.d eBUSd enable

Para desactivarlo, basta con ingresar la última instrucción, cambiando la palabra *enable* por *disable.*

C.3 Compilación y ejecución de aplicaciones utilizando la SDK

El controlador eBUSd debe estar activo cada vez que vayamos a utilizar la SDK, para que funcionen correctamente nuestras aplicaciones y también se deben cargar ciertas variables ambiente tanto para compilar como para ejecutar los programas, esto lo hacemos colocando como fuente el archivo *set_puregev_env* que se encuentra en la carpeta de bin, de la siguiente manera:

source /opt/pleora/ebus_sdk/Ubuntu-14.04-x86_64/bin/set_puregev_env

Para que estas variables se carguen automáticamente cada que se inicie una terminal, se deberá de colocar la instrucción anterior al archivo *.bashrc* que se encuentra en el directorio home y únicamente se puede editar con permisos de superusuario con la siguiente instrucción sin importar la ubicación en la que se encuentre en el sistema desde la terminal.

sudo nano ~/.bashrc

Para corroborar tanto la instalación, como las configuraciones necesarias podemos compilar los ejemplos instalados, ejecutando el archivo *./build.sh* que se encuentra en dentro de la carpeta de instalación en el directorio *share/samples*.

Algunos de los ejemplos utilizan bibliotecas de Qt en versión mínima de 4.6 (eBUSPlayer), por lo que al ejecutar el script build que compila los ejemplos, se podrá presentar un error si no se cuenta con dicha biblioteca, sin embargo todos los demás ejemplos deberán de compilarse sin problema alguno, si se realizó correctamente la instalación y configuración.

Anexo D

Configuración de la red punto a punto (cámara infrarroja - módulo de búsqueda)

Para poder realizar la transmisión de video de la cámara térmica al módulo de búsqueda, se deben asignar direcciones IP estáticas y una misma submascara de red a ambos puertos, además por supuesto de conectarse mediante un cable de red. A continuación se explicará el procedimiento para configurar cada uno de los puertos.

Configuración de la cámara

Se asignará la dirección IP estática y la submascara de red, mediante la interfaz FLIR IP Configure, esta aplicación se descarga del sitio web de forma gratuita al registrarse, la dirección es

http://www.flir.com/instruments/display/?id=53558

Este programa solo funciona en Windows, por lo que esta configuración debe realizarse en una computadora con dicho sistema operativo.

Antes de iniciar la aplicación, la cámara térmica debe de estar conectada a la computadora o sistema embebido, alimentando la cámara vía PoE o por medio de los pines correspondientes del conector M12. Si la conexión es correcta en la ventana de la aplicación nos deberá de aparecer la cámara con su dirección IP, acorde a la configuración del fabricante, como se muestra en la figura D.1.

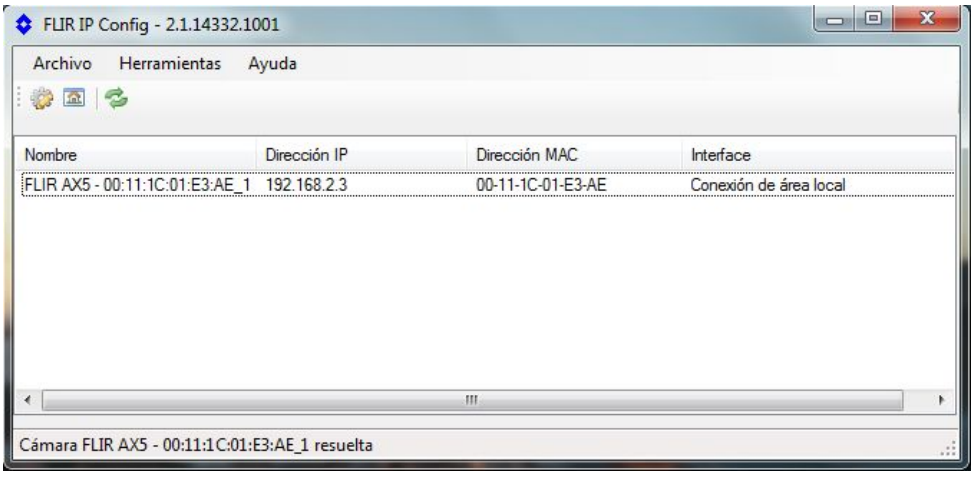


Figura D.1 Detección de cámara térmica con la aplicación FLIR IP Config

Con el cursor seleccionamos la cámara y damos click derecho par desplegar un pequeño menú en donde seleccionaremos la opción de “modificar”. Se desplegará otra ventana en la que escribiremos la configuración asignada a la cámara acorde a la Tabla 4.1, como se muestra en la figura D.2.

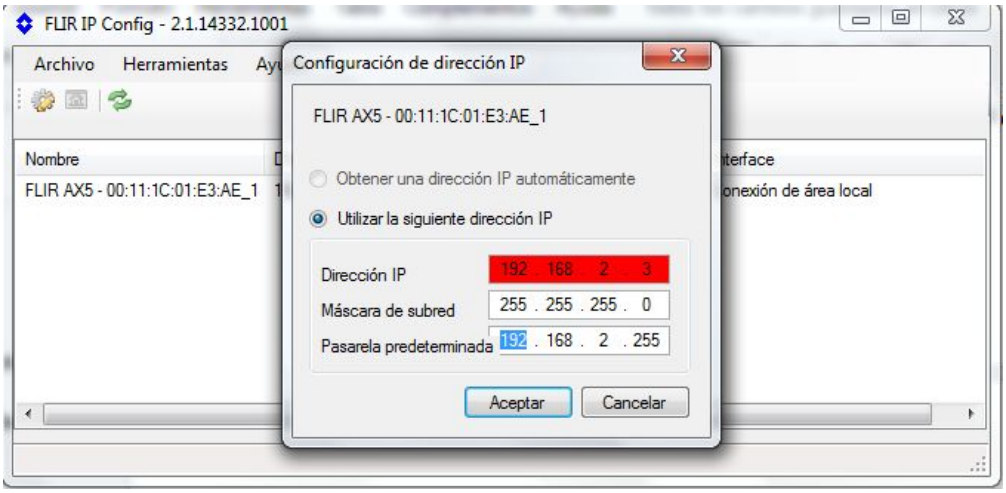


Figura D.2 Ventana de configuración de los parámetros de la tarjeta de red de la cámara FLIR A35

Para terminar, simplemente presionamos el botón de OK y los datos se guardarán en la tarjeta de red de la cámara, lo cual será confirmado por una cuadro de diálogo.

Configuración del módulo de búsqueda

En Ubuntu dichas configuraciones se realizan en el archivo *interfaces*, el cual es parte de los archivos del sistema y está alojado en la dirección */etc/network*.

Para poder escribir la configuración del puerto ethernet, se debe de abrir el archivo como superusuario desde la terminal con algún editor de texto, por ejemplo, utilizando el editor nano (instalado comúnmente en las distribuciones de Ubuntu) se haría de la siguiente forma:

sudo nano /etc/network/interfaces

En el archivo escribimos la configuración que aparece en la figura D.3, que corresponde a los parámetros definidos para el módulo de búsqueda. De existir otra información en este archivo, como lo son las primeras tres líneas que se aprecian en la figura D.3, simplemente se dejan como están y se comienza a escribir la configuración a partir de auto eth0 (en caso de tener más de un puerto, revisar a cual se conecta).

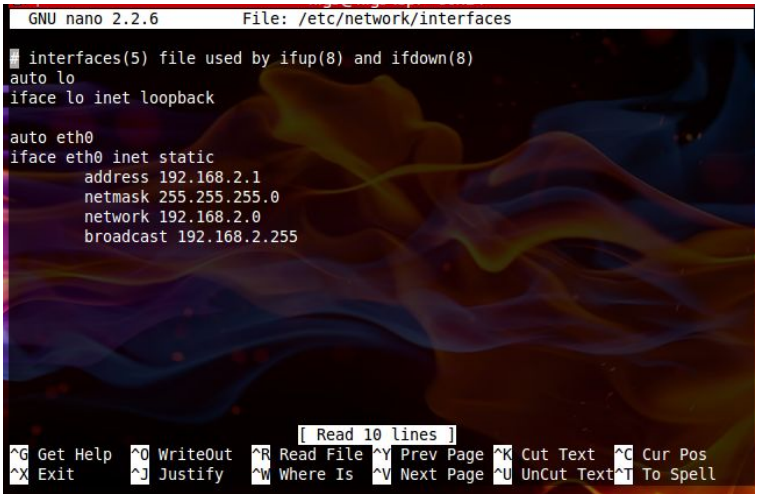


Figura D.3 Configuración del puerto ethernet del módulo de adquisición

Escrita la configuración, para salir del editor nano, se tiene que presionar las teclas $Ctrl + X$, antes de salir nos preguntará si queremos guardar los cambios al archivo, por lo que se deberá indicar que si.

Para finalizar, se reinicia la computadora o sistema embebido, para que asigne los cambios realizados al puerto y con ello se termina de configurar la red punto a punto.

Para regresar a la configuración que tenía el puerto ethernet, simplemente se pueden comentar todas las líneas que escribimos en el archivo interfaces o borrarlas, guardar los cambios y volver a reiniciar.

En resumen, la configuración de la red debe quedar como se muestra en el diagrama de la figura D.4

Figura D.4 Diagrama de configuración de red de los puertos Giga Ethernet para conectar la cámara FLIR A35 al módulo de búsqueda.

Anexo E

E.1 Código para la transmisión de video de la cámara infrarroja

La SDK eBUS únicamente funciona bajo el lenguaje de programación de C++, como parte de su documentación proporciona una serie de ejemplos, siendo uno de ellos el código para iniciar la transmisión de video que se muestra a continuación.

```
#include <PvSampleUtils.h>
#include <PvDevice.h>
#include <PvDeviceGEV.h>
#include <PvDeviceU3V.h>
#include <PvStream.h>
#include <PvStreamGEV.h>
#include <PvStreamU3V.h>
#include <PvBuffer.h>

#ifdef PV_GUI_NOT_AVAILABLE
#include <PvSystem.h>
#else
#include <PvDeviceFinderWnd.h>
#endif // PV_GUI_NOT_AVAILABLE

#include <list>
typedef std::list<PvBuffer *> BufferList;

PV_INIT_SIGNAL_HANDLER();

#define BUFFER_COUNT ( 16 )

///
/// Function Prototypes
///
#ifdef PV_GUI_NOT_AVAILABLE
const PvDeviceInfo *SelectDevice( PvSystem *aPvSystem );
#else
const PvDeviceInfo *SelectDevice( PvDeviceFinderWnd *aDeviceFinderWnd );
#endif // PV_GUI_NOT_AVAILABLE
PvDevice *ConnectToDevice( const PvDeviceInfo *aDeviceInfo );
```

```
PvStream *OpenStream( const PvDeviceInfo *aDeviceInfo );
void ConfigureStream( PvDevice *aDevice, PvStream *aStream );
void CreateStreamBuffers( PvDevice *aDevice, PvStream *aStream, BufferList
*aBufferList );
void AcquireImages( PvDevice *aDevice, PvStream *aStream );
void FreeStreamBuffers( BufferList *aBufferList );

//
// Main function
//
int main()
{
	const PvDeviceInfo *lDeviceInfo = NULL;
	PvDevice *lDevice = NULL;
	PvStream *lStream = NULL;
	BufferList lBufferList;

	PV_SAMPLE_INIT();

#ifdef PV_GUI_NOT_AVAILABLE
	PvSystem *lPvSystem = new PvSystem;
	lDeviceInfo = SelectDevice( lPvSystem );
#else
	PvDeviceFinderWnd *lDeviceFinderWnd = new PvDeviceFinderWnd();
	lDeviceInfo = SelectDevice( lDeviceFinderWnd );
#endif // PV_GUI_NOT_AVAILABLE

	cout << "PvStreamSample:" << endl << endl;

	if ( NULL != lDeviceInfo )
	{
	lDevice = ConnectToDevice( lDeviceInfo );
	if ( NULL != lDevice )
	{
	lStream = OpenStream( lDeviceInfo );
	if ( NULL != lStream )
	{
	ConfigureStream( lDevice, lStream );
	CreateStreamBuffers( lDevice, lStream, &lBufferList );
	AcquireImages( lDevice, lStream );
	FreeStreamBuffers( &lBufferList );

	// Close the stream
```

```
        cout << "Closing stream" << endl;
        lStream->Close();
        PvStream::Free( lStream );
        }

        // Disconnect the device
        cout << "Disconnecting device" << endl;
        lDevice->Disconnect();
        PvDevice::Free( lDevice );
        }
        }

        cout << endl;
        cout << "<press a key to exit>" << endl;
        PvWaitForKeyPress();

#ifdef PV_GUI_NOT_AVAILABLE
        if( NULL != lPvSystem )
        {
        delete lPvSystem;
        lPvSystem = NULL;
        }
#else
        if( NULL != lDeviceFinderWnd )
        {
        delete lDeviceFinderWnd;
        lDeviceFinderWnd = NULL;
        }
#endif // PV_GUI_NOT_AVAILABLE

        PV_SAMPLE_TERMINATE();

        return 0;
}

#ifdef PV_GUI_NOT_AVAILABLE
const PvDeviceInfo *SelectDevice( PvSystem *aPvSystem )
{
        const PvDeviceInfo *lDeviceInfo = NULL;
        PvResult lResult;

        if (NULL != aPvSystem)
        {
```

```
        // Get the selected device information.
        lDeviceInfo = PvSelectDevice( *aPvSystem );
        }

        return lDeviceInfo;
}
#else
const PvDeviceInfo *SelectDevice( PvDeviceFinderWnd *aDeviceFinderWnd )
{
        const PvDeviceInfo *lDeviceInfo = NULL;
        PvResult lResult;

        if (NULL != aDeviceFinderWnd)
        {
        // Display list of GigE Vision and USB3 Vision devices
        lResult = aDeviceFinderWnd->ShowModal();
        if ( !lResult.IsOK() )
        {
        // User hit cancel
        cout << "No device selected." << endl;
        return NULL;
        }

        // Get the selected device information.
        lDeviceInfo = aDeviceFinderWnd->GetSelected();
        }

        return lDeviceInfo;
}
#endif // PV_GUI_NOT_AVAILABLE

PvDevice *ConnectToDevice( const PvDeviceInfo *aDeviceInfo )
{
        PvDevice *lDevice;
        PvResult lResult;

        // Connect to the GigE Vision or USB3 Vision device
        cout << "Connecting to " << aDeviceInfo->GetDisplayID().GetAscii() << "." << endl;
        lDevice = PvDevice::CreateAndConnect( aDeviceInfo, &lResult );
        if ( lDevice == NULL )
        {
        cout << "Unable to connect to " << aDeviceInfo->GetDisplayID().GetAscii() << "."
<< endl;
```

```
	}

	return lDevice;
}

PvStream *OpenStream( const PvDeviceInfo *aDeviceInfo )
{
	PvStream *lStream;
	PvResult lResult;

	// Open stream to the GigE Vision or USB3 Vision device
	cout << "Opening stream to device." << endl;
	lStream = PvStream::CreateAndOpen( aDeviceInfo->GetConnectionID(), &lResult
);
	if ( lStream == NULL )
	{
	cout << "Unable to stream from " << aDeviceInfo->GetDisplayID().GetAscii() << "."
<< endl;
	}

	return lStream;
}

void ConfigureStream( PvDevice *aDevice, PvStream *aStream )
{
	// If this is a GigE Vision device, configure GigE Vision specific streaming
parameters
	PvDeviceGEV* lDeviceGEV = dynamic_cast<PvDeviceGEV *>( aDevice );
	if ( lDeviceGEV != NULL )
	{
	PvStreamGEV *lStreamGEV = static_cast<PvStreamGEV *>( aStream );

	// Negotiate packet size
	lDeviceGEV->NegotiatePacketSize();

	// Configure device streaming destination
	lDeviceGEV->SetStreamDestination( lStreamGEV->GetLocalIPAddress(),
lStreamGEV->GetLocalPort() );
	}
}

void CreateStreamBuffers( PvDevice *aDevice, PvStream *aStream, BufferList
*aBufferList )
```

```
{
	// Reading payload size from device
	uint32_t lSize = aDevice->GetPayloadSize();

	// Use BUFFER_COUNT or the maximum number of buffers, whichever is smaller
	uint32_t lBufferCount = ( aStream->GetQueuedBufferMaximum() <
BUFFER_COUNT ) ?
	aStream->GetQueuedBufferMaximum() :
	BUFFER_COUNT;

	// Allocate buffers
	for ( uint32_t i = 0; i < lBufferCount; i++ )
	{
	// Create new buffer object
	PvBuffer *lBuffer = new PvBuffer;

	// Have the new buffer object allocate payload memory
	lBuffer->Alloc( static_cast<uint32_t>( lSize ) );

	// Add to external list - used to eventually release the buffers
	aBufferList->push_back( lBuffer );
	}

	// Queue all buffers in the stream
	BufferList::iterator lIt = aBufferList->begin();
	while ( lIt != aBufferList->end() )
	{
	aStream->QueueBuffer( *lIt );
	lIt++;
	}
}

void AcquireImages( PvDevice *aDevice, PvStream *aStream )
{
	// Get device parameters need to control streaming
	PvGenParameterArray *lDeviceParams = aDevice->GetParameters();

	// Map the GenICam AcquisitionStart and AcquisitionStop commands
	PvGenCommand *lStart = dynamic_cast<PvGenCommand *>(
lDeviceParams->Get( "AcquisitionStart" ) );
	PvGenCommand *lStop = dynamic_cast<PvGenCommand *>(
lDeviceParams->Get( "AcquisitionStop" ) );
```

```
        // Get stream parameters
        PvGenParameterArray *lStreamParams = aStream->GetParameters();

        // Map a few GenICam stream stats counters
        PvGenFloat *lFrameRate = dynamic_cast<PvGenFloat *>( lStreamParams->Get(
"AcquisitionRate" ) );
//      PvGenFloat *lFrameRate = 30.0;
        PvGenFloat *lBandwidth = dynamic_cast<PvGenFloat *>( lStreamParams->Get(
"Bandwidth" ) );

        // Enable streaming and send the AcquisitionStart command
        cout << "Enabling streaming and sending AcquisitionStart command." << endl;
        aDevice->StreamEnable();
        lStart->Execute();

        char lDoodle[] = "|\\-|-/";
        int lDoodleIndex = 0;
        double lFrameRateVal = 0.0;
        double lBandwidthVal = 0.0;

        // Acquire images until the user instructs us to stop.
        cout << endl << "<press a key to stop streaming>" << endl;
        while ( !PvKbHit() )
        {
        PvBuffer *lBuffer = NULL;
        PvResult lOperationResult;

        // Retrieve next buffer
        PvResult lResult = aStream->RetrieveBuffer( &lBuffer, &lOperationResult, 1000 );
        if ( lResult.IsOK() )
        {
        if ( lOperationResult.IsOK() )
        {
        PvPayloadType lType;

        //
        // We now have a valid buffer. This is where you would typically process the
buffer.
        // -----------------------------------------------------------------------------
        // ...

        lFrameRate->GetValue( lFrameRateVal );
        lBandwidth->GetValue( lBandwidthVal );
```

```
        // If the buffer contains an image, display width and height.
        uint32_t lWidth = 0, lHeight = 0;
        lType = lBuffer->GetPayloadType();

        cout << fixed << setprecision( 1 );
        cout << lDoodle[ lDoodleIndex ];
        cout << " BlockID: " << uppercase << hex << setfill( '0' ) << setw( 16 ) <<
lBuffer->GetBlockID();
        if ( lType == PvPayloadTypeImage )
        {
                // Get image specific buffer interface.
                PvImage *lImage = lBuffer->GetImage();

                // Read width, height.
                lWidth = lImage->GetWidth();
                lHeight = lImage->GetHeight();
                cout << "  W: " << dec << lWidth << " H: " << lHeight;
        }
        else
        {
                cout << " (buffer does not contain image)";
        }
        cout << "  " << lFrameRateVal << " FPS  " << ( lBandwidthVal / 1000000.0 ) << "
Mb/s   \r";
        }
        else
        {
        // Non OK operational result
        cout << lDoodle[ lDoodleIndex ] << " " <<
lOperationResult.GetCodeString().GetAscii() << "\r";
        }

        // Re-queue the buffer in the stream object
        aStream->QueueBuffer( lBuffer );
        }
        else
        {
        // Retrieve buffer failure
        cout << lDoodle[ lDoodleIndex ] << " " << lResult.GetCodeString().GetAscii() <<
"\r";
        }
```

```
        ++lDoodleIndex %= 6;
        }

        PvGetChar(); // Flush key buffer for next stop.
        cout << endl << endl;

        // Tell the device to stop sending images.
        cout << "Sending AcquisitionStop command to the device" << endl;
        lStop->Execute();

        // Disable streaming on the device
        cout << "Disable streaming on the controller." << endl;
        aDevice->StreamDisable();

        // Abort all buffers from the stream and dequeue
        cout << "Aborting buffers still in stream" << endl;
        aStream->AbortQueuedBuffers();
        while ( aStream->GetQueuedBufferCount() > 0 )
        {
        PvBuffer *lBuffer = NULL;
        PvResult lOperationResult;

        aStream->RetrieveBuffer( &lBuffer, &lOperationResult );
        }
}

void FreeStreamBuffers( BufferList *aBufferList )
{
        // Go through the buffer list
        BufferList::iterator lIt = aBufferList->begin();
        while ( lIt != aBufferList->end() )
        {
        delete *lIt;
        lIt++;
        }

        // Clear the buffer list
        aBufferList->clear();
}
```

Anexo F

F.1 Archivo launch para inicializar el sistema de detección

Para ejecutar este archivo launch, ambos módulos tienen que estar conectados entre sí por medio de una red local, con el protocolo SSH. A continuación se presenta el código del archivo launch *start_up.launch* que se localiza en el paquete *start_sys_nsv*.

```
<launch>

        <machine name="SearchModule" address="192.168.2.8"
        env-loader="/opt/ros/ros/env.sh" user="yolo"/>

   <!-- Common Interface -->
        <arg name="device" default="192.168.2.3"/>
        <arg name="rate" default="30"/>

        <!-- Camera Settings -->
        <arg name="ip_address" default="$(arg device)"/>
        <arg name="camera_name" default="flir_a35"/>
        <arg name="camera" default="thermal_camera"/>
        <arg name="frame_id" default="$(arg camera)"/>
        <arg name="calib_url" default=""/>
        <arg name="fps" default="$(arg rate)"/>
        <arg name="raw" default="false"/>

        <!-- Node Settings -->
        <arg name="output" default="screen"/>
        <arg name="view" default="false"/>
        <arg name="calib" default="false"/>
        <arg name="calib_proc" default="false"/>
        <arg name="thermal_proc" default="false"/>

        <!-- FLIR_A35 Node -->
        <node machine = ”SearchModule” pkg = "flir_gige" type = "flir_gige_node" name
= "$(arg camera)"
        output = "$(arg output)">
        <param name="identifier" type="string" value="$(arg ip_address)"/>
        <param name="camera_name" type="string" value="$(arg camera_name)"/>
```

```
        <param name="calib_url" type="string" value="$(arg calib_url)"/>
        <param name="frame_id" type="string" value="$(arg frame_id)"/>
        <param name="fps" type="double" value="$(arg fps)"/>
        <param name="raw" type="bool" value="$(arg raw)"/>
        </node>

        <node name = "thermal_video_monitor" pkg ="find_victims" type =
"thermal_video_monitor.py" ></node>

        <!-- C920 Node -->
   <node machine = ”SearchModule” name = "video_web_publisher" pkg = "video_web"
type = "my_publisher" args ="1" >
   </node>
   <node name = "video_web_monitor" pkg ="find_victims" type =
"video_web_monitor.py" >
   </node>

   <!-- Microphone Node -->
   <node machine = ”SearchModule” name = "audio_capture" pkg = "audio_capture"
type = "audio_capture" output="screen">
         <param name ="bitrate" value = "128"/>
         </node>

</launch>
```

Anexo G

Programas para la transmisión de video y su visualización

G.1 Programa del nodo video_web_publisher

Este nodo se encuentra en el paquete *video_web*, dentro de la carpeta *sys_nsv_detector*.

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <opencv2/highgui/highgui.hpp>
#include <cv_bridge/cv_bridge.h>
#include <sstream> // for converting the command line parameter to integer

int main(int argc, char** argv)
{
  // Check if video source has been passed as a parameter
  if(argv[1] == NULL) return 1;

  ros::init(argc, argv, "video_web_publisher");
  ros::NodeHandle nh;
  image_transport::ImageTransport it(nh);
  image_transport::Publisher pub = it.advertise("camera/image", 1);

   // Convert the passed as command line parameter index for the video device to an
integer
  std::istringstream video_sourceCmd(argv[1]);
  int video_source;

// Check if it is indeed a number
  if(!(video_sourceCmd >> video_source)) return 1;

  cv::VideoCapture cap(video_source);
  cap.set(CV_CAP_PROP_FRAME_WIDTH, 320);
  cap.set(CV_CAP_PROP_FRAME_HEIGHT, 240);
```

```
// Check if video device can be opened with the given index
  if(!cap.isOpened()) return 1;
  cv::Mat frame;
  sensor_msgs::ImagePtr msg;

  ros::Rate loop_rate(15);

  while (nh.ok()) {
        cap >> frame;
        // Check if grabbed frame is actually full with some content
        if(!frame.empty()) {
        msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", frame).toImageMsg();
        pub.publish(msg);
        cv::waitKey(1);
        }

        ros::spinOnce();
        loop_rate.sleep();
  }
}
```

El tamaño de la imagen que se publica es controlado por los métodos de OpenCV de la clase VideoCapture, que se muestran a continuación

```
  cv::VideoCapture cap(video_source);
  cap.set(CV_CAP_PROP_FRAME_WIDTH, 320);
  cap.set(CV_CAP_PROP_FRAME_HEIGHT, 240);
```

En este caso se está manejando el tamaño mínimo que soporta la cámara web, teniendo la opción de hacerlo más grande ya sea antes de publicarlo o al momento de recibirlo con una herramienta de la biblioteca imutils que se aborda en la sección G.4.

G.2 Programa del nodo *video_web_monitor.py*

Este nodo se encuentra en el paquete *find_victims*, junto con todas las aplicaciones desarrolladas y descritas en el capítulo 5, en el directorio */scripts*. Todos los paquetes que se utilizaron en el sistema de detección de víctimas están la carpeta *system_nsv_detector.*

```
#!/usr/bin/env python
import roslib
import sys
import rospy
import cv2
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

def callback(data):
  global i

  bridge = CvBridge()
  cv_image = bridge.imgmsg_to_cv2(data, "bgr8")
  cv_image = cv2.flip(cv_image,1)
  cv2.imshow("video_web_monitor", cv_image)

  key = cv2.waitKey(1)

  if key == ord("s"):
        i=i+1
        cv2.imwrite('/home/kigs/Pictures/img_thermal_tesis/video_web/image%i.jpg'%i,
cv_image)
        print "screenshot captured"
        print i

def main(args):
  global i

  i = 0
  rospy.init_node('video_web_monitor', anonymous=False)
  image_sub = rospy.Subscriber("camera/image",Image,callback)

  try:
        rospy.spin()
  except KeyboardInterrupt:
```

```
        print("Shutting down")
  cv2.destroyAllWindows()

if __name__ == '__main__':
        main(sys.argv)
```

Este nodo se inicia automáticamente por el archivo launch que arranca la transmisión del video de ambas cámaras y del micrófono estéreo. El nodo se encuentra en el módulo de procesamiento y este puede ser detenido con la instrucción

rosnode kill /video_web_monitor

E iniciado como cualquier otro nodo ingresando en la terminal

rosrun find_victims video_web_monitor.py

G.3 Programa del nodo thermal_publisher

Como se mencionó en el capítulo 5, este nodo fue ligeramente modificado a partir del nodo de KumarRobotics Lab, de la universidad de Pensilvania. Todo el código de este nodo está dividido en varios archivos, contenidos en el paquete *flir_gige* los cuales se muestran a continuación.

Programa del nodo flir_gige_main.cpp

```
#include "flir_gige/flir_gige_node.h"

int main(int argc, char **argv) {
  ros::init(argc, argv, "flir_gige_node");
  ros::NodeHandle nh("~");

  try {
        flir_gige::FlirGigeNode flir_gige_node(nh);
        flir_gige_node.Run();
        ros::spin();
        flir_gige_node.End();
  }
  catch (const std::exception &e) {
        ROS_ERROR("%s: %s", nh.getNamespace().c_str(), e.what());
  }
}
```

Archivo secundario del programa principal, *flir_gige_node.cpp*

```
#include "flir_gige/flir_gige_node.h"

namespace flir_gige {

void FlirGigeNode::Setup(FlirGigeDynConfig &config) {
  flir_gige_ros_.set_fps(config.fps);
  flir_gige_ros_.Reconnect();
  flir_gige_ros_.camera().Configure(config);
  flir_gige_ros_.Start();
}

void FlirGigeNode::Acquire() {
  while (is_acquire() && ros::ok()) {
          const ros::Time time = ros::Time::now();
          flir_gige_ros_.PublishCamera(time);
          flir_gige_ros_.PublishTemperature(time);
          Sleep();
  }
}

}  // namespace flir_gige
```

Archivo de cabecera flir_gige_node.h

```
#ifndef FLIR_GIGE_NODE_H_
#define FLIR_GIGE_NODE_H_

#include "flir_gige/flir_gige_ros.h"
#include "flir_gige/FlirGigeDynConfig.h"
#include "camera_base/camera_node_base.h"

namespace flir_gige {

class FlirGigeNode : public camera_base::CameraNodeBase<FlirGigeDynConfig> {
 public:
  FlirGigeNode(const ros::NodeHandle &nh)
          : CameraNodeBase(nh), flir_gige_ros_(nh) {}

  virtual void Acquire() override;
  virtual void Setup(FlirGigeDynConfig &config) override;
```

```
 private:
  FlirGigeRos flir_gige_ros_;
};

}  // namespace flir_gige

#endif  // FLIR_GIGE_NODE_H_
```

Archivo de cabecera flir_gige_ros.h

```
#ifndef FLIR_GIGE_ROS_H_
#define FLIR_GIGE_ROS_H_

#include "flir_gige/flir_gige.h"
#include "camera_base/camera_ros_base.h"

#include <sensor_msgs/Temperature.h>

namespace flir_gige {

class FlirGigeRos : public camera_base::CameraRosBase {
 public:
  FlirGigeRos(const ros::NodeHandle& nh)
          : CameraRosBase(nh),
          flir_gige_(identifier()),
          nh_(nh),
          temp_pub_(nh_.advertise<sensor_msgs::Temperature>("spot", 1)),
          temp_msg_(new sensor_msgs::Temperature()) {
          SetHardwareId(flir_gige_.display_id());
  }

  FlirGige& camera() { return flir_gige_; }

  void Reconnect() {
          flir_gige_.StopAcquisition();
          flir_gige_.Disconnect();
          flir_gige_.Connect();
  }
  void Start() { flir_gige_.StartAcquisition(); }

  virtual bool Grab(const sensor_msgs::ImagePtr& image_msg,
                    const sensor_msgs::CameraInfoPtr& cinfo_msg) override;
```

```
  void PublishTemperature(const ros::Time& time);

 private:
  FlirGige flir_gige_;
  ros::NodeHandle nh_;
  ros::Publisher temp_pub_;
  sensor_msgs::TemperaturePtr temp_msg_;
};

}  // namespace flir_gige

#endif  // FLIR_GIGE_ROS_H_
```

Archivo de cabecera *flir_gige.h*

```
#ifndef FLIR_GIGE_H_
#define FLIR_GIGE_H_

#include <memory>

#include <PvSystem.h>
#include <PvDevice.h>
#include <PvDeviceGEV.h>
#include <PvDeviceU3V.h>
#include <PvStream.h>
#include <PvStreamGEV.h>
#include <PvStreamU3V.h>
#include <PvBuffer.h>
#include <PvPipeline.h>

#include <sensor_msgs/Image.h>
#include <sensor_msgs/CameraInfo.h>
#include <sensor_msgs/Temperature.h>
#include <flir_gige/FlirGigeDynConfig.h>

#include "flir_gige/planck.h"

namespace flir_gige {

struct FreeDevice {
  void operator()(PvDevice *device) const { PvDevice::Free(device); }
};
```

```
struct FreeStream {
  void operator()(PvStream *stream) const { PvStream::Free(stream); }
};

class FlirGige {
 public:
  FlirGige(const std::string &ip_address);

  const std::string &ip_address() const { return ip_address_; }
  const std::string &display_id() const { return display_id_; }

  void Connect();
  void Disconnect();
  void StartAcquisition();
  void StopAcquisition();
  void Configure(FlirGigeDynConfig &config);
  bool GrabImage(sensor_msgs::Image &image_msg,
                 sensor_msgs::CameraInfo &cinfo_msg);
  bool GrabTemprature(sensor_msgs::Temperature &temp_msg);

 private:
  using PvDevicePtr = std::unique_ptr<PvDevice, FreeDevice>;
  using PvStreamPtr = std::unique_ptr<PvStream, FreeStream>;
  using PvPipelinePtr = std::unique_ptr<PvPipeline>;
  using PvDeviceInfoGEVVec = std::vector<const PvDeviceInfoGEV *>;

  bool FindDevice(const std::string &ip,
                  const PvDeviceInfoGEVVec &dinfo_gev_vec);
  std::string AvailableDevice(const PvDeviceInfoGEVVec &dinfo_gev_vec) const;
  PvDeviceInfoGEVVec GatherGevDevice() const;

  void ConnectDevice();
  void OpenStream();
  void ConfigureStream();
  void CreatePipeline();
  void CacheParams();

  void SetAoi(int *width, int *height) const;
  void SetPixelFormat(bool raw) const;
  void SetNucMode(int nuc) const;
  void DoNuc(bool& nuc) const;
```

```
  //  double GetSpotPixel(const cv::Mat &image) const;

  std::string ip_address_;
  std::string display_id_;
  PvSystem system_;
  const PvDeviceInfo *dinfo_;
  PvDevicePtr device_;
  PvStreamPtr stream_;
  PvPipelinePtr pipeline_;
  PvGenParameterArray *param_array_;
  struct {
          int height;
          int width;
          double B, F, O, R;
          int bit;
  } cache_;
};

}  // namespace flir_gige

#endif  // FLIR_GIGE_H_
```

Archivo de cabecera *camera_ros_base.h*

```
#ifndef CAMERA_NODE_BASE_H_
#define CAMERA_NODE_BASE_H_

#include <thread>
#include <memory>

#include <ros/ros.h>
#include <dynamic_reconfigure/server.h>

#include "camera_base/camera_ros_base.h"

namespace camera_base {

/**
 * @brief The CameraNodeBase class
 * A base class that implements a ros node for a camera
 */
template <typename ConfigType>
class CameraNodeBase {
 public:
  explicit CameraNodeBase(const ros::NodeHandle& pnh)
          : is_acquire_(false), pnh_(pnh), cfg_server_(pnh) {}

  CameraNodeBase() = delete;
  CameraNodeBase(const CameraNodeBase&) = delete;
  CameraNodeBase& operator=(const CameraNodeBase&) = delete;
  virtual ~CameraNodeBase() = default;

  const ros::NodeHandle& pnh() const { return pnh_; }
  bool is_acquire() const { return is_acquire_; }

  /**
   * @brief Run Run the node
   * This will setup the dynamic reconfigure server, this will start the
   * acquisition automatically when the server is initialized
   */
  void Run() {
          cfg_server_.setCallback(
          boost::bind(&CameraNodeBase::ConfigCb, this, _1, _2));
  }
```

```
/**
 * @brief End
 */
void End() { Stop(); }

void Sleep() const { rate_->sleep(); }

/**
 * @brief ConfigCb Dynamic reconfigure callback
 * @param config Config type
 * @param level Reconfigure level, not really used
 * Entering this callback will stop the acquisition thread, do the
 * reconfiguration and restart acquisition thread
 */
void ConfigCb(ConfigType& config, int level) {
        if (level < 0) {
        ROS_INFO("%s: %s", pnh().getNamespace().c_str(),
        "Initializing reconfigure server");
        }
        if (is_acquire()) {
        Stop();
        }
        Setup(config);
        SetRate(config.fps);
        Start();
}

/**
 * @brief Acquire Do acquisition here
 */
virtual void Acquire() = 0;

/**
 * @brief Setup Setup your camera here
 * @param config Config type
 */
virtual void Setup(ConfigType& config) = 0;

private:
void SetRate(double fps) { rate_.reset(new ros::Rate(fps)); }

void Start() {
        is_acquire_ = true;
```

```
        acquire_thread_.reset(new std::thread(&CameraNodeBase::Acquire, this));
  }

  void Stop() {
        if (!is_acquire_) return;
        is_acquire_ = false;
        acquire_thread_->join();
  }

  bool is_acquire_;
  ros::NodeHandle pnh_;
  std::unique_ptr<ros::Rate> rate_;
  std::unique_ptr<std::thread> acquire_thread_;
  dynamic_reconfigure::Server<ConfigType> cfg_server_;
};

}  // namespace camera_base

#endif  // CAMERA_NODE_BASE_H_
```

Archivo de cabecera *camera_ros_base.h*

```
#ifndef CAMERA_ROS_BASE_H_
#define CAMERA_ROS_BASE_H_

#include <ros/ros.h>
#include <sensor_msgs/Image.h>
#include <sensor_msgs/CameraInfo.h>
#include <sensor_msgs/image_encodings.h>
#include <image_transport/image_transport.h>
#include <camera_info_manager/camera_info_manager.h>
#include <diagnostic_updater/publisher.h>
#include <diagnostic_updater/diagnostic_updater.h>

namespace camera_base {

/**
 * @brief getParam Util function for getting ros parameters under nodehandle
 * @param nh Node handle
 * @param name Parameter name
 * @return Parameter value
 */
template <typename T>
```

```
T getParam(const ros::NodeHandle& nh, const std::string& name) {
 T value{};
 if (!nh.getParam(name, value)) {
        ROS_ERROR("Cannot find parameter: %s", name.c_str());
 }
 return value;
}

/**
 * @brief The CameraRosBase class
 * This class implements a ros camera
 */
class CameraRosBase {
 public:
 explicit CameraRosBase(const ros::NodeHandle& pnh,
                const std::string& prefix = std::string())
        : pnh_(pnh),
        cnh_(pnh, prefix),
        it_(cnh_),
        camera_pub_(it_.advertiseCamera("image_raw", 1)),
        cinfo_mgr_(cnh_, getParam<std::string>(cnh_, "camera_name"),
                getParam<std::string>(cnh_, "calib_url")),
        fps_(10.0),
        diagnostic_updater_(pnh_, cnh_),
        topic_diagnostic_(
        prefix.empty() ? "image_raw" : (prefix + "/image_raw"),
        diagnostic_updater_,
        diagnostic_updater::FrequencyStatusParam(&fps_, &fps_, 0.1, 10),
        diagnostic_updater::TimeStampStatusParam(-0.01, 0.1)) {
        cnh_.param<std::string>("frame_id", frame_id_, cnh_.getNamespace());
        cnh_.param<std::string>("identifier", identifier_, "");
 }

 CameraRosBase() = delete;
 CameraRosBase(const CameraRosBase&) = delete;
 CameraRosBase& operator=(const CameraRosBase&) = delete;
 virtual ~CameraRosBase() = default;

 const std::string& identifier() const { return identifier_; }
 const std::string& frame_id() const { return frame_id_; }

 double fps() const { return fps_; }
 void set_fps(double fps) { fps_ = fps; }
```

```
/**
 * @brief SetHardwareId Set hardware id for diagnostic updater
 * @param id harware id
 */
void SetHardwareId(const std::string& id) {
        diagnostic_updater_.setHardwareID(id);
}

/**
 * @brief PublishCamera Publish a camera topic with Image and CameraInfo
 * @param time Acquisition time stamp
 */
void PublishCamera(const ros::Time& time) {
        const auto image_msg = boost::make_shared<sensor_msgs::Image>();
        const auto cinfo_msg =
        boost::make_shared<sensor_msgs::CameraInfo>(cinfo_mgr_.getCameraInfo());
        image_msg->header.frame_id = frame_id_;
        image_msg->header.stamp = time;
        if (Grab(image_msg, cinfo_msg)) {
        // Update camera info header
        cinfo_msg->header = image_msg->header;
        camera_pub_.publish(image_msg, cinfo_msg);
        topic_diagnostic_.tick(image_msg->header.stamp);
        }
        diagnostic_updater_.update();
}

void Publish(const sensor_msgs::ImagePtr& image_msg) {
        const auto cinfo_msg =
        boost::make_shared<sensor_msgs::CameraInfo>(cinfo_mgr_.getCameraInfo());
        // Update camera info header
        image_msg->header.frame_id = frame_id_;
        cinfo_msg->header = image_msg->header;
        camera_pub_.publish(image_msg, cinfo_msg);
        topic_diagnostic_.tick(image_msg->header.stamp);
        diagnostic_updater_.update();
}

/**
 * @brief Grab Fill image_msg and cinfo_msg from low level camera driver
 * @param image_msg Ros message ImagePtr
 * @return True if successful
```

```
  */
 virtual bool Grab(const sensor_msgs::ImagePtr& image_msg,
                   const sensor_msgs::CameraInfoPtr& cinfo_msgs = nullptr) = 0;

private:
 ros::NodeHandle pnh_;
 ros::NodeHandle cnh_;
 image_transport::ImageTransport it_;
 image_transport::CameraPublisher camera_pub_;
 camera_info_manager::CameraInfoManager cinfo_mgr_;
 double fps_;
 diagnostic_updater::Updater diagnostic_updater_;
 diagnostic_updater::TopicDiagnostic topic_diagnostic_;
 std::string frame_id_;
 std::string identifier_;
};

}  // namespace camera_base

#endif  // ROS_CAMERA_BASE_H_
```

Archivo de cabecera para configurar parámetros dinámicos *FlirGigeDyn.cfg*

```
#!/usr/bin/env python
PACKAGE = "flir_gige"

from dynamic_reconfigure.parameter_generator_catkin import *

gen = ParameterGenerator()

gen.add("fps", double_t, 0, "frame per second", 20, 1, 60)
gen.add("raw", bool_t, 0, "Raw 14-bit data", False)
gen.add("nuc_action", bool_t, 0, "Non-uniform image correction", False)
nuc_enum = gen.enum([gen.const("nuc_manual", int_t, 0, "Manual"),
                     gen.const("nuc_automatic", int_t, 1, "Automatic")],
                     "An enum to set nuc mode")
gen.add("nuc_mode", int_t, 0, "Non-uniform correction mode", 0, 0, 1,
edit_method=nuc_enum)
exit(gen.generate(PACKAGE, "flir_gige", "FlirGigeDyn"))
```

Para correr el nodo thermal publisher, es necesario pasar algunos parámetros al momento de de dar la instrucción de que se ejecute, dichos parámetros son:

rosrun flir_gige flir_gige_node ip_address:= 192.168.2.3 camera_name:=flir_a35 camera:= thermal_camera fps:=30 raw:=false

Estos parámetros son declarados en el archivo launch *start_up.launch* así como en el *node.launch* que se encuentra en el paquete *flir_gige*.

G.4 Programa del nodo thermal_video_monitor.py

Se encuentra en el paquete *find_victims,* y también es iniciado por el archivo *start_up.launch* con el objetivo de corroborar que la transmisión del video de la cámara infrarroja está ejecutándose adecuadamente.

```
#!/usr/bin/env python
import roslib
import sys
import rospy
import cv2
import imutils
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

def callback(data):
  global i

  bridge = CvBridge()
  cv_image = bridge.imgmsg_to_cv2(data, "mono8")
  #cv_image = imutils.resize(cv_image, width = 640)
  cv_image = cv2.flip(cv_image,1)
  cv2.imshow("thermal_video_monitor", cv_image)
  key = cv2.waitKey(1)

  if key == ord("s"):
        i=i+1
        cv2.imwrite('/home/kigs/Pictures/img_thermal_tesis/source/thermal_image%i.pn
g'%i, cv_image)
        print "screenshot captured"
        print i

def main(args):
  global i
```

```
 i = 0
 rospy.init_node('thermal_video_monitor')
 image_sub = rospy.Subscriber("thermal_camera/image_raw",Image,callback)

 try:
        rospy.spin()
 except KeyboardInterrupt:
        print("Shutting down")
 cv2.destroyAllWindows()

if __name__ == '__main__':
        main(sys.argv)
```

Adicionalmente, en este nodo se incorporó la opción de duplicar el tamaño del video para visualizarlo, la línea de código que hace esto es

```
cv_image = imutils.resize(cv_image, width = 640)
```

Este método redimensiona la imagen para tener un ancho de 640 píxeles por una longitud proporcional a la relación de aspecto, que en este caso asignará 480 píxeles de alto.

Anexo H

Transformación del espacio de colores monocromático gris al RGB con tonos personalizados

H.1 Programa del nodo color_thermal_monitor_a.py

Este programa se encuentra dentro del paquete *find_victims*, y tiene 5 diferentes versiones que corresponden a cada una de las escalas de tonos personalizadas, diferenciándose únicamente por la última letra que aparece en el nombre antes de .py, por lo que se dispone de *color_thermal_monitor_b.py*, *color_thermal_monitor_c.py*, color_thermal_monitor_d.py y *color_thermal_monitor_e.py*. A continuación se presenta el código general de dichos nodos.

```
#!/usr/bin/env python
import roslib
import sys
import rospy
import cv2
import imutils
import numpy as np
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

offset_w = 5
offset_h = 35

w = 320
h = 256

scale =cv2.imread('/home/kigs/Pictures/img_thermal_tesis/color_palette/palette_a.png')
scale = imutils.resize(scale, width=min(20, scale.shape[1]))
(sc_h, sc_w) = scale.shape[:2]

overlay = np.zeros((h, w, 3), dtype="uint8")
overlay[h - sc_h - offset_h:h - offset_h, w - sc_w - offset_w:w - offset_w] = scale
```

```
def color_presentation (cv_image):
  global i

  cv_image = cv2.flip(cv_image, 1)
  im = cv2.cvtColor(cv_image, cv2.COLOR_GRAY2BGR)

  lut = np.zeros((256, 1, 3), dtype=np.uint8)

  lee_val=[]

readFile=open("/home/kigs/catkin_ws/src/system_nsv_detector/find_victims/scripts/pale
tte_a.txt","r")

  for line in readFile:
        line=line.split()
        line=map(int,line)
        lee_val.append((line))

  readFile.close()

  a=np.array([lee_val])

  lut[:,0,2] = a[0,:,0]
  lut[:,0,1] = a[0,:,1]
  lut[:,0,0] = a[0,:,2]

  im_color = cv2.LUT(im, lut)

  output = im_color.copy()
  output = cv2.addWeighted(overlay, 1, output, 0.7, 0)

  #cv_image = imutils.resize(cv_image, width = 640)
  cv2.imshow("Color_thermal_image_a", output)
  key = cv2.waitKey(1)

  if key == ord("s"):
        i=i+1

cv2.imwrite('/home/kigs/Pictures/img_thermal_tesis/thermal_image_color_a_%i.bmp'%i
, output)
        print "screenshot captured"
        print i
```

```
  return im_color;

def callback(data):

  bridge = CvBridge()
  cv_image = bridge.imgmsg_to_cv2(data, "mono8")
  cv2.flip(cv_image,1)
  color_presentation(cv_image)


def main(args):
  global i

  i = 0
  rospy.init_node('color_monitor_a')
  image_sub = rospy.Subscriber("thermal_camera/image_raw",Image,callback)

  try:
        rospy.spin()
  except KeyboardInterrupt:
        print("Shutting down")
  cv2.destroyAllWindows()

if __name__ == '__main__':
        main(sys.argv)
```

Los demás nodos varían en el código únicamente por el archivo .txt que se lee con la instrucción readFile, ya que cada escala de tonos fue definida en una matriz que se guardó en un archivo de texto plano. También se cambia el nombre de las ventanas así como la dirección de la que guardará las imágenes de cada nodo, al presionar la tecla s.

Anexo I

Segmentación de imagen por gradientes de temperatura y ubicación angular

I.1 Programa del nodo get_angle.py

```
#!/usr/bin/env python
import roslib
import sys
import rospy
import cv2
import math
import copy

from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

min_th = 140
max_th = 255

min_area = 1000
max_area = 50000

img_center = [165,128]
origin = [165,0]

vi = [img_center[0] - origin[0], img_center[1] - origin[1]]

def dotproduct(v1,v2):
        return sum((a*b) for a, b in zip(v1,v2))

def length(v):
        return math.sqrt(dotproduct(v,v))
```

```
def angle(v1,v2):
        return math.acos(dotproduct(v1,v2) / (length(v1)*length(v2)))

def get_angle(centers):

        if centers[0][0] > origin[0]:
        v1 = [centers[0][0] - origin[0], centers[0][1] - origin[1]]
        return angle(v1,vi)
        else:
        v1 = [origin[0] - centers[0][0], centers[0][1] - origin[1]]
        return angle(v1,vi)*-1

def human_scaner(image):
        global i, flag
        centers = []

        clone = copy.copy(image)
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(3,3))
        grayimg = clahe.apply(clone)

        grayimg = cv2.GaussianBlur(grayimg,(21,21),0)

        ret1,th1 = cv2.threshold(grayimg, min_th, 255,cv2.THRESH_BINARY)
        re2,th2 = cv2.threshold(grayimg, max_th, 255, cv2.THRESH_BINARY_INV)

        band_thresh = cv2.bitwise_and(th1, th2)

        cv2.imshow("band", band_thresh)
        cv2.moveWindow("band",450,0)

        contours,                                    hierarchy                                    =
cv2.findContours(band_thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
        contours = sorted(contours, key=cv2.contourArea, reverse = True)[:1]

        for c in contours:
        if cv2.contourArea(c) > min_area and cv2.contourArea(c) < max_area:

        i=i+1
        M = cv2.moments(c)

        if M["m00"] != 0:
        cX = int(M["m10"]/ M["m00"])
        cY = int(M["m01"]/ M["m00"])
```

```
        centers.append((cX,cY))

        cv2.drawContours(clone, [c], -1, 0, 2)
        cv2.circle(clone, (cX, cY), 7, 0, 2)

        cv2.imshow("result", clone)
        cv2.moveWindow("result",0,400)
        cv2.waitKey(1)

        if len(centers) != 0:
                angle_p = get_angle(centers)

                pub = rospy.Publisher('thermal_angle', String, queue_size=1)
                pub.publish(str(angle_p))

                print "angulo en grados"
                print math.degrees(angle_p)
        else:
                print "no hay blobs"

def callback(data):
  global i

  bridge = CvBridge()
  cv_image = bridge.imgmsg_to_cv2(data, "mono8")
  cv_image = cv2.flip(cv_image,1)
  cv2.imshow("Original_Image", cv_image)
  cv2.moveWindow("Original_Image",0,0)
  human_scaner(cv_image)

def main(args):
  global i

  i = 0

  rospy.init_node('get_angle', anonymous=False)
  image_sub = rospy.Subscriber("thermal_camera/image_raw",Image,callback)

  try:
        rospy.spin()
  except KeyboardInterrupt:
        print("Shutting down")
```

```
  cv2.destroyAllWindows()

if __name__ == '__main__':
        main(sys.argv)
```

Anexo J

Programas de entrenamiento y de implementación de los clasificadores en cascada Haar

J.1 Proceso de entrenamiento de clasificadores en cascada

Los clasificadores en cascada Haar que se obtuvieron se entrenaron utilizando los métodos y herramientas de la biblioteca OpenCV, a partir de un banco de imágenes que se obtuvieron con la cámara infrarroja, a continuación se menciona el procedimiento que se siguió para obtener los tres clasificadores de cuerpo, manos y cabeza.

1. Se debe de tener un banco de imágenes positivas (que contengan el objeto que deseamos detectar) y otro de muestras negativas (imágenes que no contiene el objeto de interés a detectar) por cada clasificador. En este trabajo se hizo un banco de 4680 imágenes, 2180 positivas y 2500 negativas, de las cuales 580 fueron utilizadas para hacer el clasificador del cuerpo, 980 el de las manos y 720 el de la cabeza. Estas imágenes estarán disponibles en el servidor del laboratorio de procesamiento digital de señales de la Facultad de Ingeniería, cuya dirección de acceso es http://odin.fi-b.unam.mx/labdsp/

2. Primero se deben de crear un conjunto de muestras positivas a partir del banco de imágenes que se va a trabajar, para esto se debe de contar con una lista de la dirección de las imágenes positivas que adicionalmente contenga, la cantidad de objetos de interés en la imagen y las coordenadas del área comprendida por un rectángulo, como se observa en el siguiente ejemplo.

   img_thermal_image/hand/hand_img_2.jpg 1 50 38 185 178

   En este caso se tiene un solo objeto de interés en la imagen que comprende un área rectangular delimitado por las coordenadas (50, 38) y (185, 178).

   Esto se debe de hacer con cada una de las imágenes positivas con las que se cuenta o como alternativa, las imágenes positivas deben de ser de tal manera que sólo

comprendan el objeto de interés. En nuestro caso de estudio se realizó el código de la sección J.2 para obtener dicha lista de imágenes junto con los datos especificados.

3. Una vez que se tiene la lista de imágenes positivas, se crea una copia de este archivo .txt, para cambiar su terminación a .info para poder crear el archivo vectorial que se utilizará en el entrenamiento.

4. Posteriormente se procede a ocupar la primer herramienta, que es *opencv_createsamples* para crear el archivo *.vec*, se ejecuta en la terminal la siguiente instrucción

   opencv_createsamples -info hand.info -num 980 -w 80 -h 60 -vec hand.vec

   Como se puede observar se tiene que especificar el archivo .info, la cantidad de imágenes positivas declaradas, el tamaño final de las muestras y el archivo de salida.

5. En cuanto respecta a las imágenes negativas, se tiene que generar una lista de todas ellas en un archivo .txt, esto se puede hacer con la siguiente instrucción

   find /img_thermal_image/negative_images -iname ".jpg" > negatives.txt*

   El comando find localiza todos aquellos archivos con terminación .jpg y los ordena en una lista junto con su dirección, para ello se tiene que especificar la carpeta en la que se va a buscar que en este caso es */img_thermal_image/negative_images*. La lista de todas las imágenes encontradas se declara a final de la línea.

6. Con las dos listas de objetos listos (las imágenes negativas en el archivo .txt y las muestras positivas en el .vec) se procede a configurar los parámetros que se utilizarán para entrenar al clasificador con la instrucción *opencv_traincascade,* como se muestra y explica a continuación

   opencv_traincascade -data data_hand -vec hand.vec -bg negatives.txt -numStages 10 -minHitRate 0.999 -maxFalseAlarmRate 0.25 -numPos 980 -numNeg 2500 -w 80 -h 60 -featureType HAAR

   En el campo *-data* se define una carpeta para guardar los archivos que genera el algoritmo durante el entrenamiento, *-numStages* especifica la cantidad de clasificadores que queremos aplicar en cascada, *-minHitRate* define el porcentaje de concordancia que debe de haber con la característica aplcada, *-maxFalseAlarmRate* es el error mínimo absoluto que se debe de tener para finalizar cada clasificador y *-featureType* es el tipo de característica con la que se va a clasificar, que en este caso corresponden a las Haar.

7. Al ejecutar la instrucción de *opencv_traincascade* comienza el entrenamiento de los clasificadores en cascada. El tiempo que se lleve la computadora en realizar el clasificador dependerá del número de etapas declaradas, del porcentaje de error y del grado de coincidencia.

J.2 Programa para selección de ROI en imágenes positivas

Este código permite obtener la lista de imágenes positivas, con su dirección número de objetos de interés en la escena y las coordenadas de la sección rectangular definida por el usuario, dibujando la ventana dando clic derecho del mouse y sin soltar el cursor hasta el punto de la imagen que se quiera abarcar. El programa está en el lenguaje python y para iniciarlo requiere que se le ingrese como parámetro inicial la carpeta de imágenes que va a leer para realizar la lista, como se muestra en la siguiente línea.

python cropped_image_set_size_2.py --images ~/Pictures/img_thermal_image/hand

A continuación se muestra el código de este programa.

```
#!/usr/bin/env python
from imutils import paths
import argparse
import imutils
import cv2
import sys

f = open("test.txt", "a")

def select_objective(event, x, y, flags, param):

	global refPt, cropping, i

	if event == cv2.EVENT_LBUTTONDOWN:
		refPt = [(x, y)]
		cropping = True

	# check to see if the left mouse button was released
	elif event == cv2.EVENT_LBUTTONUP:
		refPt.append((x, y))
		cropping = False
		print refPt
		print len(refPt)
		# draw a rectangle around the region of interest
```

```
            cv2.rectangle(param, refPt[0], refPt[1], (0, 255, 0), 2)
            cv2.putText(param,"Found",(int(refPt[0][0] - 5),
int(refPt[0][1]-5)),cv2.FONT_HERSHEY_SIMPLEX,0.43,(0,0,255),2)
            cv2.imshow("Image", param)


def write_cord(refPt):

    f.write("1")
    f.write(" ")
    f.write(str(refPt[0][0]))
    f.write(" ")
    f.write(str(refPt[0][1]))
    f.write(" ")
    f.write(str(refPt[1][0]))
    f.write(" ")
    f.write(str(refPt[1][1]))
    f.write("\n")

def main(args):

    global refPt

    # construct the argument parse and parse the arguments
    ap = argparse.ArgumentParser()
    ap.add_argument("-i", "--images", required=True, help="path to images
directory")
    args = vars(ap.parse_args())

    cv2.namedWindow("Image")
    cv2.moveWindow("Image",0,0)

    # loop over the image paths
    imagePaths = list(paths.list_images(args["images"]))
    i=0

    for imagePath in imagePaths:

        f.write(format(imagePath))
        f.write(" ")

        print("[INFO]: {}".format(imagePath))
```

```
        image = cv2.imread(imagePath)
        cv2.imshow("Image", image)
        cv2.setMouseCallback("Image", select_objective, image)

        while True:
                key = cv2.waitKey(1) & 0xFF  # 0xFF is for 64 bit computer
                if key == ord("c"):
                        break

                elif key == ord("s"):
                        i=i+1
                        print "iteracion"
                        print i
                        write_cord(refPt)

        #cv2.waitKey(0)

if __name__ == '__main__':
   main(sys.argv)
```

J.3 Programa del nodo haar_detector.py

Este programa aplica el método que utiliza el clasificador en cascada entrenado, mostrando si se realiza o no la detección marcando un rectángulo en la imagen que muestra en la ventana desplegada.

```
#!/usr/bin/env python
import roslib
import sys
import rospy
import cv2
import numpy as np
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

head_cascade = cv2.CascadeClassifier('cascades/nsv_head_cascade.xml')

def head_detection (cv_image):

        cv_image = cv2.flip(cv_image, 1)
```

```
        gray_vid = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)

        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
        gray_vid = clahe.apply(cv_image)

        head = head_cascade.detectMultiScale(gray_vid, 1.2, 5)

        for (x, y, w, h) in head:
                cv2.rectangle(cv_image, (x,y), (x+w, y+h), (0, 255, 0), 2)
                #cv2.putText(param,"Found",(x-5),
                (y-5),cv2.FONT_HERSHEY_SIMPLEX,0.43,(0,0,255),2)
                roi_gray = gray_vid[y:y+h, x:x+w]
                roi_color = cv_image[y:y+h, x:x+w]

        #cv2.imshow("video_stream", cv_image)
        #cv2.waitKey(1)

        return cv_image

def callback(data):

  bridge = CvBridge()
  cv_image = bridge.imgmsg_to_cv2(data, "mono8")
  img = face_detection(cv_image)

def main(args):

  rospy.init_node('hand_detector', anonymous=False)
  image_sub = rospy.Subscriber("thermal/image_raw",Image,callback)

  try:
    rospy.spin()
  except KeyboardInterrupt:
    print("Shutting down")
  cv2.destroyAllWindows()

if __name__ == '__main__':
    main(sys.argv)
```

Anexo K

Programa de detección y extracción de regiones de interés

K.1 Programa del nodo roi_extractor_set_wnd.py

```
#!/usr/bin/env python
import roslib
import copy
import numpy as np
import sys
import rospy
import cv2
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

min_th = 245
max_th = 255

w_width = 60
w_height = 40

def roi_finder (cv_image):
	pre_rois = []

	clone = copy.copy(cv_image)
	clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(3,3))
	grayimg = clahe.apply(clone)

	grayimg = cv2.GaussianBlur(grayimg,(5,5),0)

	ret1,th1 = cv2.threshold(grayimg, min_th, 255,cv2.THRESH_BINARY)
	re2,th2 = cv2.threshold(grayimg, max_th, 255, cv2.THRESH_BINARY_INV)

	band_thresh = cv2.bitwise_and(th1, th2)
```

```
        contours, hierarchy =
cv2.findContours(band_thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        contours = sorted(contours, key=cv2.contourArea, reverse=True)[:3]

        i = 0
        cv_image = cv2.cvtColor(cv_image, cv2.COLOR_GRAY2RGB)

        for c in contours:

        i=i+1
        M = cv2.moments(c)

        if M["m00"] != 0:
        cX = int(M["m10"]/ M["m00"])
        cY = int(M["m01"]/ M["m00"])

        xA = cX - w_width/2
        yA = cY - w_height/2

        xB = cX + w_width/2
        yB = cY + w_height/2

        cv2.rectangle(cv_image, (xA,yA),(xB,yB),(0,0,255),2)
        aux_roi = clone[yA:yB,xA:xB]
        pre_rois.append(aux_roi)

        cv2.putText(cv_image,"ROI_%i"%i,(int(xA - 5),
int(yA-5)),cv2.FONT_HERSHEY_SIMPLEX,0.43,(0,0,255),2)

        if len(pre_rois) != 0:
        cv2.imshow("find_roi", cv_image)
        return pre_rois


def callback(data):
  bridge = CvBridge()
  cv_image = bridge.imgmsg_to_cv2(data, "mono8")
  cv_image = cv2.flip(cv_image,1)
  rois = roi_finder(cv_image)

  if len(rois) != 0:
        cv2.waitKey(1)
```

```
        if len(rois) == 1:
        try:
        image_pub = rospy.Publisher("/rois_detected/rois_1", Image, queue_size=10)
        image_pub.publish(bridge.cv2_to_imgmsg(rois[0],"mono8"))

        except CvBridgeError as e:
        print (e)

        elif len(rois) == 2:
        try:

        image_pub = rospy.Publisher("/rois_detected/rois_1", Image, queue_size=10)
        image_pub.publish(bridge.cv2_to_imgmsg(rois[0],"mono8"))

        image_pub_2 = rospy.Publisher("/rois_detected/rois_2", Image, queue_size=10)
        image_pub_2.publish(bridge.cv2_to_imgmsg(rois[1],"mono8"))

        except CvBridgeError as e:
        print (e)

        elif len(rois) == 3:
        try:

        image_pub = rospy.Publisher("/rois_detected/rois_1", Image, queue_size=10)
        image_pub.publish(bridge.cv2_to_imgmsg(rois[0],"mono8"))

        image_pub_2 = rospy.Publisher("/rois_detected/rois_2", Image, queue_size=10)
        image_pub_2.publish(bridge.cv2_to_imgmsg(rois[1],"mono8"))

        image_pub_3 = rospy.Publisher("/rois_detected/rois_3", Image, queue_size=10)
        image_pub_3.publish(bridge.cv2_to_imgmsg(rois[2],"mono8"))

        except CvBridgeError as e:
        print (e)
  else:
        print ":("

def main(args):

  rospy.init_node('roi_extractor')
  image_sub = rospy.Subscriber("thermal_camera/image_raw",Image,callback)
```

```
  try:
        rospy.spin()
  except KeyboardInterrupt:
        print("Shutting down")
  cv2.destroyAllWindows()

if __name__ == '__main__':
        main(sys.argv)
```

Anexo L

Programa de detección de paredes arteriales

L.1 Programa del nodo canny_edge_detector.py

Este programa tiene 2 copias, en las cuales lo único que cambia es el topic al que se suscriben, que en este caso son los mensajes publicados que corresponden a las ROI publicadas por el nodo *roi_extractor_set_wnd.py*, el objetivo es poder analizar en paralelo más de una ROI.

```
#!/usr/bin/env python
import roslib
import sys
import rospy
import cv2
from std_msgs.msg import String
import numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

def canny_detector(image, sigma):
  # compute the median of the single channel pixel intensities
        v = np.median(image)

# apply automatic Canny edge detection using the computed median
        lower = int(max(0, (1.0 - sigma) * v))
        upper = int(min(255, (1.0 + sigma) * v))
        edged = cv2.Canny(image, lower, upper*3, image, 3, True)

# return the edged image
        return edged

def callback(data):
  bridge = CvBridge()
  cv_image = bridge.imgmsg_to_cv2(data, "mono8")
  clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
  equ = clahe.apply(cv_image)
  grayimg = cv2.GaussianBlur(equ,(5,5),0)
  et1,th1 = cv2.threshold(grayimg,230,255,cv2.THRESH_TOZERO)
```

```
  result = canny_detector(th1,0.33)

  gray = cv2.addWeighted(result,1,grayimg,0.75,1)


  cv2.imshow("Original_Image", cv_image)
  cv2.moveWindow("Original_Image",450,0)
  cv2.imshow("Canny_filter", gray)
  cv2.moveWindow("Canny_filter",450,200)
  cv2.waitKey(1)

def main(args):

  rospy.init_node('canny_filter', anonymous=False)
  image_sub = rospy.Subscriber("/rois_detected/rois_2",Image,callback)

  try:
        rospy.spin()
  except KeyboardInterrupt:
        print("Shutting down")
  cv2.destroyAllWindows()

if __name__ == '__main__':
        main(sys.argv)
```

Anexo M

Extracción y transmisión de audio del micrófono estéreo de la cámara logitech C920

M.1 Programa del nodo audio_capture_stereo

```
#include <stdio.h>
#include <gst/gst.h>
#include <gst/app/gstappsink.h>
#include <boost/thread.hpp>

#include <ros/ros.h>

#include "audio_common_msgs/AudioData.h"

namespace audio_transport
{
 class RosGstCapture
 {
        public:
        RosGstCapture()
        {
        _bitrate = 192;

        std::string dst_type;

        // Need to encoding or publish raw wave data
        ros::param::param<std::string>("~format", _format, "mp3");

        // The bitrate at which to encode the audio
        ros::param::param<int>("~bitrate", _bitrate, 192);

        // only available for raw data
        ros::param::param<int>("~channels", _channels, 2);
```

```
ros::param::param<int>("~depth", _depth, 16);
ros::param::param<int>("~sample_rate", _sample_rate, 16000);

// The destination of the audio
ros::param::param<std::string>("~dst", dst_type, "appsink");

// The source of the audio
//ros::param::param<std::string>("~src", source_type, "alsasrc");

_pub = _nh.advertise<audio_common_msgs::AudioData>("audio", 10, true);

_loop = g_main_loop_new(NULL, false);
_pipeline = gst_pipeline_new("ros_pipeline");
_bus = gst_pipeline_get_bus(GST_PIPELINE(_pipeline));
gst_bus_add_signal_watch(_bus);
g_signal_connect(_bus, "message::error",
        G_CALLBACK(onMessage), this);
g_object_unref(_bus);

// We create the sink first, just for convenience
if (dst_type == "appsink")
{
_sink = gst_element_factory_make("appsink", "sink");
g_object_set(G_OBJECT(_sink), "emit-signals", true, NULL);
g_object_set(G_OBJECT(_sink), "max-buffers", 100, NULL);
g_signal_connect( G_OBJECT(_sink), "new-sample",
        G_CALLBACK(onNewBuffer), this);
}
else
{
printf("file sink\n");
_sink = gst_element_factory_make("filesink", "sink");
g_object_set( G_OBJECT(_sink), "location", dst_type.c_str(), NULL);
}

_source = gst_element_factory_make("alsasrc", "source");
_convert = gst_element_factory_make("audioconvert", "convert");

gboolean link_ok;

if (_format == "mp3"){
_encode = gst_element_factory_make("lamemp3enc", "encoder");
g_object_set( G_OBJECT(_encode), "quality", 2.0, NULL);
```

```
        g_object_set( G_OBJECT(_encode), "bitrate", _bitrate, NULL);

        gst_bin_add_many(  GST_BIN(_pipeline),  _source,  _convert,  _encode,  _sink,
NULL);
        link_ok = gst_element_link_many(_source, _convert, _encode, _sink, NULL);
        } else if (_format == "wave") {
        GstCaps *caps;
        caps = gst_caps_new_simple("audio/x-raw-int",
                        "channels", G_TYPE_INT, _channels,
                        "width",        G_TYPE_INT, _depth,
                        "depth",        G_TYPE_INT, _depth,
                        "rate",         G_TYPE_INT, _sample_rate,
                        "signed",   G_TYPE_BOOLEAN, TRUE,
                        NULL);

        g_object_set( G_OBJECT(_sink), "caps", caps, NULL);
        gst_caps_unref(caps);
        gst_bin_add_many( GST_BIN(_pipeline), _source, _sink, NULL);
        link_ok = gst_element_link_many( _source, _sink, NULL);
        } else {
        ROS_ERROR_STREAM("format must be \"wave\" or \"mp3\"");
        exitOnMainThread(1);
        }
        /*}
        else
        {
        _sleep_time = 10000;
        _source = gst_element_factory_make("filesrc", "source");
        g_object_set(G_OBJECT(_source), "location", source_type.c_str(), NULL);

        gst_bin_add_many( GST_BIN(_pipeline), _source, _sink, NULL);
        gst_element_link_many(_source, _sink, NULL);
        }
        */

        if (!link_ok) {
        ROS_ERROR_STREAM("Unsupported media type.");
        exitOnMainThread(1);
        }

        gst_element_set_state(GST_ELEMENT(_pipeline), GST_STATE_PLAYING);

        _gst_thread = boost::thread( boost::bind(g_main_loop_run, _loop) );
```

```
}

~RosGstCapture()
{
g_main_loop_quit(_loop);
gst_element_set_state(_pipeline, GST_STATE_NULL);
gst_object_unref(_pipeline);
g_main_loop_unref(_loop);
}

void exitOnMainThread(int code)
{
exit(code);
}

void publish( const audio_common_msgs::AudioData &msg )
{
_pub.publish(msg);
}

static GstFlowReturn onNewBuffer (GstAppSink *appsink, gpointer userData)
{
RosGstCapture *server = reinterpret_cast<RosGstCapture*>(userData);
GstMapInfo map;

GstSample *sample;
g_signal_emit_by_name(appsink, "pull-sample", &sample);

GstBuffer *buffer = gst_sample_get_buffer(sample);

audio_common_msgs::AudioData msg;
gst_buffer_map(buffer, &map, GST_MAP_READ);
msg.data.resize( map.size );

memcpy( &msg.data[0], map.data, map.size );

server->publish(msg);

return GST_FLOW_OK;
}

static gboolean onMessage (GstBus *bus, GstMessage *message, gpointer
userData)
```

```
        {
        RosGstCapture *server = reinterpret_cast<RosGstCapture*>(userData);
        GError *err;
        gchar *debug;

        gst_message_parse_error(message, &err, &debug);
        ROS_ERROR_STREAM("gstreamer: " << err->message);
        g_error_free(err);
        g_free(debug);
        g_main_loop_quit(server->_loop);
        server->exitOnMainThread(1);
        return FALSE;
        }

        private:
        ros::NodeHandle _nh;
        ros::Publisher _pub;

        boost::thread _gst_thread;

        GstElement *_pipeline, *_source, *_sink, *_convert, *_encode;
        GstBus *_bus;
        int _bitrate, _channels, _depth, _sample_rate;
        GMainLoop *_loop;
        std::string _format;
  };
}

int main (int argc, char **argv)
{
  ros::init(argc, argv, "audio_capture");
  gst_init(&argc, &argv);

  audio_transport::RosGstCapture server;
  ros::spin();
}
```