Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Tesis de Licenciatura

Estrategias para el seguimiento de personas con robots aéreos no tripulados

Que para obtener el título de:

Ingeniero en Ciencias de la Computación

Presenta:

Fabiola Guevara Soriano

Asesor:

Dr. Abraham Sánchez López

Movis Research Group

Puebla, México

Junio 2017

Dedicatoria

Agradecimientos

Índice General

Índice de Figuras

Índice de Tablas

Abreviaturas

Abreviatura	**Término**
MAV	Micro Aerial Vehicle (Micro Vehículo aéreo)
UAV	Unnamed Aerial Vehicle (Vehículo Aéreo No Tripulado)
FPS	Frames Per Second (Imágenes por segundo)
UDP	User datagram Protocol
TCP	Transmission Control Protocol
FTP	File transfer Protocol(Protocolo de transferencia de archivos)
MSV	Máquina de Soporte Vectorial
$SIFT$	Scale-Invariant Feature Transform
HOG	Histogram of Oriented Gradients (Histograma de Gradientes Orientados)

Capítulo 1

Introducción

Actualmente la robótica se ha convertido en un aspecto fundamental en la vida cotidiana de la sociedad moderna, por lo tanto su investigación y aplicación se incrementan cada vez más. El objetivo principal de la robótica es la construcción de dispositivos que funcionen de manera automática y que realicen trabajos arduos o incluso imposibles para los seres humanos. Dichos dispositivos se pueden clasificar de acuerdo a varios criterios, por ejemplo, de acuerdo al ambiente en el que se desarrollan, por lo tanto un robot se puede clasificar como marino, terrestre, aéreo, espacial, etc.

Durante los últimos 30 años el número de robots aéreos en el mundo han crecido de manera exponencial. En 2010 hubo predicciones de una industria de 10 mil millones. La mayor parte de las aplicaciones están en el sector militar, el sector civil y por supuesto el sector DIY donde hay millones de usuarios que desarrollan todo tipo de plataformas de software y hardware para este tipo de robots.
Los robots aéreos son dispositivos que se encuentran cada vez más presentes en nuestro entorno, gracias a la miniaturización de los componentes electrónicos que los conforman, haciendo posible utilizarlos en situaciones variadas como pueden ser aplicaciones de servicios para: localizar personas, filmación de eventos o inspección de zonas de difícil acceso, como es el caso de la inspección de los reactores nucleares dañados en Fukushima en 2011, o la inspección sobre la ciudad fantasma de Chernobyl en 2014, que sufrió las consecuencias del desastre nuclear de 1986, entre otras.

Los micro-vehículos aéreos son aeronaves no tripuladas con un tamaño que va en el orden de los centímetros, los cuales pueden o no ser autónomos. En esta tesis se hará referencia a ellos por las siglas MAV - del inglés Micro Aerial Vehicle - ya que son las más utilizadas en la literatura. En años recientes, los MAVs han llegado a ser una herramienta importante no sólo en el dominio militar, sino también en ambientes civiles. Los MAV son propulsados por cuatro rotores y pueden ser controlados alternando la velocidad entre ellos. Una cámara a bordo se utiliza para capturar y transmitir imágenes de la vista de primera persona durante el vuelo, en tiempo real, de manera inalámbrica a una estación base.

El seguimiento de objetos con MAVs ha creado un gran interés entre la comunidad científica en los últimos años, sobre todo por sus aplicaciones como la de inspección de objetos (grabación) que se mueven, sin la necesidad de controlar el MAV desde una estación remota. Es por ello que en este documento se describe el objetivo principal de este trabajo, que es el del seguimiento mediante un MAV y la metodología con la que se puede lograr.

En el laboratorio MOVIS (MOvimiento y VISión) tenemos experiencia importante en el desarrollo de algoritmos de planificación de movimientos para diferentes tipos de robots. Igualmente, hemos propuesto algoritmos de navegación que involucran el uso de robots de tipo móvil, humanoide y recientemente drones.

1.1. Objetivos del proyecto

1.1.1. Objetivo general

Proponer estrategias que permitan realizar el seguimiento de personas (tracking) en ambientes exteriores, utilizando un MAV (en este caso, un Parrot Ar.Drone 2.0). Dichas estrategias deberán considerar que se dispone de un algoritmo robusto de planificación de movimientos, así como de una o varias estrategias para el seguimiento de personas.

1.1.2. Objetivos específicos

- Estudiar otras estrategias propuestas en la comunidad de visión artificial, para el caso de seguimiento de personas.
- Revisar las herramientas propuestas en la comunidad de robótica para el uso de robots (ROS, GAZEBO, etc.).
- Acoplar a la estrategia de seguimiento de personas antes propuesta (filtro de partículas e histograma de gradientes orientados), un algoritmo de planificación de movimientos.
- Comparar las estrategias de seguimiento y elegir la más robusta.

1.2. Contribución de la tesis

En el presente trabajo de tesis la contribución más importante, será la que corresponde al estado del arte del seguimiento de personas con robots aéreos no tripulados. Específicamente se hace una aportación en lo referente a la selección de parámetros para la detección de personas con el algoritmo de histogramas de gradientes orientados y en el seguimiento de

personas con el filtro de partículas. Además se contribuye con la propuesta de una estrategia para el seguimiento de personas con drones, utilizando filtro de partículas, histograma de gradientes orientados y un algoritmo para la planificación de movimientos para el drone. Finalmente se tendrán una serie de simuladores propios para futuros proyectos, enfocados a las áreas de visión por computadora, seguimiento de personas y robótica aérea, todos ellos orientados a pruebas en tiempo real.

1.3. Organización de la tesis

La estructura de este documento de tesis está dividida en 6 capítulos, detallados a continuación.

- **Capítulo 2**: En este capítulo, se explica en forma sucinta una descripción de los conceptos fundamentales necesarios para la comprensión del tema a desarrollar, prescindiendo de muchos detalles en aras a la claridad. Se presenta el estado del arte haciendo énfasis en los algoritmos empleados, donde se describen las técnicas necesarias para la integración con un vehículo aérero no tripulado. Los temas aquí descritos se plantearán con más detalle en los capítulos posteriores.

- **Capítulo 3**: Se describe explícitamente los conceptos teóricos que conforman los métodos utilizados para el desarrollo del sistema propuesto en este trabajo de tesis, y a los que se ha recurrido en el proceso. Se analiza en detalle cada fase de la implementación de los algoritmos desarrollados que son: el algoritmo HOG para la detección de personas, el algoritmo de aprendizaje con entrenamiento de la máquina de Soporte Vectorial y el algoritmo del filtro de partículas para seguimiento de personas.

- **Capítulo 4**: Se presentan las características y especificaciones de hardware y sofware que permitieron la realización de la propuesta, además se detallan las caracterísiticas físicas del MAV utilizado para este trabajo de tesis, se mencionan los conceptos relativos a la adquisición de imágenes y captura de video, se detalla la manera en que se establece la comunicación entre la estación de trabajo y el MAV utilizado. Para validar más eficientemente el sistema, las pruebas fueron realizadas tomando en cuenta diferentes valores en los factores de interés. Posteriormente se efectuá el análisis y evaluación de los resultados obtenidos.

- **Capítulo 5**: Se describe el proceso de integración de los algoritmos desarrollados en el capítulo 2 con el AR.Drone 2.0. Se presenta la fase de pruebas del funcionamiento que tiene por finalidad la evaluación del desempeño del seguidor final. El objetivo de este capítulo consiste en dar a conocer los logros alcanzados en cuanto al seguimiento de personas en tiempo real por el ARDrone 2.0, con las técnicas aplicadas sobre el MAV descritas en el Capítulo 2.

- **Capítulo 6**: Se exponen las conclusiones a las que se llegaron, tomando en cuenta las restricicones del proyecto. También se presenta la perspectiva y discusión de trabajos futuros que pudieran derivar de la continuación del desarrollo del proyecto en este campo de investigación.

Capítulo 2

Marco teórico

2.1. Visión artificial

La visión es uno de los mecanismos sensoriales de percepción más importantes que tiene el ser humano y muchos de los organismos biológicos; se utiliza para percibir el entorno que nos rodea y poder interactuar eficientemente con él. En este sentido es de gran importancia el poder detectar los objetos que son de nuestro interés por medio de su forma, color, relieve, dimensiones, distancia a la que se encuentra, etcétera.

Por su parte, la visión artificial o por computadora es una disciplina mediante la cual se dota a una máquina de la capacidad de percibir el mundo que le rodea, para deducir la estructura y las propiedades del mundo tridimensional a partir de una o más imágenes bidimensionales.

De acuerdo con Ballard y Brown, la visión por computadora se refiere a la construcción de descripciones explícitas y significativas de los objetos físicos a partir de imágenes [9]. Es decir, la capacidad de percibir permite a una máquina extraer y analizar información espectral, espacial y temporal de los distintos objetos contenidos en una imagen. Mientras que la información espectral incluye frecuencia (color) e intensidad (tonos de gris), la información espacial se refiere a aspectos como forma y posición (una, dos y tres dimensiones) y la información temporal comprende aspectos estacionarios (presencia y/o ausencia) y dependientes del tiempo (eventos, movimientos, procesos).

La tarea de reconocimiento de objetos es de particular interés para esta investigación, sabiendo que esta tarea involucra los procesos de extracción de características y clasificación. Considerando lo mencionado, resulta evidente, y así lo expresan Pietikäinen et al. [10], que para obtener un buen desempeño en esta tarea es necesario contar con descriptores robustos y un clasificador de buena calidad.

Ahora se puede definir formalmente al reconocimiento de objetos como la tarea, dentro de un sistema de visión artificial, de encontrar e identificar objetos en una imagen o una secuencia de video. Existe una gran cantidad de áreas y/o aplicaciones que hacen uso del recono-

cimiento de objetos, por ejemplo: clasificación de frutos, detección de rostros, detección de personas, reconocimiento de rostros, detección de placas, seguimiento automático de objetos, reconocimiento automático de señales de tránsito, entre muchas más [11].

2.1.1. Reconocimiento de objetos

El reconocimiento de objetos es la tarea que consiste en encontrar e identificar objetos en una imagen o secuencia de vídeo. Los humanos reconocemos una multitud de objetos en imágenes con poco esfuerzo, a pesar del hecho que la imagen del objeto puede variar un poco en diferentes puntos de vista, en diferentes tamaños o escalas e incluso cuando son trasladados o rotados. Los objetos pueden ser reconocidos cuando están parcialmente obstruidos desde una vista. No obstante esta tarea es un desafío para los sistemas de visión por computadoras.

2.1.1.1. Histograma de Gradientes Orientados

El algoritmo fue introducido en 2005 por Navneet Dalal y Bill Triggs [12] se usa en ámbitos como la visión por computadora y el procesamiento de imágenes con el propósito de detectar objetos en una imagen. La esencia de dicho algoritmo es que la forma de un objeto en una imagen puede ser descrito por medio de la distribución de los gradientes.

El objetivo principal de esta técnica es la extracción de características de una imagen, las cuales identifiquen la silueta de una persona o algún otro objeto. Las características son extraídas teniendo en cuenta los bordes. El proceso de obtener información de los bordes presentes en una imagen, se consigue calculando los gradientes y las orientaciones de los píxeles.

La idea principal es que la apariencia y la forma de los objetos pueden ser caracterizados de mejor manera por la distribución de gradientes de intensidad locales, inclusive sin conocimiento del gradiente correspondiente. Esto se hace mediante la división de la imagen en varias regiones pequeñas (celdas), cada una de las cuales tiene un histograma de las direcciones de su gradiente o también llamada orientación de borde de los píxeles de la celda.

Las entradas de histograma combinadas forman la representación. Para un mejor desempeño cuando no se tienen condiciones de iluminación o sombras, se recomienda hacer una normalización en contraste antes de procesarlas. Esto se hace acumulando una medida local de "energía" sobre regiones más grandes (bloques) y utilizando los resultados para normalizar todas las celdas en el bloque. Este descriptor de normalización se conoce como descriptor de Histograma de Gradientes Orientados (HOG). Dividiendo la imagen con una densa maya (que de hecho se superponen) de descriptores HOG y utilizando el vector de características

combinadas en una Máquina de Soporte Vectorial (MSV) convencional, es posible realizar la detección de seres humanos.

2.1.2. Máquinas de Soporte Vectorial (MSV)

La teoría de las Máquinas de Soporte Vectorial (SVM por su nombre en inglés Support Vector Machine) fue desarrollada por Vapnik basado en la idea de minimización del riesgo estructural (SRM). Es un método de aprendizaje automático que a partir de los datos de entrada, reconoce patrones en estos datos y resuelve problemas de clasificación binaria. Algunas de las aplicaciones de clasificación o reconocimiento de patrones son: reconocimiento de firmas, reconocimiento de imágenes como rostros y categorización de textos, etc.

A diferencia de las Redes Neuronales Artificiales (RNA) que utilizan durante la fase de entrenamiento, el principio de Minimización del Riesgo Empírico (ERM), las MSV se basan en el principio de Minimización del Riesgo Estructural (SRM), la cual ha mostrado un mejor desempeño que el ERM, ya que las Máquinas de Soporte Vectorial minimizan un límite superior al riesgo esperado a diferencia del ERM que minimiza el error sobre los datos de entrenamiento (Vapnik, 2000). La MSV mapean los puntos de entrada a un espacio de características de una dimensión mayor, para luego encontrar el hiperplano que los separe y maximice el margen entre las clases.

Pertenecen a la familia de clasificadores lineales puesto que inducen separadores lineales o hiperplanos en espacios de características de muy alta dimensionalidad (introducidos por funciones núcleo o kernel) con un sesgo inductivo muy particular. La formulación matemática de las Máquinas de Soporte Vectorial varía dependiendo de la naturaleza de los datos; es decir, existe una formulación para los casos lineales y, por otro lado, una formulación para casos no lineales.

Es importante tener claro que, el método crea un clasificador discriminatorio que se define mediante un hiperplano que separa los datos. Estos datos deben estar etiquetados como positivos y como negativos para que la salida del método sea un hiperplano óptimo que categorizará nuevos datos de entrada.

Las MSV han sido desarrolladas como una técnica robusta para clasificación y regresión aplicado a grandes conjuntos de datos complejos con ruido; es decir, con variables inherentes al modelo que para otras técnicas aumentan la posibilidad de error en los resultados pues resultan difíciles de cuantificar y observar [7].

2.1.3. Seguimiento de objetos

El seguimiento de objetos es el proceso de estimar en el tiempo la ubicación de uno o más objetos móviles mediante el uso de una cámara. La rápida mejora en cuanto a calidad y resolución de los sensores de imagen, juntamente con el dramático incremento en cuanto a la potencia de cálculo en la última década, ha favorecido la creación de nuevos algoritmos y aplicaciones mediante el seguimiento de objetos. El seguimiento de objetos puede ser un proceso lento debido a la gran cantidad de datos que contiene un vídeo. Además, la posible necesidad de utilizar técnicas de reconocimiento de objetos para realizar el seguimiento incrementa su complejidad.

2.1.3.1. Filtro de Partículas

Los Filtros de Partículas son una implementación alternativa no-paramétrica de los filtros bayesianos (a diferencia, de los filtros Kalman, que son paramétricos). La idea principal en los filtros de partículas es representar una estimación de un vector de estados en un momento t, mediante un conjunto de vectores de estados muestreados en el momento $t-1$. Estas muestras se denominan partículas, y representan una hipótesis de cómo es el vector de estados en el momento t.

El filtro de partículas bajo el enfoque de seguimiento de objetos, también conocido como el Algoritmo de condensación y el Método de Monte Carlo, utiliza un gran número de partículas para explorar el espacio de estados. Cada partícula representa una posición hipotética de un objetivo en el espacio de estados. Inicialmente, las partículas están uniformemente distribuidas de manera aleatoria en el espacio de estados, y por cada trama subsiguiente, el algoritmo recorre los siguientes pasos iterativamente.

1. Deriva determinística: las partículas se mueven de acuerdo a un modelo de movimiento determinista. (Se utilizó un modelo de movimiento de velocidad constante.)

2. Actualizar la función de densidad de probabilidad (FDP): determinar la probabilidad para cada ubicación de las partículas nuevas.

3. Remuestrear las partículas: el 90 % de las partículas son remuestreadas con reemplazo, para que de esta manera la probabilidad de escojer una muestra de partículas sea igual a la FDP en ese momento; El 10 % restante de las partículas se distribuyen aleatoriamente a lo largo del espacio de estados.

4. Partículas difusas: las partículas se mueven a una pequeña distancia en el estado del espacio bajo un movimiento browniano.

Esto da resultado en partículas que se congregan en regiones de alta probabilidad y se dispersan de otras regiones. Por lo tanto, la densidad de partículas indica los estados “objetivo”

(candidatos) más probables. Las principales ventajas del enfoque del filtro de partículas para la localización y el seguimiento son su escalabilidad (el requerimiento computacional varía linealmente dependiendo del número de partículas) y su capacidad de tratar múltiples hipótesis (por lo tanto es más fácilmente recuperarse de los errores de seguimiento). El filtro de partículas fue utilizado para diversas razones. Proporciona una eficiente búsqueda de un objetivo en un espacio de estados multidimensional. Reduce el problema de búsqueda a un problema de verificación [13].

2.1.4. Segmentación

Segmentar una imagen digital significa dividirla en zonas disjuntas e individualizadas. Es decir, consiste en diferenciar en una imagen los diversos objetos y sus límites respecto al fondo donde se ecuentran, que puede ser una tarea más o menos compleja.
Al final de la etapa de segmentación, se tienen que conocer perfectamente los objetos que hay para extraer las características propias de cada uno de ellos. Además, cada píxel de la imagen tiene que tener una etiqueta que los defina, de forma que simplemente por agrupación de puntos con la misma etiqueta y conectados espacialmente, se pueda determinar la lista de objetos (estos objetos son realmente zonas o regiones individualizadas dentro de la imagen, ya que un objeto, en el sentido estricto de la palabra, puede estar repartido en varias regiones diferentes dentro de la imagen obtenida).

Realmente, la etapa de segmentación es crucial para el reconocimiento de formas pudiéndose complicar o simplificar enormemente según sea la escena más o menos compleja. Por ejemplo, puede ocurrir que las escenas se compliquen más o menos, dependiendo del conjunto de objetos que se tengan que reconocer y de la disposición de estos en el entorno. Que los objetos tengan niveles de intensidad uniformes o cuasiuniformes, puede permitir distinguirlos del fondo fácilmente [14].

La etapa de segmentación es crucial en el reconocimiento debido a que la consecuencia de ésta etapa es tener los objetos perfectamente ubicados en la escena. Se ha generado una gran cantidad de trabajos que presentan diversas técnicas, modelos y algoritmos. Éstas técnicas se dividen en cuatro grandes grupos [15]:

- Técnicas de segmentación basadas en los valores de píxel.
- Técnicas de segmentación basadas en el área.
- Técnicas de segmentación basadas en orillas.
- Técnicas de segmentación basadas en la física.

2.1.5. OpenCV

OpenCV (Open Source Computer Vision) es una librería para visión por computador, de código abierto, que está escrita en lenguajes C y C++. Fue diseñada para obtener alta eficiencia computacional enfocándose en aplicaciones en tiempo real. La librería cuenta con más de 500 funciones que abarcan muchas áreas en visión por computador, incluyendo, manufactura, inspección de productos, imágenes médicas, seguridad, interfaz para el usuario, calibración de cámara, visión estéreo y robótica, entre otras. Además incluye una librería de aprendizaje de máquina de propósito general.

OpenCV surgió como una iniciativa de investigación de Intel para avanzar en aplicaciones intensivas de CPU. Al principio los objetivos de OpenCV eran entre otros, avanzar en investigaciones de visión por computador al brindar un código abierto y optimizado para infraestructura de visión básica; diseminar el conocimiento de visión por computador al entregar una infraestructura común sobre la que los desarrolladores pudieran trabajar, haciéndolo más fácil de leer y de escribir; por último, se buscaba avanzar en aplicaciones comerciales basadas en visión por computador, haciendo que el código fuera portable, con rendimiento optimizado y sin costo.
OpenCV se estructura en cinco componentes principales, cuatro de los cuales se muestran y se describen en la Figura **2-1**.

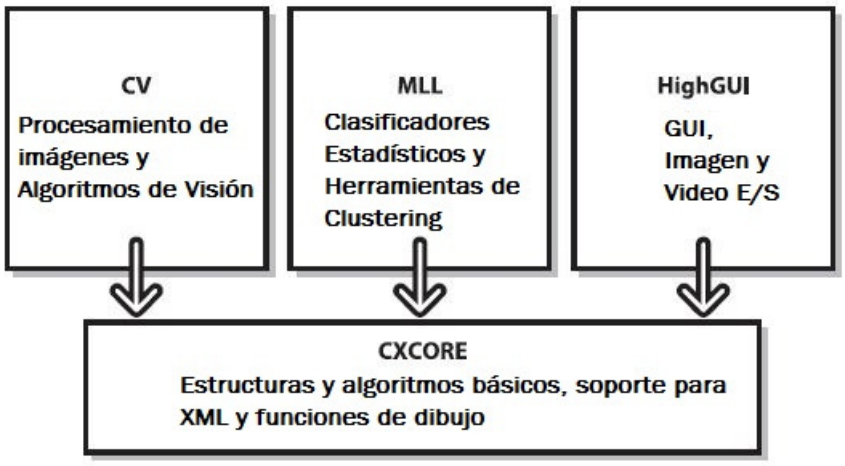


Figura 2-1: Estructura básica de OpenCV. Tomada de [1].

OpenCV contiene funciones para el procesamiento básico de imágenes y los algoritmos de visión por computador de más alto nivel; MLL es la librería de aprendizaje de máquina, que incluye, varios clasificadores estadísticos y herramientas de clustering; HighGUI contiene rutinas Entrada/Salida y funciones para almacenar y cargar videos e imágenes y CXCore contiene las estructuras de datos básicas y el contenido. Otro componente es el CvAux, que

contiene áreas extintas (Reconocimiento Facial) y algoritmos experimentales (Segmentación de Fondo y Primer Plano).

OpenCV fue diseñado para ser de plataforma múltiple, ya que tiene versiones para GNU/Linux, Mac OS X y Windows. Al ser OpenCV software libre, se da a los usuarios la libertad para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De una forma más precisa, los usuarios del software tienen cuatro libertades:

- La libertad de usar el programa con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa y adaptarlo a sus necesidades (libertad 1). El acceso al código fuente es una condición previa para esto.
- La libertad de distribuir copias (libertad 2).
- La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie. (libertad 3). El acceso al código fuente es un requisito previo para esto [1].

2.2. Vehículos aéreos no tripulados

Los MAV, se han desarrollado en los últimos años con mayor frecuencia en los ámbitos científico y comercial. La gran popularidad que han tenido los convierte en una de las tendencias de desarrollo más recientes y han sido un área de investigación muy activa.

Esta nueva plataforma de investigación, ha sido crucial en el desarrollo histórico y científico de la sociedad moderna. Los MAV gozan en la actualidad de gran popularidad y aceptación a nivel mundial debido a las ventajas que brindan para el cumplimiento de misiones en lugares de difícil acceso, de reconocimiento, supervisión y vigilancia. Estos pueden usarse desde importantes aplicaciones civiles como son las de rescate y salvamento, estudio de ecosistemas, supervisión de líneas eléctricas, oleoductos, hasta aplicaciones en el campo militar, ya sea para tareas de supervisión y espionaje o como portadores de armas de combate.

Los MAV son aeronaves capaces de volar sin necesidad de un piloto humano abordo que las controle. Pueden ser remotamente controlados, volar de forma autónoma basada en planes de vuelo preprogramados mediante el uso de sistemas de control complejos. Además, pueden estar controlados por una estación a bordo que realiza toda la toma de decisiones de forma automática o por operadores que con el uso de un radio control pueden realizar acciones de forma manual como son el despegue y el aterrizaje. La autonomía se relaciona con los algoritmos de control que poseen estos vehículos para responder de manera satisfactoria al

encontrarse en eventos inesperados o aleatorios, durante sus misiones de vuelo [16].

Actualmente la tecnología para la fabricación de MAV totalmente autónomos se encuentra en pleno desarrollo; ésta recae en las siguientes categorías:

- **Fusión de Sensores:** Combinar información desde diversos dispositivos, para ser utilizados en el centro del vehículo.
- **Comunicaciones:** Realizar la coordinación entre múltiples agentes en la presencia de información que sea imprecisa e incompleta.
- **Planeamiento de Vuelos:** Determinar el camino óptimo que deberá seguir el vehículo, a la vez de que cumple con ciertas restricciones impuestas por los objetivos y la misión, tales como obstáculos o consumo de combustibles.
- **Generación de Trayectorias:** Determinar la maniobra óptima para llegar o seguir a un objetivo determinado.
- **Regulación de Trayectorias:** Las estrategias de control especificas requeridas para cumplir con las restricciones.
- **Programación y asignación de tareas:** Determinar la distribución óptima de tareas entre un grupo de agentes, satisfaciendo las restricciones en tiempo y equipo.
- **Tácticas cooperativas:** Formular una secuencia óptima y una distribución especial de actividades entre los diferentes agentes, con el fin de maximizar las probabilidades de éxito de alguna misión dada.

El helicóptero de 4 rotores, cuatrirrotores o más comúnmente llamado cuadricóptero es un tipo de MAV, el cual tiene una configuración en particular, que consta de 4 rotores colocados en las puntas de una estructura en forma de cruz. El cuadricóptero ha recibido especial atención por parte de los investigadores debido a su facilidad de control comparado con otros vehículos aéreos y por sus capacidades para realizar maniobras agresivas, i.e. giro completo (flip, en inglés). En la figura **2-2** se puede observar el ejemplo de la anatomía de un MAV.

Figura 2-2: Vehículo aéreo no tripulado Bebop de Parrot [2].

Este tipo de aeronaves poseen grandes ventajas frente a los vehículos tripulados, estos no ponen en riesgo vidas humanas lo cual contribuye a que sus límites vayan más allá de las capacidades y resistencia humanas, es por ello que no están sujetos en su diseño a satisfacer necesidades de ergonomía. Al no haber la necesidad de una cabina que proteja al piloto, el peso y tamaño del dispositivo son menores brindándole un acceso a sitios inalcanzables para vehículos tripulados. Esta característica también reduce el consumo de la aeronave, teniendo un menor impacto ambiental, producen menos contaminación por gases, ruido y consumo energético, lo cual impacta directamente en un menor costo de mantenimiento. Otra de las ventajas que estas aeronaves poseen es la capacidad de realizar despegues horizontales y verticales (con o sin asistencia).

2.2.1. Los MAVs para el seguimiento de objetos

Los MAV como se mencionó anteriormente, han sido enfocados a muchas y muy diversas aplicaciones, dentro de las que podemos encontrar la planificación de movimientos para un vuelo autónomo con métodos pre-programados usados para el auto guiado, las cuales se llevan a cabo con cámaras y se basan en algoritmos de procesamiento de video. Unas de las técnicas más utilizadas en estas aplicaciones son las relacionadas con el concepto de tracking o seguimiento automático de objetivos en secuencias de video. El video tracking es la capacidad de estimar la posición de uno o múltiples objetos de interés en la imagen y seguirlos en el tiempo, es decir, en la secuencia de imágenes posteriores a su detección/inicialización.

El tracking por sí mismo es uno de los ámbitos de procesamiento de señales de video en los que más se está trabajando en la comunidad científica, ya que proporciona una base de información que puede ser muy útil para diferentes aplicaciones de nivel superior. La posi-

bilidad de montar cámaras en vehículos como pueden ser los MAVs ofrece una ampliación del escenario de aplicación de las técnicas de tracking. Un claro ejemplo son los sistemas de videovigilancia que tratan de monitorizar usuarios en áreas amplias y complejas, que a priori necesitan varias cámaras, y se solventan mediante una única cámara móvil en un MAV. Otras aplicaciones del tracking en MAVs fuera de la videovigilancia son por ejemplo las citadas anteriormente, la búsqueda y localización de humanos perdidos en zonas remotas de difícil acceso, o la monitorización del tráfico.

Se puede concluir que el uso de MAVs con cámaras tiene un potencial de aplicación enorme y que por lo tanto, el correcto funcionamiento de los algoritmos de procesamiento de video, y más concretamente los de video tracking, en estos entornos cobra una gran relevancia. Sin embargo, es necesario un trabajo de investigación en profundidad orientado al análisis de las características que deben de presentar los algoritmos de tracking que se vayan a emplear en estas condiciones. El objetivo es recopilar y analizar las propiedades, tanto de la plataforma de captura, esto es, las cámaras de los MAVs, como de los trackers, y seleccionar aquellos que mejor se adapten para su posterior evaluación.

Capítulo 3

Especificación y uso de algoritmos

3.1. Histograma de Gradientes Orientados (HOG)

Uno de los objetivos de este trabajo de tesis es proponer un modelo de reconocimiento de personas para su posterior integración con el algoritmo de seguimiento que, a su vez será utilizado en un MAV. Se ha definido el algoritmo HOG como base para el cumplimiento de dicho objetivo, la elección de este método para llevar a cabo la detección de personas se basa en que destaca por su robustez frente a diferentes condiciones de iluminación, pequeños cambios en el contorno de la imagen y diferencias de fondos y de escalas. Para calcular el Histograma de Gradientes Orientados de las imágenes de entrada, se deben seguir los siguientes pasos del algoritmo, como se ilustran en la Figura **3-1**:

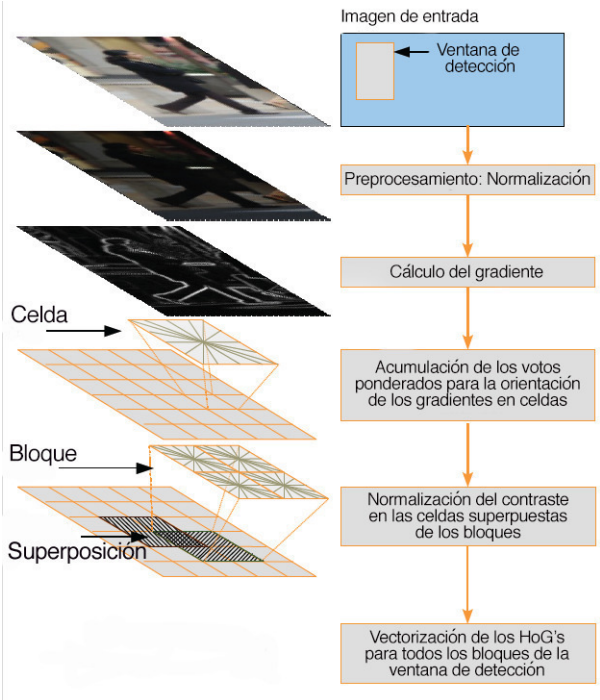


Figura 3-1: Diagrama de Flujo del Algoritmo HOG (Dalal & Triggs, 2005).

A partir de ahora se muestra la metodología seguida para el desarrollo de cada paso del algoritmo.

3.1.1. Normalización de la imagen

Antes de calcular las características para la clasificación es necesaria una fase previa de adaptación de la imagen de entrada. Primero se aplica una normalización global para reducir la influencia de los efectos de iluminación y por lo general se utiliza corrección gamma. La imagen a color de tres canales con codificación RGB capturada por la cámara es transformada en una imagen en escala de grises de un solo canal. Dicha transformación se puede realizar calculando la media ponderada de los tres canales RGB que da como resultado el valor del canal en escala de grises.

Para llevar a cabo la conversión de canales en cada píxel de la imagen de entrada existen diferentes fórmulas propuestas, sin embargo, el método preferido en este trabajo es la transformación HSV (Hue, Saturation, Value o tono, saturación y valor) que se muestra en la ecuación (3-1), ya que el canal de valor es una mejor representación de la luminancia.

$$RGB[A]toGray : Y \leftarrow 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B \tag{3-1}$$

El beneficio de realizar esta conversión recide en que el número de píxeles que se tienen que computar disminuye, puesto que hemos pasado de una imagen con tres canales a una con uno solo. Cabe mencionar que la importancia de este punto en la detección se debe a que, para realizar el cálculo de bordes, no son relevantes los valores concretos de intensidad en los canales Red (Rojo), Green (Verde) y Blue(Azul), sino más bien la variación de intensidad global que se produce.

El algoritmo desarrollado primero debe procesar la imagen en la cual deseamos detectar a unas persona, después convertir la imagen a escala de grises. Para convertir una imagen a escala de grises contamos con la función *cvCvtColor* de OpenCV.

Llamamos a la función *cvCvtColor*. Utilizando un parámetro de entrada que es la imagen inicial (testImage), un parámetro de salida que corresponde a la imagen destino (Image) y la opción del tipo de conversión de color a través de la palabra *CV_RGB2GRAY*, que solicitará a *cvCvtColor* que la variable img sea convertida de formato RGB a formato escala de grises.

```
cvtColor(testImage, Image, CV_BGR2GRAY);
```

3.1.2. Cálculo del gradiente

El objetivo de este paso del algortimo es la extracción de características de una imagen, las cuales identifiquen la silueta de una persona o algún otro objeto. Las características son extraídas teniendo en cuenta los bordes. El proceso de obtener información de los bordes presentes en una imagen, se consigue calculando los gradientes y las orientaciones de los píxeles.

La idea principal es que la apariencia y la forma de los objetos pueden ser caracterizados de mejor manera por la distribución de gradientes de intensidad locales. Esto se hace mediante la división de la imagen en varias regiones pequeñas (celdas), cada una de las cuales tiene un histograma de las direcciones de su gradiente o también llamada orientación de borde de los píxeles de la celda.

El gradiente es un vector, en donde sus componentes miden la rapidez en que los valores de los píxeles cambian en la distancia y en las direcciones X y Y. En imágenes discretas se puede considerar dx y dy en términos del número de píxeles entre dos puntos. En orden de detectar la presencia de una discontinuidad en el gradiente, debemos calcular el cambio en el gradiente en el punto (i,j). Esto se puede hacer referenciando la medida aportada por la magnitud del cambio de dirección de máxima variación del gradiente y su dirección donde el cambio de intensidad es máximo.

Para realizar el cálculo de los gradientes en x se utilizó la Ecuación (3-2) y para y la Ecuación (2-3).

$$dx = f(x+1, y) - f(x-1, y) \tag{3-2}$$

$$dy = f(x, y+1) - f(x, y-1) \tag{3-3}$$

El gradiente de la imagen se calcula a través de máscaras derivativas discretas, como Sobel, Prewitt, Roberts, o el Laplaciano. Al aplicar una de estas máscaras, se obtienen los gradientes direccionales a partir de los cuales se pueden determinar las componentes de magnitud y fase del gradiente de la imagen.

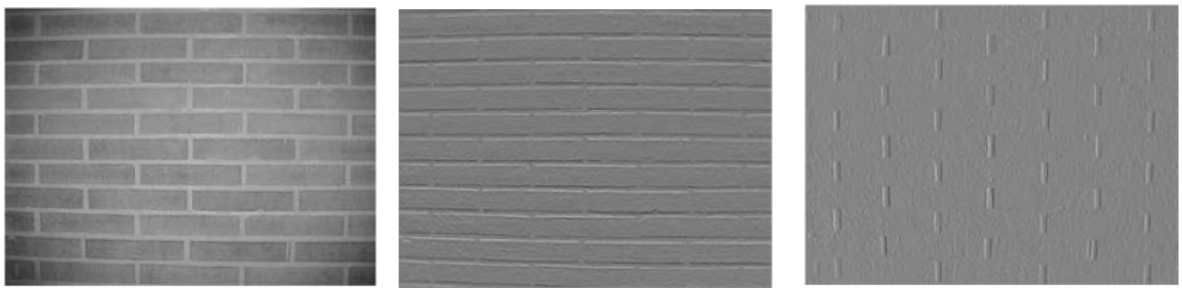
La magnitud del gradiente está dada por la Ecuación (3-4):

$$G(x, y) = 2\sqrt{dx^2 + dy^2} \tag{3-4}$$

Mientras que la orientación del gradiente se calcula a partir de la Ecuación (3-5):

$$\theta(x, y) = \arctan \frac{dy}{dx} \tag{3-5}$$

Un ejemplo de la aplicación del cálculo de gradientes a una imagen se puede apreciar en la Figura **3-2**.



(a) (b) (c)

Figura 3-2: Resultados del cálculo de gradientes. (a) Imagen de entrada. (b) Gradiente horizontal. (c) Gradiente Vertical. Tomada de [3].

En el algoritmo los gradientes se calculan para la imagen completa mediante el algoritmo de OpenCV Sobel.

```
// Calculamos las derivadas en las direcciones x e y.
Mat gradiente_x, gradiente_y;
Mat abs_gradiente_x, abs_gradiente_y;

// Gradiente X
Sobel( src_gray, gradiente_x, ddepth, 1, 0, 3, escala, delta, BORDER_DEFAULT );
// Gradiente Y
Sobel( src_gray, gradiente_y, ddepth, 0, 1, 3, escala, delta, BORDER_DEFAULT );

// Se convierten los resultados parciales en CV_8U
convertScaleAbs( gradiente_x, abs_gradiente_x );
convertScaleAbs( gradiente_y, abs_gradiente_y );

// Por último, tratamos de aproximar el gradiente mediante la adición de ambos gradientes direccionales
addWeighted( abs_gradiente_x, 0.5, abs_gradiente_y, 0.5, 0, gradiente );
```

Los parámetros que utiliza la función son:

Src_gray: Imagen de entrada.
Gradiente_x / gradiente_y: Imagen de salida.
Ddepth: Profundidad de la imagen de salida. (Se establece en CV_16S para evitar el desbordamiento.)
X_order: Orden de la derivada en la dirección x.
Y_order: Orden de la derivada en la dirección y.
Escala, delta y BORDER_DEFAULT: Valores predeterminados.

3.1.3. Construcción del histograma de orientación

Un histograma es una representación gráfica de una variable en forma de un diagrama de barras (comúnmente). El cálculo del histograma, es probablemente el paso más importante del algoritmo, para realizarlo se deberá dividir la imagen en bloques, y cada bloque en celdas. Una vez teniendo los bloques y celdas definidos, se generará un histograma de cada celda sumando las magnitudes de los gradientes que estén dentro de un mismo intervalo definido de orientación del gradiente. Después, se deberán normalizar los histogramas obtenidos con alguno de los métodos existentes. Finalmente, se acomodarán los histogramas de cada celda para obtener así el vector de características.

Una celda es una sub-región espacial local de la Región de Interés (ROI por sus siglas en inglés) formada por un grupo de píxeles adyacentes. Para definir el tamaño de una celda, es necesario escoger dos parámetros, cx que es el número de píxeles de la celda en el eje x, y cy que es el número de píxeles de la celda en el eje y. Por tanto, el número total de píxeles dentro de una celda viene determinado por la Ecuación (3-6):

$$C = (c_x \times c_y)[pixeles] \tag{3-6}$$

Los bloques son un conjunto de celdas de la imagen que deben estar distribuidos a lo largo y ancho de la misma y con cierto solape entre ellos. De esta forma, el avance de bloques se realiza eliminando la columna de las celdas de la izquierda y añadiendo la columna de la derecha para el desplazamiento horizontal, mientras que para el vertical se elimina la fila de las celdas de arriba, añadiendo la fila de celdas de abajo. La división en celdas y bloques de una imagen se pueden observar en la Figura **3-3**:

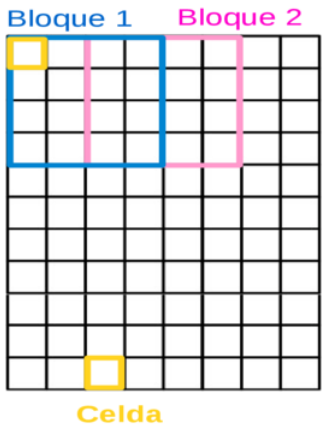


Figura 3-3: División en bloques y celdas.

Una vez definidos los bloques y celdas, se divide el rango de orientaciones en un número de intervalos fijos, se asigna cada píxel de la celda a un intervalo en función de la orientación del gradiente y se suman las magnitudes de los gradientes que estén dentro de un mismo intervalo.

El histograma de orientaciones representa la ponderación de dirección que posee cada celda en la imagen, y a continuación se detalla la manera en que se llevó a cabo este paso del algoritmo:

Dividir el rango de orientaciones en n intervalos fijos, el rango de orientaciones se trabaja usualmente entre [0°, 180°] o [0°, 360°] y el intervalo de orientaciones puede tomar valores de n, para el caso de un rango de 0° a 180° de manera general se utiliza una división en 9 intervalos de 20° cada uno, sin embargo esta división puede cambiar. La Figura **3-4** ejemplifica el uso de intervalos.

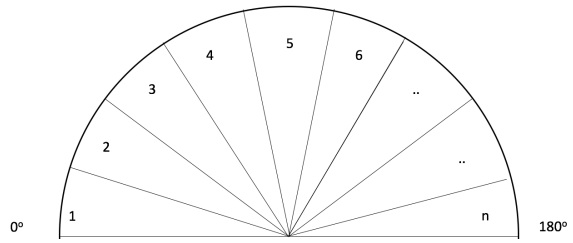


Figura 3-4: Caso del rango de 0° a 180° con 9 intervalos.

Asignar cada píxel de la celda a un intervalo en función de la orientación del gradiente. El factor de asignación se define de tal manera que para todos los gradientes cuya orientación esté dentro de los límites definidos por el intervalo valga 1, mientras que para los que están fuera del intervalo valga 0. Tal como se muestra en la Ecuación (3-7):

$$W_k = \begin{cases} 1, & \text{Si } \in intervalo\, k \\ 0, & \text{Caso contrario} \end{cases} \tag{3-7}$$

Existen dos problemas en la asignación a intervalos:

- Píxeles muy cercanos pueden asignarse a diferentes celdas.
- Sensibilidad a pequeñas variaciones de la forma del objeto.

Solución:

- Asignar cada píxel a las cuatro celdas más cercanas con un peso proporcional a la distancia del píxel al centro de cada celda.

Acumular la magnitud del gradiente de todos los píxeles asignados a un intervalo siguiendo la Ecuación (3-8):

$$h(k) = \sum W_k M_g \tag{3-8}$$

La creación de los histogramas gráficamente se muestra en la Figura **3-5**:

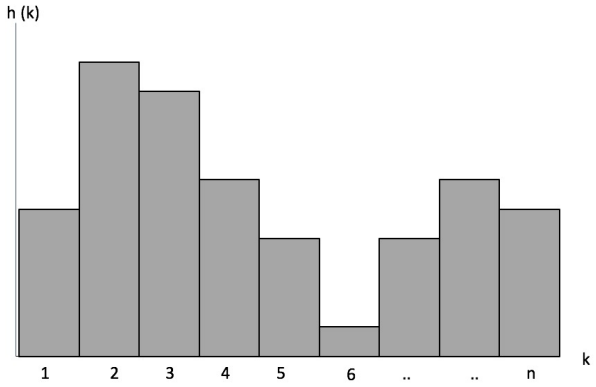


Figura 3-5: Gráfico del histograma de orientación.

Finalmente, formar el histograma de orientación para cada intervalo, utilizando las ponderaciones de intensidad de cada orientación, como se muestra en Figura **3-6**.

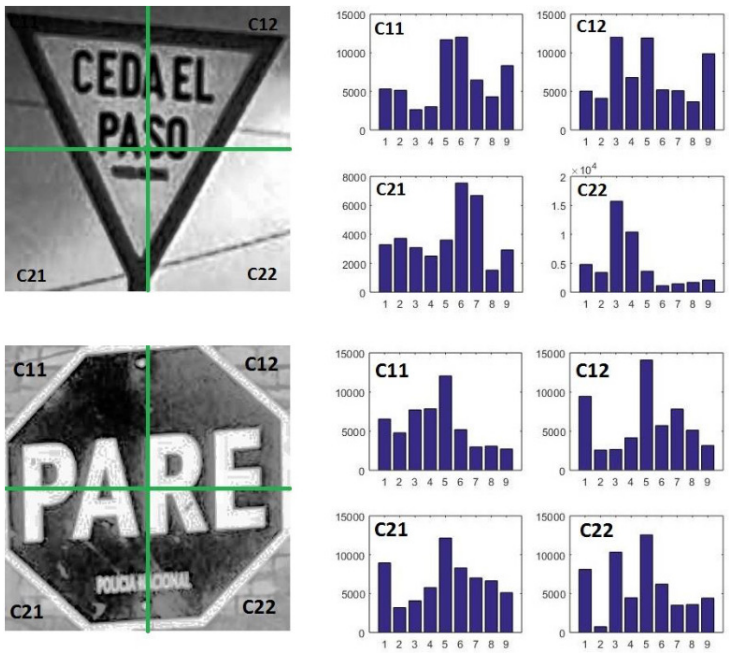


Figura 3-6: Histograma de orientación. Tomada de [4]

3.1.4. Normalización de los histogramas de orientación

Los cambios en la intensidad del gradiente se reflejan en los histogramas de orientación obtenidos de imágenes que poseen diferentes ponderaciones de intensidad debido a las variaciones de contraste, producidos por los cambios de iluminación, para contrarrestar esas variaciones se emplea el método de normalización, el cual consiste en:

- La agrupación de celdas en bloques, para cada bloque se obtiene el histograma de gradiente orientado de cada celda.
- Luego se concatenan todos los histogramas uno a continuación del otro obteniendo en vector.
- Y se realiza la normalización de ese vector utilizando la norma L2, que se muestra en la Ecuación (3-9), obteniendo un vector normalizado.

$$v = (x_1, x_2, x_3, ..., x_n)$$
$$v' = \frac{v}{\sqrt{\|v\|^2 + \epsilon}} \tag{3-9}$$

El proceso de normalización paso a paso se muestra en la Figura **3-7**.

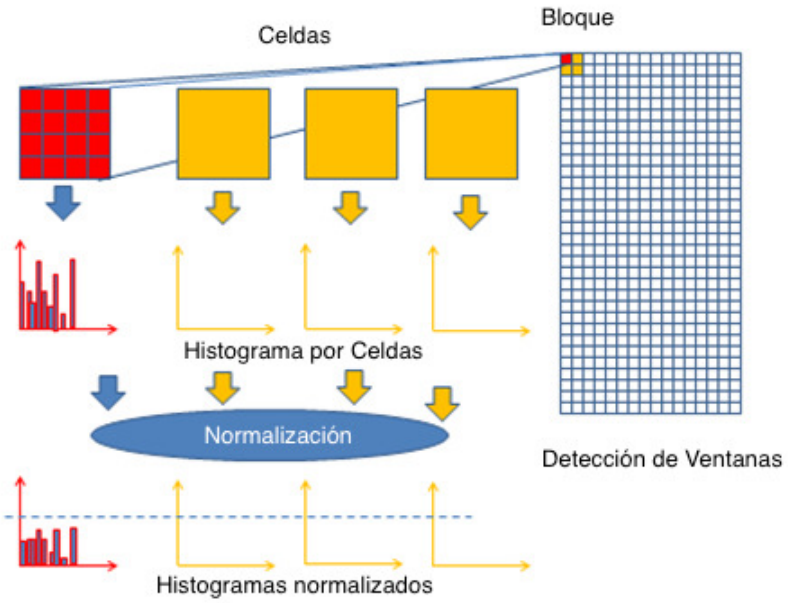


Figura 3-7: Proceso de Normalización de imagen. Tomada de [5]

La aplicación del proceso de normalización a una imagen se presenta en la Figura **3-8**

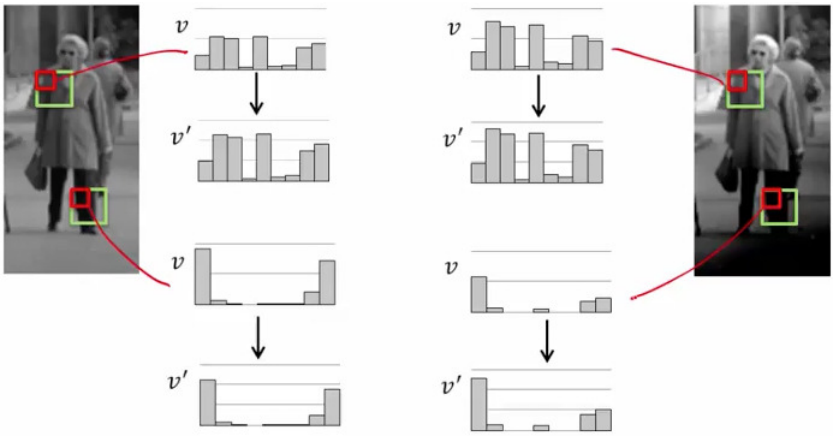


Figura 3-8: Aplicación de la normalización a una imagen. Tomada de [6]

3.1.5. Construcción del vector de características HOG

El vector de características HOG que representa a la imagen se compone de la concatenación de histogramas normalizados, los cuales se obtienen desplazando el bloque por toda la imagen, el desplazamiento del bloque permite el solapamiento de celdas, haciendo que la celda pueda pertenecer a varios bloques y haciendo que ésta tenga una normalización diferente para cada bloque, esto permitirá obtener una eliminación de la varición de contraste de forma local para cada celda.
El proceso de construcción del vector de características se muestra de manera ordenada en las Figuras **3-9** y **3-10**.

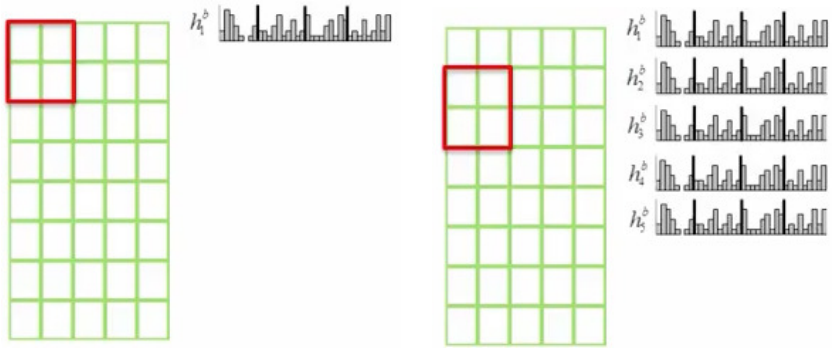
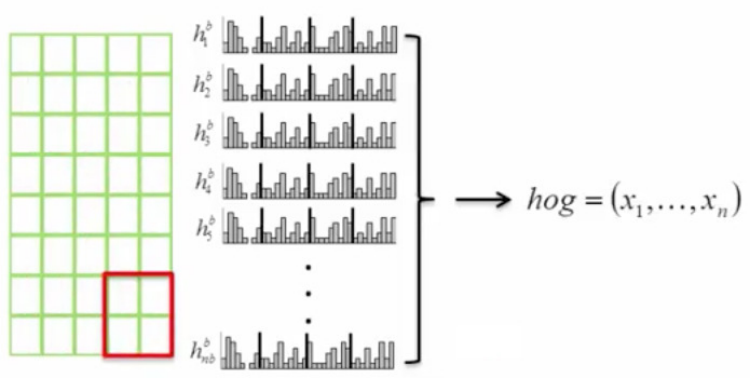


Figura 3-9: Construcción del descriptor final. (a) Desplazamiento del bloque en la imagen. (b) Normalización de los histogramas por bloque.Tomada de [6]



(a)

Figura 3-10: Construcción del descriptor final. (a) Concatenación de todos los histogramas y obtención del descriptor final. Tomada de [6]

3.1.5.1. Parámetros del detector

Para determinar el tamaño que tendrá el vector de características HOG, se utiliza la Ecuación 3-10.

$$No.Hog = No.Bloques \times No.Celdas/bloques \times No.Orientaciones \tag{3-10}$$

En la siguiente tabla se muestran los valores necesarios para realizar la ecuación, su definición y los valores utilizados para el algoritmo desarrollado. Los valores que aquí se definen serán utilizados para el entrenamiento del algoritmo de clasificación utilizando una Máquina de Soporte Vectorial.

Término	Definición	Valor
T_Img	Tamaño de la imagen	64x128
T_Celda	Tamaño de celda	8x8
No.Celdas	Número de celdas por bloque	2x2
No.Orientaciones	Número de intervalos de orientación del gradiente	9
T_Intervalo	Tamaño de los intervalos	20°
Signo	Signo del gradiente	Sin signo
No.Hog	Tamaño del vector	Por calcular

Tabla 3-1: Parámetros del detector.

3.2. Entrenamiento del detector

La etapa de entrenamiento, es importante dentro de la detección de personas, esto debido a que, su correcto funcionamiento determinará la eficiencia del clasificador y por lo tanto del

detector. Es decir, el entrenamiento define el número de aciertos y de errores al momento de comprar las imágenes obtenidas por la cámara, y por ende influye en la etapa de pruebas y resultados que se obtengan más adelante.

MSV para casos lineales

Las MSV ofrecen la ventaja de que las mismas pueden ser utilizadas para resolver tanto problemas lineales como no lineales. En el caso de ser linealmente separable, las MSV conforman hiperplanos que separan los datos de entrada en dos subgrupos que poseen una etiqueta propia. En medio de todos los posibles planos de separación de las dos clases etiquetadas como {-1, +1}, existe sólo un hiperplano de separación óptimo, de forma que la distancia entre el hiperplano óptimo y el valor de entrada más cercano sea máxima (maximización del margen) con la intención de forzar la generalización de la máquina que se esté construyendo. Aquellos puntos o ejemplos sobre los cuales se apoya el margen máximo son los denominados vectores de soporte. Un ejemplo de este caso se puede observar en la Figura **3-11**.

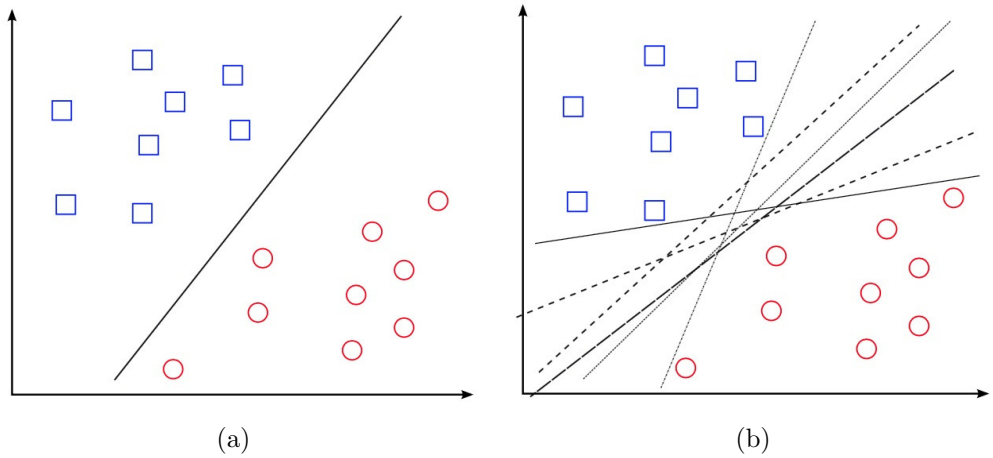


Figura 3-11: Hiperplanos de separación en un espacio bidimensional de un conjunto de ejemplos separables en dos clases. (a) Ejemplo de hiperplano de separación. (b) Otros ejemplos de hiperplanos de separación, de entre los infinitos posibles. Tomada de [7].

SVM^{Light}

SVM^{Light} [17], es una implementación de MSV en C, creada por Thorsten Joachims, (versión 6.02). Es una implementación de [Vapnik, 1995] para el problema de reconocimiento de patrones, para el problema de regresión y para el problema del aprendizaje de una función de clasificación. Esta herramienta fue creada con variantes para aprendizaje supervisado y

para semisupervisado transductivo. Además es una implementación de una MSV adecuada para trabajar con grandes conjuntos de datos.
Se utilizó como herramienta para la detección de personas debido a que el algoritmo tiene requisitos de memoria escalable, resuelve problemas de clasificación y regresión lo que lo vuelve óptimo para el objetivo de esta tesis, además es un software libre y su código puede ser utilizado para investigación.

Implementación

Para poder implementar el algoritmo detector de personas fue necesario realizar el entrenamiento de la máquina de Soporte Vectorial, para ello se utilizaron las librerías de *OpenCV*, el software SVM^{Light}, una base de datos de imágenes positivas y negativas el procedimiento se llevó a cabo en tres etapas y el cumplimiento de estas permite el posterior análisis del rendimiento tanto del entrenamiento como del clasificador.

Etapas del entrenamiento del detector:

1. Creación del dataset.
2. Entrenamiento.
3. Clasificación.

Para el desarrollo de los pasos 2 y 3 se utilizaron algunas funciones desarrolladas en [18] como base para la implementación del algoritmo final de entrenamiento del clasificador.

3.2.1. Creación del dataset

Para poder entrenar este clasificador se requiere de dos conjuntos de ejemplos: positivos y negativos, es por ello que la primera etapa es la construcción del DataSet que utilizará el algoritmo para generar los datos del entrenamiento de las imágenes positivas y negativas. En este caso las imágenes positivas hacen referencia a imágenes que contienen a una persona y las negativas a aquellas que no contienen a una persona. Es importante señalar que se decidió utilizar ventanas de detección de 64x128 píxeles para las imágenes tanto positivas como negativas, ya que en este tamaño cabe completamente una persona y ello nos permite la estandarización de los ejemplos.

Para la creación del conjunto de ejemplos positivos se utilizó la base de datos MIT Pedestrian [19], que representa una base de datos estándar en la detección de personas y contiene un total de 924 imágenes positivas de tamaño 64x128, por lo que en este caso no fue necesario adaptarlas al tamaño establecido para el entrenamiento. Dichas personas se encuentran en determinadas posiciones, con diferentes iluminaciones y diferentes fondos como se puede

observar en la Figura **3-12**.

(a) (b) (c) (d)

Figura 3-12: Base de datos MIT Pedestrian.

La base de datos INRIA [20] se compone de un total de 3302 imágenes positivas de personas de tamaño 64x128 que pudieron ser usadas sin cambiar sus dimensiones. El conjunto de imágenes del que dispone cuenta diferentes vistas de las personas, aumentando así el número de posturas reconocibles, ejemplo de ello se muestra en la Figura **3-13**.

(a) (b) (c) (d)

Figura 3-13: Base de datos INRIA.

Además se creó una base de datos propia, con ejemplos positivos tomados de los videos grabados por la cámara web en los lugares de mayor afluencia de personas, en diferentes horas del día, para poder tener diferentes rangos de iluminación y fondos variantes. El lugar en el que se realizaron las tomas fue la Benemérita Universidad Autónoma de Puebla. Se obtuvo un total de 274 imágenes ya ajustadas al tamaño necesario. Ejemplos de estas imágenes se pueden apreciar a continuación en la Figura **3-14**:

(a) (b) (c) (d)

Figura 3-14: Base de datos creada para el entrenamiento.

Sin embargo este número de imágenes no es suficiente para poder obtener buenos resultados en el entrenamiento, por lo que también se utilizó la base de datos de personas PETA [21] que contiene 19000 imágenes de las que solo se utilizaron 3835 y se adaptaron al tamaño estándar utilizado.

Por otro lado, las imágenes negativas fueron seleccionadas de manera aleatoria y variada, a fin de conseguir un mejor entrenamiento, integrando imágenes del entorno en el cuál se desarrollarán las pruebas finales, como árboles, arbustos, botes de basura, postes, etc. La base de datos MIT Pedestrian no contiene imágenes negativas, en cambio se usaron las imágenes negativas de INRIA que son 453 y se modificaron en tamaño. Debido a que el número de imágenes positivas es muy grande en comparación con el de las negativas fue necesario incorporar otra base de datos, el dataset seleccionado fue The Oxford Buildings Dataset [22] que pone a disposición un conjuntos de 5062 imágenes entre edificios y calles, objetos que son convenientes utilizar a fin de lograr el objetivo de la detección de personas. Parte de la base de datos se muestra en la Figura **3-15**.

Para incrementar la base de datos de ejemplos negativos se realizó el mismo proceso de toma de imágenes por la cámara web en la Universidad, bajo las mismas condiciones pero obteniendo muestras negativas de las que se obtuvieron 1510 imágenes. Algunos ejemplos del conjunto negativos se observan en la Figura **3-16**.

(a) (b) (c) (d)

Figura 3-15: Base de datos The Oxford Buildings Dataset.

(a) (b) (c) (d)

Figura 3-16: Base de datos creada con ejemplos negativos.

Finalmente el total de imágenes negativas es de 7025 y de positivas 8335. Una vez concluído este paso se procede a realizar el entrenamiento.

3.2.2. Entrenamiento

La función de entrenamiento en el algortimo desarrollado se encarga de encontrar y almacenar una serie de características propias de cada imagen (positivas y negativas), para que éstas puedan ser comparadas. Dichas características se reconocen como patrones del conjunto de datos de entrenamiento y la finalidad de su obtención es poder crear un modelo que posteriormente sea empleado en la clasificación de nuevos datos de entrada que en este caso se define como nuevas imágenes de entrada.

Técnicamente, la finalidad del entrenamiento recide en determinar los parámetros del hiperplano de separación entre las clases. El entrenamiento de la MSV se realiza utilizando un kernel lineal y el resultado de esta etapa es el modelo que podrá ser utilizado después en la clasificación. Lo que se busca es contar con un sistema capaz de reconocer personas e ignorar los casos en que los objetos captados no lo sean. Una buena separación entre las clases permitirá una correcta clasificación. El conjunto de entrenamiento es sometido al proceso de extracción de características, una vez calculados o generados los vectores de características, son almacenados para posteriormente presentarlos al clasificador, de esta forma no tiene que calcularse el mismo vector de características de cada imagen en cada iteración del clasificador.

A continuación se describe el proceso llevado a cabo para esta etapa.

Primeramente se crearon dos carpetas contenedoras llamadas positivas y negativas que contienen los conjuntos de imágenes obtenidas en la sección 3.2.1 respectivamente. Se crearon además, los archivos *caracteristicas.dat* y *vectorDescriptor.dat* donde se guardarán las características extraídas y el vector final del descriptor, correspondientemente.

El programa se encarga de leer las imágenes de los conjuntos de entrenamiento de los directorios especificados, de igual forma se valida que el tamaño estándar de imagen se respete y que además los formatos de las imágenes sean aceptados, se establecieron los formatos .png, .jpg, .ppm y .bmp como válidos debido a la integración de al menos cuatro diferentes DataSets. Ver los Algoritmos 1 y 2.

```
1  static string directorioPos = "positivos/";
2  static string directorioNeg = "negativos/";
3  static string caracteristicas = "caracteristicas.dat";
4  static string modeloSVM = "modelosvmlight.dat";
5  static string vectorDescriptor = "vectordescriptor.dat";
6  static string cvHOGFile = "clasificadorHOG.yaml";
7  ...
8  extensionValida.push_back("jpg");
9  extensionValida.push_back("png");
10 extensionValida.push_back("ppm");
11 extensionValida.push_back("bmp");
```

Algoritmo 1: Creación de archivos y asignación de carpetas.

```
1  obtenerArchivosDelDirectorio(directorioPositivos, imagenesPosEntrenamiento, extensionValida);
2  obtenerArchivosDelDirectorio(directorioNegativos, imagenesNegEntrenamiento, extensionValida);
3  static void obtenerArchivosDelDirectorio(const string& nombreDirectorio, vector<string>
4  & nombreDelArchivo, & const vector<string>& extensionValida){
6     printf("Abriendo el directorio %s \n ", nombreDirectorio.c_str());
7     #ifdef __MINGW32__
8        struct stat s;
9     #endif
10    struct dirent* ep;
11    size_t extensionLocation;
12    DIR* dp = opendir(nombreDirectorio.c_str());
13    if (dp != NULL) {
14       while ((ep = readdir(dp))) {
15       #ifdef __MINGW32__
16          stat(ep→d_name, &s);
17          if (s.st_mode & S_IFDIR) {
18             continue;
19          }
20       #else
21          if (ep→d_type & DT_DIR) {
22             continue;
23          }
24    #endif
25       ubicarExtension = string(ep→d_name).find_last_of(".") ;
26       string tempExt = toLowerCase(string(ep→d_name).substr(ubicarExtension + 1));
27       if (find(extensionValida.begin(), extensionValida.end(), tempExt) != extensionValida.end()) {
28          nombreDelArchivo.push_back((string) nombreDirectorio + ep→d_name);
29       }
30       else {
31          printf("Extensión inválida, %s \n", ep→d_name);
32       }
33    }
34    (void) closedir(dp);
35 }
45 else {
41    printf("Error al abrir directorio %s' ! \n", nombreDirectorio.c_str());
42 }
43 return;
44 }
```

Algoritmo 2: Función encargada de validar archivos.

Seguido de la validación de los archivos, se lee cada una de las imágenes de cada carpeta, se muestra en pantalla el número de imagen procesada, el porcentaje de avance en la lectura y el nombre del archivo. Ver Algoritmo 3.

```
1   ...
2   float porcentaje;
3   fstream archivo;
4   archivo.open(caracteristicas.c_str(), ios::out);
5   if (archivo.good() && archivo.is_open()) {
6      #if TRAINHOG_USEDSVM == SVMLIGHT
7         // ***** Validar el uso del SVMLIGHT ******* //
8      #endif
9      for (unsigned long archivoActual = 0; archivoActual < overallSamples; ++archivoActual) {
10        guardarCursor();
11        const string archivoImagenActual = ( archivoActual <imagenesPosEntrenamiento.size() ?
12        imagenesPosEntrenamiento.at(archivoActual): imagenesNegEntrenamiento.at(
13        archivoActual-imagenesPosEntrenamiento.size()));
14     if ((archivoActual+1) % 10 == 0 || (archivoActual+1) == overallSamples ) {
15        porcentaje = ((archivoActual+1) * 100 / overallSamples); Ã,
16        porcentaje = printf("%5lu (%3.0f%%): Archivo '%s'", (archivoActual+1),
17        porcentaje, archivoImagenActual.c_str());
18        fflush(stdout);
19        reiniciarCursor();
20     }
21     ...
22     }
23     ...
```

Algoritmo 3: Proceso de lectura de archivos.

Además, por cada imagen leída se generan las características HOG de la misma y se agregan al vector. La distinción entre las características de una imagen positiva o negativa se hace con el uso de 1 y -1, una característica necesaria para poder hacer uso de la MSV más adelante. Una vez que se extraen las características, la imagen que estaba siendo utilizada se libera, a fin de agilizar el tiempo de ejecución al no consumir tanta memoria. Al finalizar el procesamiento del 100 % de las imágenes se guardan todos los datos en el archivo características.dat. Ver Algoritmo 4.

A continuación se leen y envían todas las características y sus clases respectivas al algoritmo de aprendizaje automático, es decir, a la SVM^{Light} para ser entrenada. En este paso el algoritmo hace uso de la herramienta SVM^{Light} previamente importada por lo que en siguiente código sólo se aprecian los llamados a las funciones respectivas. Se generar el vector de características único a partir de los vectores de soporte calculados por la MSV y el modelo MSV producido. Ver Algoritmo 5.

```
...
vector<float> vectorCaracteristicas;
calcularCaracteristicasActual(archivoImagenActual, vectorCaracteristicas, hog);
...
if (!vectorCaracteristicas .empty()) {
    archivo << (( archivoActual < imagenesPosEntrenamiento.size()) ? "+1": "-1");
    for (unsigned int caracteristica = 0; feature < vectorCaracteristicas.size(); ++caracteristica){
        archivo << " "<< ( caracteristica + 1) << ":"<< vectorCaracteristicas.at(caracteristica);
    }
    archivo << endl;
    }
    archivo.flush();
    archivo.close();
    ...
else{
    printf("Error al abrir archivo '%s'!  n", caracteristicas.c_str());
    return EXIT_FAILURE;
}
...
```

Algoritmo 4: Función que calcula las características HOG de las imágenes.

```
...
printf("Llamando a %s \n", TRAINHOG_SVM_TO_TRAIN::getInstance()→getSVMName());
TRAINHOG_SVM_TO_TRAIN::getInstance()→read_problem(const_cast<char*> (caracteristicas.c_str()));
TRAINHOG_SVM_TO_TRAIN::getInstance()→train();
printf("Entrenamiento listo, guardando en archivo \n");
TRAINHOG_SVM_TO_TRAIN::getInstance()→saveModelToArchivo(modeloSvm);
printf("Generando la representación final de características HOG utilizando SVMLight \n");
vector<float> vectorDesciptor;
vector<unsigned int> vectorDesciptorIndices;
TRAINHOG_SVM_TO_TRAIN::getInstance()→getSingleDetectingVector(vectorDesciptor,
vectorDesciptorIndices);
guardarEnArchivoVectorDescriptor(vectorDesciptor, vectorDesciptorIndices, vectorDesciptor);
...
```

Algoritmo 5: Entrenamiento de la SVM^{Light}

3.2.3. Clasificación

Ahora se procede a realizar la clasificación, es decir, obtener imágenes desde una cámara en tiempo real y análizar la detección de personas para conocer el grado de fiabilidad del entrenamiento generado. Una función importante que brinda *OpenCV* y que se ha incorporado al algoritmo es la función $hog.detectMultiScale$, la cual realiza la detección de objetos con una ventana de varias escalas. Esto hace referencia a la generación de candidatos en pirámide. Los objetos en una secuencia de video en tiempo real pueden aparecer a diferentes distancias, es decir, a distintas escalas, razón por la cual es conveniente hacer barridos de la imagen con diferentes tamaños de ventana y el límite de la escala es donde quepa la imagen

con el tamaño estándar dado. En este caso el mínimo tamaño de escala es 64x128.

Antes de aplicar la función *hog.detectMultiScale*, se debe alimentar el HOG utilizando el modelo generado por la MSV, empleando la función *hog.setSVMDetector* y basta con realizarlo una única vez. Se integra en esta sección el algoritmo detector de personas desarrollado en 3.1, y a partir de este punto el algoritmo se centra en la detección de personas usando el descriptor HOG recién entrenado. Ver Algoritmo 6.

Los parámetros necesarios para la implementación de la función *hog.detectMultiScale* se describen a continuación:

- **testImage:** La imagen en la que queremos que detecte.
- **found:** Límites de lo objetos detectados.
- **hitThreshold:** Umbral para la distancia entre las características y el plano de clasificación de la MSV.
- **winStride:** El paso de la ventana. Debe ser un múltiplo de blockStride.
- **padding:** Parámetro Mock para mantener la compatibilidad de interfaz de la CPU(0,0).
- **scale:** Coeficiente de la ventana de detección de aumento.
- **group_threshold:** Coeficiente para regular el umbral de similitud. Cuando se detecta, algunos objetos pueden ser cubiertos por muchos rectángulos. 0 significa no realizar el agrupamiento.

Después de implementar la función se pintan en la ventana de *OpenCV* rectángulos donde se ha detectado una persona y es posible realizarlo empleando la función *showDetections*. Como se está trabajando con una cámara web y en tiempo real no le pasamos ningún parámetro al programa. Lo primero que se hace es abrir la cámara correctamente. Obtener frames cada cierto tiempo y pasarlos a escala de grises para llevar a cabo la clasificación. Por último, se muestra la imagen con las personas detectadas. Ver Algoritmo 7.

```
1   const double hitThreshold = TRAINHOG_SVM_TO_TRAIN::getInstance()→getThreshold();
2   hog.setSVMDetector(vectorDesciptor);
3   hog.save(cvHog);
4   ...
5   detectTrainingSetTest(hog, hitThreshold, imagenesPosEntrenamiento, imagenesNegEntrenamiento);
6   ...
7   printf("Detección en tiempo real \n");
8   VideoCapture cap(1);
9   if(!cap.isOpened()) {
10      printf("Error al abrir la cámara \n");
11      return EXIT_FAILURE;
12  }
13  Mat imagen;
14  while (waitKey(30)>=0) {
15      cap >> imagen;
16      cvtColor(imagen, imagen2, CV_BGR2GRAY);
17      detectTest(hog, hitThreshold, imagen);
18      imshow("Detección de Personas", imagen2);
19  }
```

Algoritmo 6: Entrenamiento del Detector HOG.

```
1   ...
2   while (true){
3       cap >> imagen;
4       if (imagen .empty()){
5           continue;
6       }
7       vector<Rect> found, found_filtered;
8       hog.detectMultiScale(imagen, found, 0, Size(8,8), Size(32,32), 1.05, 2);
9       size_t i, j;
10      for (i=0; i<found.size(); i++) {
11          Rect r = found[i];
12              for (j=0; j<found.size(); j++)
13                  if (j!=i && (r & found[j]) == r)
14                      break;
15                  if (j== found.size())
16                      found_filtered.push_back(r);
17      }
18      for (i=0; i<found_filtered.size(); i++) {
19          Rect r = found_filtered[i];
20          r.x += cvRound(r.width*0.1);
21          r.width = cvRound(r.width*0.8);
22          r.y += cvRound(r.height*0.07);
23          r.height = cvRound(r.height*0.8);
24          rectangle(imagen, r.tl(), r.br(), Scalar(225,0,110), 3);
25      }
26  }
27  ...
```

Algoritmo 7: Algoritmo detector de personas.

3.3. Filtro de Partículas

El Filtro de Partículas representa la densidad a posteriori mediante una distribución de partículas (muestras) en el espacio de estados. Las partículas son estados posibles del proceso, que se pueden representar como puntos en el espacio de estados de dicho proceso.

Este algoritmo se enfoca en la propagación de las partículas (muestras de la función de densidad a posteriori), utilizando el modelo de movimiento $P(x_t|x_{t-1}, a_t)$ y el modelo de verosimilitud P $(z_t|x_t)$ de forma que el peso combinado de las partículas de una región aproxima la integral de la función de densidad a posteriori en esa región. En concreto, el Filtro de Partículas representa la densidad a posteriori mediante un conjunto discreto de N partículas $m_1, \ldots, m_N$ y sus probabilidades asociadas $\pi_1, \ldots, \pi_N$.

Inicialmente, el conjunto de partículas se escoge a partir de la distribución a priori $p(x_0)$.

Si no existe información a priori, entonces las partículas se distribuyen uniformemente por el espacio de estados. Posteriormente, en cada instante de tiempo t, se actualizan las N partículas en función de la acción anterior a_{t-1} y la observación actual z_t . Para ello, se aplica el modelo de movimiento $P(x_t|x_{t-1}, a_{t-1})$ a cada una de las N partículas, generando un nuevo conjunto de partículas. Las partículas nuevas representan la predicción de la variable de estado, sin considerar la observación, se obtiene el peso π^i asociado a cada partícula. El conjunto de pesos de cada partícula es proporcional a la probabilidad de su estado y a la suma normalizada de sus pesos. La densidad de los pesos es igual al producto de la densidad previa (del muestreo) y la probabilidad (de los pesos). En la Figura **3-17** se representa un muestreo discreto de una función de densidad de probabilidad continua mediante partículas, cuyo tamaño hace referencia al peso asignado a las mismas.

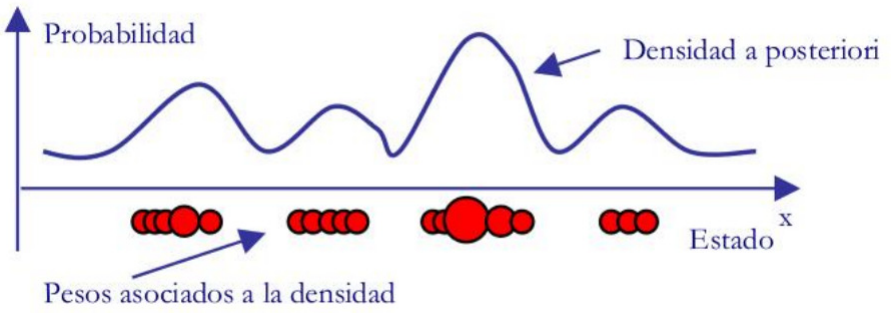


Figura 3-17: Muestreo y pesos obtenidos de las partículas al aplicar el Filtro.

En un último paso, se remuestrea el conjunto de partículas, extrayendo (con reemplazo) N partículas del conjunto actual, proporcional al peso de cada una. En este nuevo conjunto tendrán más probabilidad de desaparecer aquellas partículas para las que no hay evidencia de verosimilitud o, lo que es lo mismo, que tenga menor peso. Una vez construido el nuevo conjunto de partículas, según la probabilidad de éstas se asocia un peso a cada una. Este nuevo conjunto de partículas constituye una representación muestral de la probabilidad a posteriori.

En la Figura **3-18**, se observa de manera gráfica la evolución de las partículas en cada fase, suponiendo que las partículas están estimando un único parámetro, distribuido en el eje horizontal. El área de los círculos representa el peso de cada partícula.

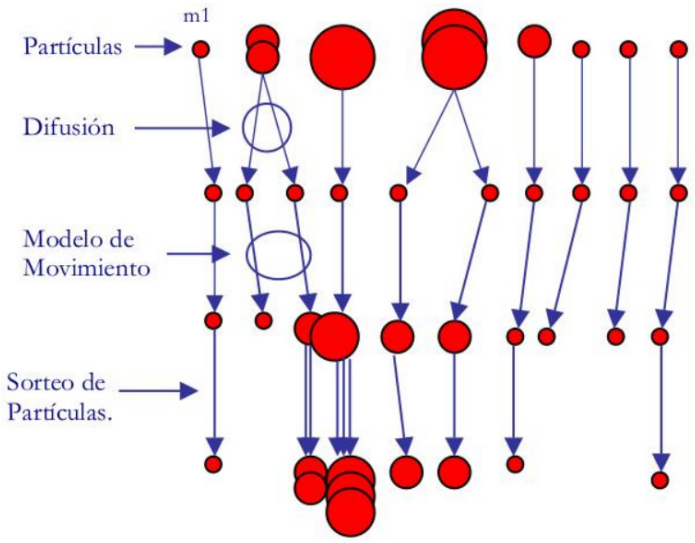


Figura 3-18: Funcionamiento del Filtro de Partículas.

Los elementos que se consideran en el método del Filtro de Partículas son:

- Los Vectores de medidas Z, en este caso, serán provenientes de las imágenes. Las medidas dependen del estado del objeto, y el estado del objeto se deriva estadísticamente de las medidas.
- El Modelo de Movimiento F, se utiliza para predecir la posición del objeto en el instante actual, a partir de la densidad de probabilidad del instante anterior ($x_{t+1} = F(x_t)$).
- El Modelo de Verosimilitud $P(Z_t|x_t)$, la estimación de la función de probabilidad condicional a posteriori $P(Z_t|x_t)$ define la verosimilitud de la medida observada dado un punto en el espacio de estados (es decir, dado un estado del sistema).

Para poder implementar el algoritmo de actualización es necesario un conjunto bien ponderado de partículas en el instante t con pesos iguales $\frac{1}{N}$, y así actualizar este conjunto para reflejar las nuevas medidas obtenidas en el instante $t + \delta t$. El algoritmo de actualización queda como sigue:

1. Propagar cada partícula m_i en el tiempo utilizando el modelo de movimiento F del objeto para obtener un conjunto actualizado de partículas (m_i^*).
2. Obtener un nuevo vector de medidas Z y evaluar la densidad de probabilidad posterior π_i^* para cada m_i^*, $\pi_i^* = p(m_i^*|Z)$ que cuantifica la verosimilitud de (un estado) m_i^* dado un vector de medidas Z. Esto puede ser escrito usando la regla de Bayes descrita en la Ecuación 3-11.

$$Pr(m_i^*|Z) = \frac{p(Z|m_i^*)p(m_i^*)}{p(Z)} \tag{3-11}$$

Donde:
$p(Z)$ es la probabilidad a priori de la medida que se asume constante y conocida.
$p(m_i^*) = \frac{1}{N}$, por lo que: $p(m_i^*|Z) = Kp(Z|m_i^*)$ donde $p(Z|m_i^*)$ puede ser calculado sin inversión de las ecuaciones de medida.

3. Volver a muestrear a partir del conjunto m_i^* con probabilidades π_i^* y generar un nuevo conjunto bien ponderado ($m_i^{'}$) con pesos iguales ($\frac{1}{N}$) para cada partícula. El muestreo de cada partícula consiste en un estado, un peso y otra información posible (una covarianza por instante).

4. Repetir los pasos 1 y 3 para instantes sucesivos.

Una vez que se realiza la detección de personas, se aplica el filtro de partículas para hacer el seguimiento de la persona. En esta sección se describe el desarrollo del algoritmo encargado de determinar la posición que tendrá una persona en el siguiente frame. En este tipo de algoritmos es fundamental procesar los datos de las imágenes en tiempo real, y por lo general esto resulta en un proceso lento debido a la gran cantidad de datos que contiene un video.

Basado en la posición x_t que tiene en el instante actual la persona, se hace una predicción de la siguiente posición $x_t + 1$.

A continuación se realiza el proceso de actualización de los pesos de las partículas utilizando como componente la función gaussiana que se muestra en la Ecuación 3-12, para obtener los datos de la posición de la persona en cada una de las imágenes.

$$\frac{1}{\sigma\sqrt{2\pi}} exp\left[-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right] \tag{3-12}$$

Si x_R es el ruido de la medición y es igual a la varianza σ^2, entonces $\sigma^2 = x_R$ y la covarianza $\sigma = \sqrt{x_R}$.

t es la observación actual z en el instante t, por lo cual $t = z$.

Una vez que la posición de las partículas son actualizadas, se actualizan las observaciones z_i^{update} para cada partícula i, por lo cual $\mu = z_i^{update}, i = 1, \ldots, 30$

De esta forma, la Ecuación 3-12 queda como la Ecuación 3-13 y al simplicarla obtenemos la ecuación final 3-14

$$\frac{1}{\sqrt{x_R}\sqrt{2\pi}} exp\left[-\frac{1}{2}\left(\frac{z - z_i^{update}}{\sqrt{x_R}}\right)^2\right] \tag{3-13}$$

$$\frac{1}{\sqrt{2\pi x_R}} exp\left[-\frac{(z - z_i^{update})^2}{2x_R}\right] \tag{3-14}$$

La Ecuación 3-14 es la que se utiliza en cada iteración durante todo el proceso de seguimiento.

A continuación se describen los métodos y funciones más importantes utilizados para la implementación del algoritmo filtro de partículas, estas funciones están implementadas en OpenCV y permiten definir varias funciones de probabilidad. Una vez calculadas las probabilidades, se puede hacer la evaluación del seguidor.

El proceso de buscar medidas para iniciar sus seguimientos se presenta en el Algoritmo 8.

```
typedef struct {
        int MP;                 // Dimensión del vector de medición
        int DP;                 // Dimensión del vector de estado
        float* DynamMatr;       // Matriz del sistema dinámico lineal
        float* State;           // Vector de estado;
        float** flSamples;      // Matriz de los vectores muestra
        float** flNewSamples;   // Matriz temporal de los vectores muestra
        float* flConfidence;    // Confianza para cada muestra
        float* flCumulative;    // Confianza acumulada
        float* RandomSample;    // Vector aleatorio para actualizar el conjunto muestra
        CvRandState* RandS;     // Array de estructuras para generar vectores aleatorios
}CvConDensation;
```

Algoritmo 8: Estructura del filtro de partículas de OpenCV. Creación de archivos y asignación de carpetas.

La función *CreateConDensation* crea la estructura *cvConDensation* y devuelve el puntero a la estructura.

CvConDensation* cvCreateConDensation(int DynamParams, int MeasureParams, int SamplesNum)

DynamParams: Dimensión del vector de estado.

MeasureParams: Dimensión del vector de medición.

SamplesNum: Número de muestras.

La función *ConDensInitSampleSet*, inicializa el conjunto de muestras en la estructura *CvCondensation* para el algoritmo de condensación, con los valores dentro de los rangos especificados.

void cvConDensInitSampleSet(CvConDensation* ConDens, CvMat* lowerBound CvMat* upperBound)

ConDens: Puntero a la estructura a ser inicializar.

LowerBound: Vector del límite inferior para cada dimensión

UpperBound: Vector del límite superior para cada dimensión.

La función *ConDensUpdateByTime*, estima el estado del modelo estocástico posterior desde su estado actual. Es decir, estima el estado del modelo subsiguiente.

Void cvConDensUpdateByTime (CvConDensation* ConDens);

ConDens: Puntero a la estructura a actualizar.

Capítulo 4

Pruebas de los algoritmos

En esta sección se muestra el análisis de resultados obtenidos de las pruebas desarrolladas, se interpretan y analizan los valores obtenidos de la ejecución de los algoritmos propuestos, a fin de mostras la eficiencia y rendimiento del sistema conjunto.
Antes de comenzar con la fase de pruebas es importante mencionar las características del computador en el que se llevaron acabo, las versiones del software utilizadas en este trabajo de tesis, las características de las imágenes de muestra y de entrada del algoritmo final, el MAV utilizado así como sus características y especificaciones y finalmente mencionar las consideraciones que se deben tomar en cuenta.

Características del computador:

- Sistema Operativo: Ubuntu 14.04 LTS 64 bits.
- Procesador: AMD A6-6310 APU with AMD Radeon R4 Graphics X4
- Memoria RAM: 8GB
- Gráficos: Gallium 0.4 on AMD MULLINS

Versiones del software y herramientas:

- OpenCV: Versión 2.3.1
- SVMlight: Versión 6.02
- G++: Versión 4.6.3

Características de las imágenes de muestra para entrenar el detector y para la detección final:

- Tamaño de las imágenes es de 64x128 píxeles
- Tamaño de celda de 8x8 píxeles
- Tamaño de bloque de 2x2 celdas

- Rango de valores [0°, 180°], sin signo.
- HOG dividido en 9 intervalos
- Tamaño del intervalo de 20°

Teniendo en cuenta estos valores y las relaciones entre celdas bloques y tamaño de ventana, se puede calcular el tamaño del vector de características generado, esto se observa en la Ecuación 4-1, donde el resultado es 3780 valores. Este vector, representa el dato de entrada al clasificador.

$$
\begin{aligned}
x = 64px \quad y = 128px \quad celdax = 8px \quad celday = 8px \\
bloquex = 2\,Celdas \quad bloquey = 2\,Celdas \quad intervalo = 9 \\
CeldasX = \frac{x}{celdax} = \frac{64px}{8px} = 8\,Celdas \\
CeldasY = \frac{y}{celday} = \frac{128px}{8px} = 16\,Celdas \\
BloquesX = 1 + CeldasX - bloquex = 1 + 8 - 2 = 7\,Bloques \\
BloquesY = 1 + CeldasY - bloquey = 1 + 16 - 2 = 16\,Bloques \\
TotalCeldas = CeldasX \times CeldasY = 8x16 = 128\,Celdas \\
TotalBloques = BloquesX \times BloquesY = 7 \times 15 = 105\,Bloques \\
NoHOG = CeldaX \times CeldaY \times TotalBloques = \\
2 \times 2 \times 105 = 420\,HOGs \\
NoValores = intervalo \times NoHOG = 9 \times 420 = 3780\,valores
\end{aligned}
\tag{4-1}
$$

4.1. AR.Drone 2.0 Elite Edition

El AR.Drone 2.0 Elite Edition (Figura **4-1**) es un cuadricóptero desarrollado por Parrot, el vuelo de este drone está basado en 4 potentes motores conectados por un único engranaje a sus respectivas hélices, capaces de actuar de manera totalmente independiente. Esto,

combinado con su giroscopio, acelerómetro, magnetómetro, sensor de presión, sensor de ultrasonidos para medir distancias y una cámara QVGA de 60 fps para medir velocidad de tierra, lo convierte en una nave capaz de hacer diferentes maniobras, y hasta autosostenerse en el aire sin asistencia alguna. A su vez cuenta con dos carcasas fabricadas en telgopor: una para interiores, con aros que protegen las hélices, y otra para exteriores que sólo cubre el cuerpo central del drone, eliminando así peso excedente y fricción con el aire.

(a) (b)

Figura 4-1: Parrot AR.Drone 2.0 Elite Edition.

La arquitectura del software del AR.Drone se encuentra estructurado en capas. En la Figura **4-2** se muestran las capas de la arquitectura de una aplicación montada sobre el SDK del AR.Drone 2.0. Las partes de color amarillo del diagrama son aquellas partes sobre las que el usuario tiene control directo (por ejemplo, el programador tendrá la capacidad de leer en flujos de datos, así como enviar flujos de datos). Se debe tener en cuenta que el motor de control AR.Drone sólo existe específicamente para dispositivos iOS. Esto permite que la interfaz de programación de aplicaciones (API) del AR.Drone sea portátil para dichos dispositivos.

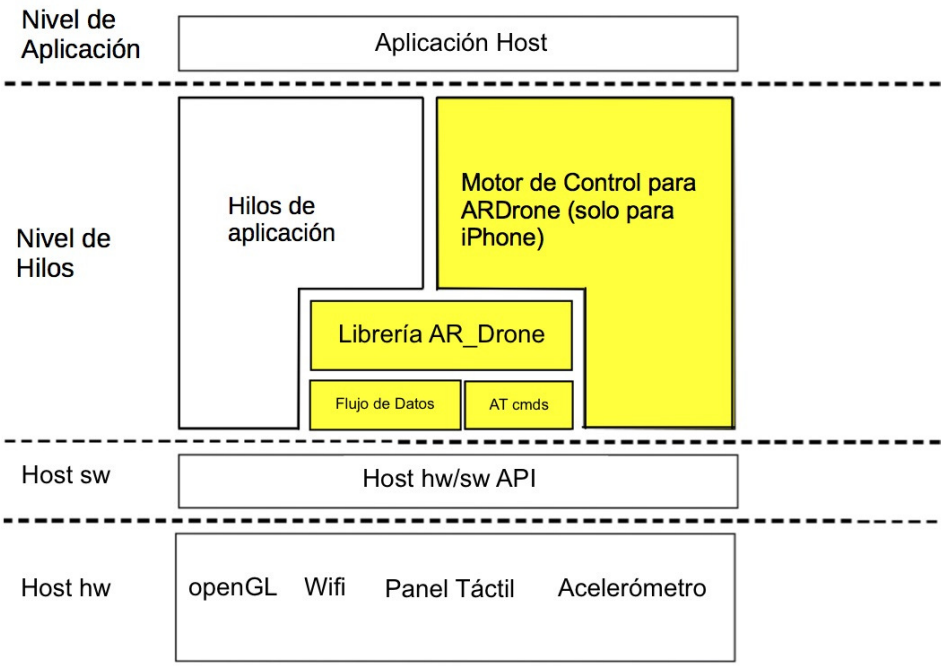


Figura 4-2: Arquitectura en capas del AR.Drone 2.0 SDK. Tomada de [8].

4.1.1. Movimientos

La estructura del AR.Drone está conformada por cuatro motores unidos en forma de cruz donde el hardware de radio frecuencia y la batería se encuentra en el centro. Cada par de motores opuestos giran de la misma forma. Un par gira en el sentido de las manecillas del reloj y el otro en el sentido contrario. Tiene cuatro movimientos distintos que hacen posible un sinfín de maniobras en el aire, entre las que destacan las siguientes:

1. Elevación (Throttle)
2. Giro lateral (Roll)
3. Giro hacia adelante (Pitch)
4. Giro sobre el mismo eje (Yaw)

En la Figura **4-3** podemos apreciar los cuatro tipos de movimientos:

De esta manera, las maniobras son obtenidas cambiando los ángulos de Pitch, Roll y de Yaw. Variando la velocidad de los motores izquierdo y derecho de forma opuesta (es decir, a uno se le aumenta y al otro se le disminuye) se logra un cambio en el ángulo de Roll y

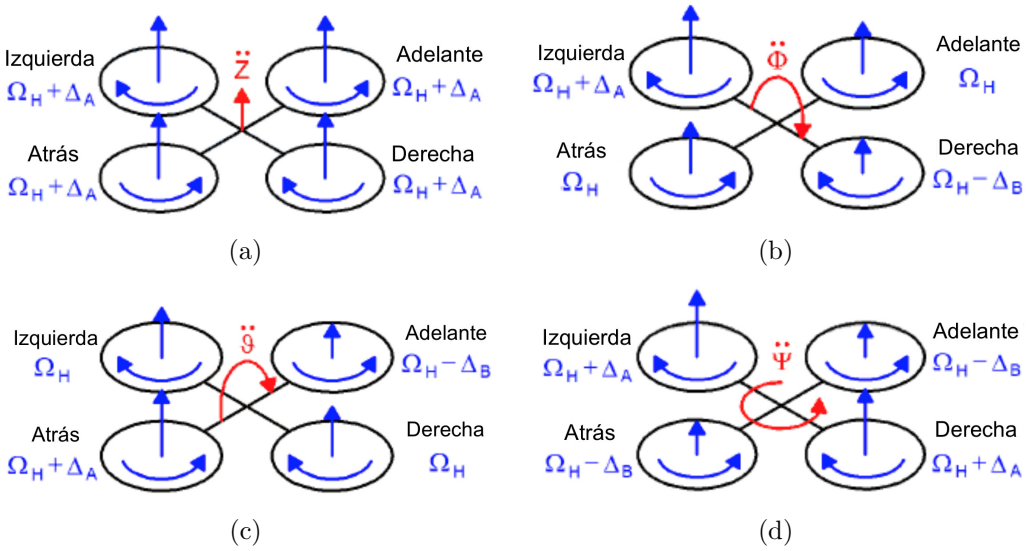


Figura 4-3: Movimientos del AR.Drone. (a) Eleva o baja el cuadricóptero. (b) Mueve el cuadricóptero hacia la derecha o hacia la izquierda. (c) Mueve el cuadricóptero hacia adelante o hacia atrás. (d) Hace que el cuadricóptero gire sobre su mismo eje en sentido horario o sentido antihorario. Tomada de [8].

hace que el cuadricóptero se mueva hacia su derecha o su izquierda. Variando la velocidad de los motores delantero y trasero de forma opuesta se logra un movimiento hacia adelante o hacia atrás. Variando la velocidad de cada par de motores de forma opuesta, se logra un giro sobre su propio eje hacia la izquierda o hacia la derecha.

4.1.2. Video

El AR.Drone 2.0 utiliza códecs de video estándar, con una encapsulación para administrar el flujo por la red.

Códecs de video

El AR.Drone 2.0 utiliza la norma H264 (MPEG4.10) para el flujo de video de alta calidad y la grabación de video. Los siguientes parámetros pueden ser ajustados para el flujo en vivo H264:

- FPS: Entre 15 y 30.
- Velocidad de transferencia: Entre 250 kbps y 4 Mbps.
- Resolución: 360p (640x360 píxeles) o 720p (1280x720 píxeles).

Mientras se realiza grabación de video, el codificador por hardware H264 no está disponible para el flujo de video en vivo, en consecuencia el AR.Drone 2.0 utiliza un codificador visual por software MPEG4.2 para el flujo de video en vivo. Los siguientes parámetros pueden ser ajustados para el flujo de video en vivo MPEG4.2:

- FPS: Entre 15 y 30.
- Velocidad de Transferencia: Entre 250 kbps y 1 Mbps.

Encapsulamiento del video en la red

Para la transmisión por la red, los frames (marcos) de video son enviados por medio de encabezados de código, que contienen mucha información acerca de cada frame. Estos encabezados contienen información valiosa y contienen en una estructura los siguientes datos:

- Tamaño del encapsulamiento
- Ancho del flujo codificado (píxeles)
- Largo del flujo codificado (píxeles)
- Posición del frame en el flujo
- Tiempo en milisegundos
- Tipo de Frame (puede ser I-Frame o P-Frame)
- Total de rebanadas de imagen que componen el frame actual
- Posición de la rebanada de imagen en el frame

Cámara del AR.Drone

La cámara frontal es un sensor CMOS con un lente con ángulo de 92°. El AR.Drone 2.0 codifica y envía las imágenes entrantes al dispositivo huésped. Utiliza imágenes con una resolución de 360p (640x360 píxeles) o de 720p (1280x720 píxeles). La frecuencia de actualización puede ser ajustada entre 15 y 30 FPS y una tasa de transferencia entre 250 kbps y 4 M bps. Utiliza un perfil base de codificación H264. Transmisión de baja latencia. Foto JPEG, almacenamiento de videos durante el vuelo con el dispositivo remoto y almacenamiento instantáneo de videos con Wi-Fi, directamente en tu dispositivo remoto o en una memoria USB. La cámara frontal del AR.Drone se muestra en la Figura **4-4**.

(a) (b)

Figura 4-4: Cámara frontal del AR.Drone 2.0.

4.1.3. Comunicación

El AR.Drone 2.0 puede ser controlado por cualquier dispositivo que soporta WiFi. Su control es realizado por tres servicios de comunicación principales.

El control y la configuración del drone es realizado por medio del envío de comandos AT en el puerto UDP 5556. La latencia de transmisión de los comandos de control son críticos para la experiencia de usuario. Estos comandos tienen que ser enviados de manera regular (usualmente 30 veces por segundo).

La información acerca del drone, como el estado, su posición, velocidad de rotación de los motores, llamado navdata (de navigation data), son enviados por el drone a su cliente en el puerto UDP 5554.

La secuencia de video es enviada por el AR.Drone 2.0 al dispositivo cliente por el puerto TCP 5555. Las imágenes de este video son codificadas de acuerdo al formato discutido en las secciones anterirores.

Otro canal de comunicación, llamado control port, puede ser establecido por el puerto TCP 5559 para enviar información crítica, en oposición a los otros canales de comunicación donde la información puede ser perdida sin efectos peligrosos. Es utilizado para recuperar datos de configuración, y el envío de configuraciones.

EL SDK se comunica a través de un PC externo utilizando los protocolos UDP y TCP para diferentes propósitos. Estos puertos de comunicación se describen en la Tabla **4-1**.

Número de Puerto	Tipo de Comunicación	Función
5554	UDP	Flujo de navegación enviado del AR.Drone al cliente
5555	TCP/IP	Flujo de imágenes enviado del AR.Drone al cliente
5556	UDP	Control y configuración del AR.Drone mediante comandos
5559	TCP/IP	Transferencia de información crítica del sistema

Tabla 4-1: Puertos de comunicación utilizados por el AR.Drone.

Transmisión del flujo de video

El flujo de video del AR.Drone 2.0 es transmitido por el puerto TCP 5555. El drone comenzará enviando un frame inmediatamente cuando un cliente se conecta al socket. Un frame puede ser enviado en varios paquetes TCP, y por lo tanto debe ser ensamblado de nuevo por la aplicación antes de llegar al decodificador de video.

Grabación del flujo de video

La grabación de video utiliza el puerto TCP 5553 para transmitir los frames H264-720p. Este flujo de datos es deshabilitado cuando la aplicación no realiza la grabación. Este flujo utiliza el mismo encapsulamiento para su transmisión al igual que el flujo de video en vivo. La conversión del códec H264 y un archivo .mov/.mp4 es realizada por la aplicación.

4.1.4. Control del AR.Drone

El control del cuadricóptero se realiza por medio de una plataforma que permita ejecutar las actividades más rutinarias. Esto de acuerdo a los movimientos explicados en la sección 4.1.1. Una plataforma que permite hacer las maniobras del AR.Drone 2.0 es la plataforma de Parrot [23], la cual permite controlar remotamente el cuadricóptero desde algún dispositivo programable con una tarjeta de red WiFi con TCP/UDP/IP, sin hacer significativas modificación del código. El SDK no permite un acceso directo al hardware del drone.

4.1.5. AR.Drone SDK

La biblioteca AR.Drone se provee actualmente como una biblioteca de código abierto. Contiene una parte denominada ARDroneTool, la cual implementa de manera eficiente los cuatro servicios descritos en la sección 4.1.3.

Provee:

- Un hilo de administración de comandos AT, que recolecta comandos enviados por todos los otros hilos, y los envía de una manera ordenada con una secuencia numérica correcta.
- Un hilo navdata de administración que recibe automáticamente un flujo de datos de navegación, lo decodifica, y provee a la aplicación del cliente de datos de navegación listos para usarse mediante una llamada a una función.
- Un hilo de administración del video, el cual recibe automáticamente el flujo de video y provee a la aplicación de video listo para usar mediante la llamada a una función.
- Un hilo para grabación de video, que administra la grabación del flujo en HD y la encapsulación .mp4/.mov.
- Un hilo de control que maneja solicitudes de otros hilos para enviar comandos seguros del drone.
- Un conjunto de hilos para AR.Drone Academy 1, que automáticamente recibe las imágenes (formato .jpg) por FTP.

Administración de video

El SDK incluye métodos para administrar el flujo de video. El proceso completo es manejado por un video pipeline (conducto), construido como una secuencia de stages (etapas) el cual desempeña pasos básicos, como recibir los datos del video de un socket, decodificar los marcos y desplegarlos visualmente.

4.1.6. RT_ARDrone

Sebastien Druon es un investigador de la Universidad de Montpellier, Francia, que ha desarrollado entre otros controladores de código abierto para robots, el RT_ARDrone [24], en colaboración con otros investigadores. Es un framework para sistemas Linux que permite controlar el AR.Drone 2.0 de manera fácil y rápida, desarrollado en lenguaje C, el cual puede encontrarse en su cuenta de Github. Es este framework el que se ha utilizado en este trabajo de tesis, y a través del cual se ha estructurado el proyecto como se describe en el Capítulo 4.

4.1.7. ROS

ROS (Robot Operative System) es un framework para escribir software para robots de código abierto. Es un conjunto de herramientas, bibliotecas, y convenios que tienen por objeto

simplificar la tarea de crear comportamientos complejos y robustos a través de una amplia variedad de plataformas robóticas [25].

ROS ha permitido crear una extensa cantidad de algoritmos que ayudan a manipular diversos modelos de dispositivos robóticos que se encuentran en el mundo científico. Entre estos frameworks se encuentra el controlador *ardrone_autonomy*.

4.1.8. Ardrone_autonomy

Ardrone_autonomy es un controlador de ROS para el cuadricóptero AR.Drone 1.0 y 2.0. Éste controlador está basado en el SDK del AR.Drone ([26], [27]). Es un controlador desarrollado por el grupo Autonomy Lab [28] de la Universidad Simon Fraser [29]. Cuenta con su propia documentación [30].

Consideraciones

Las consideraciones que deben tomarse en cuenta al ejecutar pruebas del entrenamiento del detector contemplan:

- Tener en cuenta que el programa puede consumir una cantidad considerable de memoria principal, memoria del disco duro y tiempo de ejecución, dependiendo de la cantidad de muestras de entrenamiento. Para el número de imágenes utilizado dichas consideraciones son importantes de analizar y comparar.
- Tener en cuenta que, existen limitaciones para el tamaño máximo de archivo que puede tener efecto al escribir el archivo de características. Estas limitaciones se observan más en casos de sistemas operativos de 32bits, por lo que es necesario realizar pruebas en un sistema de 64 bits.

4.2. Entrenamiento del clasificador

La etapa de clasificación se realiza utilizando el modelo entrenado, es por ello que antes de llevar a cabo las pruebas del detector es necesario probar y evaluar el rendimiento del entrenamiento. En esta sección se incluyen los resultados del entrenamiento del clasificador que se utilizó para hacer la detección de personas. Los valores de salida proporcionados como resultado de la ejecución del algoritmo de entrenamiento con la SVM son los siguientes:

- Verdadero Positivo (**VP**): Corresponde al número de detecciones hechas que sí corresponden a una persona. Si esto sucede se dice que se tuvo una clasificación exitosa.
- Verdadero Negativo (**VN**): Corresponde al número de rechazos exitosos, es decir, que no detectó una persona y en efecto no había.

- Falso Positivo (**FP**): Corresponde al número de detecciones positivas y que en realidad no corresponden a una persona, es decir, se tiene una falsa detección.
- Falso Negativo (**FN**): Corresponde a las personas que no fueron detectadas a pesar de estar presentes, es decir, se tiene un falso rechazo.

Usando el algoritmo de entrenamiento y aprendizaje desarrollado en 3.2 se realizaron las pruebas de rendimiento. Cabe mencionar que la eficiencia del clasificador se define, entre otras cosas, tomando en cuenta los valores: FP, FN, VP y VN, que van en función del número de imágenes negativas y positivas usadas para el entrenamiento y aprendizaje. Es por esta razón que se produjeron varias corridas del algoritmo modificando el tamaño del dataset positivo y negativo hasta llegar a un conjunto donde el número de valores FP Y FN, así como VP y VN hayan sido los mejores. A continuación se muestran los valores de salida de las corridas antes mencionadas.

Prueba 1: Se incian las pruebas tomando el tamaño de los conjuntos establecidos en la sección 3.2.1, que son: 8335 imágenes positivas y 7025 imágenes negativas. Los valores de salida correspondientes se muestran en la Figura **4-5**.

Prueba 2: Se disminuyó el tamaño del conjunto negativo, la finalidad es realizar una prueba con un dataset igualitario entre positivos y negativos y ver los valores proporcionados. Es decir, el número de imágenes son: 7025 positivos y 7025 negativos. Los valores de salida se muestran en la Figura **4-6**.

Prueba 3: Resulta de interés poder conocer los valores resultado de la ejecución de pruebas al disminuir el tamaño del conjunto negativo. Es por ello que esta prueba contiene: 8335 imágenes positivas y 6023 negativas. El resultado de esta prueba se muestra en la Figura **4-7**.

Prueba 4: Se continuó con el principio de reduccción del conjunto negativo como se menciona en la prueba anterior. La reducción del conjunto negativo fue de 2000 muestras, quedando un tamaño de: 8335 positivos y 4020 negativos. Los valores de salida de la prueba se muestran en la Figura **4-8**

Prueba 5: El principio de reducción de muestras negativas continúa en esta prueba, teniendo conjuntos muestra de tamaño: 8335 positivos y 3220 negativos. Los valores de salida de la prueba se muestran en la Figura **4-9**.

Prueba 6: Se finalizan las pruebas con reducción de muestras negativas teniendo conjuntos finales de: 8335 positivos y 2757 negativos. Los valores de salida de la prueba se muestran en la Figura **4-10**.

Prueba 7: Se desarrollaron también pruebas con el principio de reduccción en el conjunto positivo, para posteriormente analizar los resultados obtenidos. El tamaño de los conjunto es de: 5497 positivos y 7025 negativos. Los valores de salida de la prueba se muestran en la Figura **4-11**.

Prueba 8: Se continuó con el principio de reduccción del conjunto positivo. La reducción para esta prueba tiene un tamaño de: 4218 positivos y 7025 negativos. Los valores de salida de la prueba se muestran en la Figura **4-12**.

Prueba 9: Se finalizan las pruebas con el principio de reducción de conjuntos positivos, los tamaños finales son: 3900 positivos y 7025 negativos. Los valores de salida de la prueba se muestran en la Figura **4-13**.

```
Leyendo archivos, generando características HOG y guardando en el archivo.
15360 (100%):    Leyendo el archivo:'neg/010152_2.jpg'_001011.jpg'pg'
Llamando  SVMlight
Finalizado
Leyendo ejemplos en la memoria...Finalizado con 15360 ejemplos leídos
Optimizando
Finalizado con 7908 iteraciones
Optimización terminada (maxdiff=0.00098).
Tiempo de ejecución (segundos de cpu) 371.72
Número de SV: 11986 (incluyendo 10654 en el límite superior)
L1 loss: loss=2898.81063
Norma del vector w: |w|=2.42359
Norma del vector de ejemplo más largo: |x|=10.23493
Número de evaluaciones del kernel: 685883
Entrenamiento finalizado, guardando el modelo en archivo.
Finalizado
Generando la representación de características del vector HOG utilizando svmlight
Calculando el vector descriptor único
Tamaño del Vector resultante 3780
Guardando el vector del descriptor en el archivo
3779 (100%)
Resultados:
        Positivos verdaderos: 8227
        Negativos verdaderos: 6898
        Falsos Positivos: 127
        Falsos Negativos: 108
```

Figura 4-5: Valores de salida de la prueba 1 del entrenamiento.

```
Leyendo archivos, generando características HOG y guardando en el archivo.
14050 (100%):    Leyendo el archivo:'neg/010152_2.jpg'_001011.jpg'pg'
Llamando  SVMlight
Finalizado
Leyendo ejemplos en la memoria...Finalizado con 14050 ejemplos leídos
Optimizando
Finalizado con 7438 iteraciones
Optimización terminada (maxdiff=0.00100).
Tiempo de ejecución (segundos de cpu) 346.24
Número de SV: 10918 (incluyendo 9618 en el límite superior)
L1 loss: loss=2602.46634
Norma del vector w: |w|=2.37066
Norma del vector de ejemplo más largo: |x|=10.23493
Número de evaluaciones del kernel: 638489
Entrenamiento finalizado, guardando el modelo en archivo.
Finalizado
Generando la representación de características del vector HOG utilizando svmlight
Calculando el vector descriptor único
Tamaño del Vector resultante 3780
Guardando el vector del descriptor en el archivo
3779 (100%)
Resultados:
        Positivos verdaderos: 6907
        Negativos verdaderos: 6940
        Falsos Positivos: 85
        Falsos Negativos: 118
```

Figura 4-6: Valores de salida de la prueba 2 del entrenamiento.

```
Leyendo archivos, generando características HOG y guardando en el archivo.
14358 (100%):    Leyendo el archivo:'neg/magdalen_000000.jpg''jpg''''
Llamando  SVMlight
Finalizado
Leyendo ejemplos en la memoria...Finalizado con 14358 ejemplos leídos
Optimizando
Finalizado con 6913 iteraciones
Optimización terminada (maxdiff=0.00100).
Tiempo de ejecución (segundos de cpu) 349.28
Número de SV: 11261 (incluyendo 9967 en el límite superior)
L1 loss: loss=2751.60398
Norma del vector w: |w|=2.42084
Norma del vector de ejemplo más largo: |x|=10.23493
Número de evaluaciones del kernel: 614955
Entrenamiento finalizado, guardando el modelo en archivo.
Finalizado
Generando la representación de características del vector HOG utilizando svmlight
Calculando el vector descriptor único
Tamaño del Vector resultante 3780
Guardando el vector del descriptor en el archivo
3779 (100%)
Resultados:
        Positivos verdaderos: 8239
        Negativos verdaderos: 5885
        Falsos Positivos: 138
        Falsos Negativos: 96
```

Figura 4-7: Valores de salida de la prueba 3 del entrenamiento.

```
Leyendo archivos, generando características HOG y guardando en el archivo.
12355 (100%):    Leyendo el archivo:'neg/magdalen_000000.jpg'082.jpg'
Llamando  SVMlight
Finalizado
Leyendo ejemplos en la memoria...Finalizado con 12355 ejemplos leídos
Optimizando
Finalizado con 6012 iteraciones
Optimización terminada (maxdiff=0.00098).
Tiempo de ejecución (segundos de cpu) 263.56
Número de SV: 9654 (incluyendo 8439 en el límite superior)
L1 loss: loss=2320.82103
Norma del vector w: |w|=2.35986
Norma del vector de ejemplo más largo: |x|=10.23493
Número de evaluaciones del kernel: 532537
Entrenamiento finalizado, guardando el modelo en archivo.
Finalizado
Generando la representación de características del vector HOG utilizando svmlight
Calculando el vector descriptor único
Tamaño del Vector resultante 3780
Guardando el vector del descriptor en el archivo
3779 (100%)
Resultados:
        Positivos verdaderos: 8286
        Negativos verdaderos: 3865
        Falsos Positivos: 155
        Falsos Negativos: 49
```

Figura 4-8: Valores de salida de la prueba 4 del entrenamiento.

```
Leyendo archivos, generando características HOG y guardando en el archivo.
11555 (100%):    Leyendo el archivo:'neg/magdalen_000000.jpg'082.jpg'
Llamando  SVMlight
Finalizado
Leyendo ejemplos en la memoria...Finalizado con 11555 ejemplos leídos
Optimizando
Finalizado con 5526 iteraciones
Optimización terminada (maxdiff=0.00099).
Tiempo de ejecución (segundos de cpu) 245.82
Número de SV: 8900 (incluyendo 7704 en el límite superior)
L1 loss: loss=2075.06983
Norma del vector w: |w|=2.29434
Norma del vector de ejemplo más largo: |x|=10.23493
Número de evaluaciones del kernel: 492345
Entrenamiento finalizado, guardando el modelo en archivo.
Finalizado
Generando la representación de características del vector HOG utilizando svmlight
Calculando el vector descriptor único
Tamaño del Vector resultante 3780
Guardando el vector del descriptor en el archivo
3779 (100%)
Resultados:
        Positivos verdaderos: 8293
        Negativos verdaderos: 3043
        Falsos Positivos: 177
        Falsos Negativos: 42
```

Figura 4-9: Valores de salida de la prueba 5 del entrenamiento.

```
Leyendo archivos, generando características HOG y guardando en el archivo.
12522 (100%):    Leyendo el archivo:'neg/010152_2.jpg'.jpg'g's.jpg'g'
Llamando  SVMlight
Finalizado
Leyendo ejemplos en la memoria...Finalizado con 12522 ejemplos leídos
Optimizando
Finalizado con 6876 iteraciones
Optimización terminada (maxdiff=0.00098).
Tiempo de ejecución (segundos de cpu) 324.09
Número de SV: 9790 (incluyendo 8556 en el límite superior)
L1 loss: loss=2320.88642
Norma del vector w: |w|=2.29038
Norma del vector de ejemplo más largo: |x|=10.23493
Número de evaluaciones del kernel: 582803
Entrenamiento finalizado, guardando el modelo en archivo.
Finalizado
Generando la representación de características del vector HOG utilizando svmlight
Calculando el vector descriptor único
Tamaño del Vector resultante 3780
Guardando el vector del descriptor en el archivo
3779 (100%)
Resultados:
        Positivos verdaderos: 5363
        Negativos verdaderos: 6963
        Falsos Positivos: 62
        Falsos Negativos: 134
```

Figura 4-10: Valores de salida de la prueba 6 del entrenamiento.

```
Leyendo archivos, generando características HOG y guardando en el archivo.
11092 (100%):    Leyendo el archivo:'neg/oxford_000884.jpg'.jpg'.jpg'
Llamando  SVMlight
Finalizado
Leyendo ejemplos en la memoria...Finalizado con 11092 ejemplos leídos
Optimizando
Finalizado con 5473 iteraciones
Optimización terminada (maxdiff=0.00099).
Tiempo de ejecución (segundos de cpu) 230.59
Número de SV: 8460 (incluyendo 7264 en el límite superior)
L1 loss: loss=1922.07344
Norma del vector w: |w|=2.24757
Norma del vector de ejemplo más largo: |x|=10.23493
Número de evaluaciones del kernel: 481628
Entrenamiento finalizado, guardando el modelo en archivo.
Finalizado
Generando la representación de características del vector HOG utilizando svmlight
Calculando el vector descriptor único
Tamaño del Vector resultante 3780
Guardando el vector del descriptor en el archivo
3779 (100%)
Resultados:
        Positivos verdaderos: 8308
        Negativos verdaderos: 2569
        Falsos Positivos: 188
        Falsos Negativos: 27
```

Figura 4-11: Valores de salida de la prueba 7 del entrenamiento.

```
Leyendo archivos, generando características HOG y guardando en el archivo.
11243 (100%):    Leyendo el archivo:'neg/010152_2.jpg'00043.jpg'g''g'
Llamando  SVMlight
Finalizado
Leyendo ejemplos en la memoria...Finalizado con 11243 ejemplos leídos
Optimizando
Finalizado con 6408 iteraciones
Optimización terminada (maxdiff=0.00100).
Tiempo de ejecución (segundos de cpu) 270.57
Número de SV: 8398 (incluyendo 7175 en el límite superior)
L1 loss: loss=1634.88961
Norma del vector w: |w|=2.10745
Norma del vector de ejemplo más largo: |x|=10.23493
Número de evaluaciones del kernel: 534981
Entrenamiento finalizado, guardando el modelo en archivo.
Finalizado
Generando la representación de características del vector HOG utilizando svmlight
Calculando el vector descriptor único
Tamaño del Vector resultante 3780
Guardando el vector del descriptor en el archivo
3779 (100%)
Resultados:
        Positivos verdaderos: 4162
        Negativos verdaderos: 7002
        Falsos Positivos: 23
        Falsos Negativos: 56
```

Figura 4-12: Valores de salida de la prueba 8 del entrenamiento.

```
Leyendo archivos, generando características HOG y guardando en el archivo.
10925 (100%):    Leyendo el archivo:'neg/010152_2.jpg'87.jpg'181.jpg'
Llamando  SVMlight
Finalizado
Leyendo ejemplos en la memoria...Finalizado con 10925 ejemplos leídos
Optimizando
Finalizado con 6573 iteraciones
Optimización terminada (maxdiff=0.00099).
Tiempo de ejecución (segundos de cpu) 264.33
Número de SV: 8148 (incluyendo 6935 en el límite superior)
L1 loss: loss=1616.15039
Norma del vector w: |w|=2.08632
Norma del vector de ejemplo más largo: |x|=10.23493
Número de evaluaciones del kernel: 538854
Entrenamiento finalizado, guardando el modelo en archivo.
Finalizado
Generando la representación de características del vector HOG utilizando svmlight
Calculando el vector descriptor único
Tamaño del Vector resultante 3780
Guardando el vector del descriptor en el archivo
3779 (100%)
Resultados:
        Positivos verdaderos: 3829
        Negativos verdaderos: 7000
        Falsos Positivos: 25
        Falsos Negativos: 71
```

Figura 4-13: Valores de salida de la prueba 9 del entrenamiento.

Una vez finalizadas las pruebas del entrenamiento se prosigue con el análisis de los valores de salida proporcionados por cada una de ellas. La Tabla **4-2** muestra una comparación entre los resultados.

Variable	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6	Prueba 7	Prueba 8	Prueba 9
Total de muestras	15360	14050	14358	12355	11555	11092	12522	11243	10925
Muestras positivas	8335	7025	8335	8335	8335	8335	5497	4218	3900
Muestras negativas	7025	7025	6023	4020	3220	2757	7025	7025	7025
Verdaderos positivos	8227	6907	8239	8286	8293	8308	5363	4162	3829
Verdaderos negativos	6898	6940	5885	3865	3043	2569	6963	7002	7000
Falsos positivos	127	85	138	155	177	188	62	23	25
Falsos negativos	108	118	96	49	42	27	134	56	71
Tiempo de ejecución (s)	371.72	346.24	349.28	263.56	245.82	230.59	324.09	270.57	264.33
L1 loss	2898.81	2602.46	2751.6	2320.92	2075.06	1922.07	2323.88	1634.88	1616.15
No. iteraciones	7908	7438	6913	6012	5526	5473	6876	6408	6573
Norma $\|\|\mathbf{w}\|\|$	2.4259	2.37066	2.42084	2.35986	2.29434	2.24757	2.29038	2.10745	2.08632
Evaluaciones del Kernel	685883	638489	614955	532537	492345	481628	582803	534981	538854

Tabla 4-2: Valores de salida de las pruebas realizadas.

Se puede observar, que la columna correspondiente a la prueba número 8 presenta los mejores resultados en los valores de salida respecto a la norma $||w||$ y los valores VP, VN, FP y FN. El éxito de las máquinas de Soporte Vectorial radica en su capacidad automática de minimizar $||w||$ (donde $||w||$ significa la norma del vector w) extraído de un pequeño número de vectores de soporte de los datos de entrenamiento que son relevantes para la clasificación. Finalmente se ha seleccionado el mejor tamaño del dataset que corresponde a un total de 11243 imágenes muestra donde 4218 corresponden a muestras positivas y 7025 a negativas.

Tener definido el tamaño final del dataset así como sus correspondientes valores de salida, permite realizar el análisis correspondiente al rendimiento del entrenamiento. A continuación se desarrollan las ecuaciones necesarias para determinar la eficacia del entrenamiento. Las medidas de desempeño fueron:

La sensibilidad es la fracción de casos VP clasificados como positivos respecto a los VP.

$$sensibilidad = \frac{V}{VP+FN}$$

La especificidad es la fracción de los casos VN, clasificados como negativos con respecto a los VN.

$$especificidad = \frac{VN}{VN+FP}$$

La precisión es la calidad de la respuesta del clasificador, mientras su valor sea más cercano a 1, mejor será la clasificación.

$$precisión = \frac{VP}{VP+FP}$$

Es la proximidad entre el resultado y la clasificación perfecta, mientras su valor sea más cercano a 1, mejor será la clasificación.

$$exactitud = \frac{VP+VN}{VP+FN+VN+FP}$$

Los cálculos se realizaron tomando los valores de salida de la prueba número 8 como se definió anteriormente (VP=4162, VN=7002, FP=23 y FN=56) y los valores resultantes son:

sensibilidad = 0.9867, especificidad = 0.9967, precisión = 0.9945 y exactitud = 0.9929.

Los resultados demostraron que el clasificador obtenido con entrenamiento realizado tuvo alta precisión, es decir, más del 98 % de los casos fueron clasificados correctamente como positivos, menos del 6 % de los casos normales fueron clasificados como falsos positivos y más del 99 % de los casos fueron clasificados correctamente como negativos.

4.3. Detector de personas

La presente sección muestra las pruebas de funcionamiento del detector de personas, mismas que se realizaron en las instalaciones de la Benemérita Universidad Autónoma de Puebla. Los escenarios elegidos para desarrollar las pruebas son los que muestran mayor afluencia de peatones y que por consiguiente representan mejores resultados de salida, cabe mencionar que las pruebas se realizaron en tiempo real utilizando una cámara web. Como se mencionó anteriormente las condiciones de luz y oclusiones entre los peatones son características importantes para el análisis de los resultados que se obtengan, así como el ángulo de visión de la cámara utilizada. Tomando esto en consideración se definieron diferentes horas del día para contar con diferentes intensidades de luz.

Las imágenes que a continuación se muestran fueron tomadas del conjunto total de pruebas realizadas y fueron clasificadas de acuerdo al tipo de iluminación, teniendo así pruebas con buena iluminación en la Figura **4-14**, con iluminación media en la Figura **4-15** e iluminación baja en la Figura **4-16**. Además de que estas imágenes son las más significativas por contener ejemplos con personas en diferentes distancias, con varias oclusiones, etc.

(a) Frame 1 (b) Frame 2 (c) Frame 3

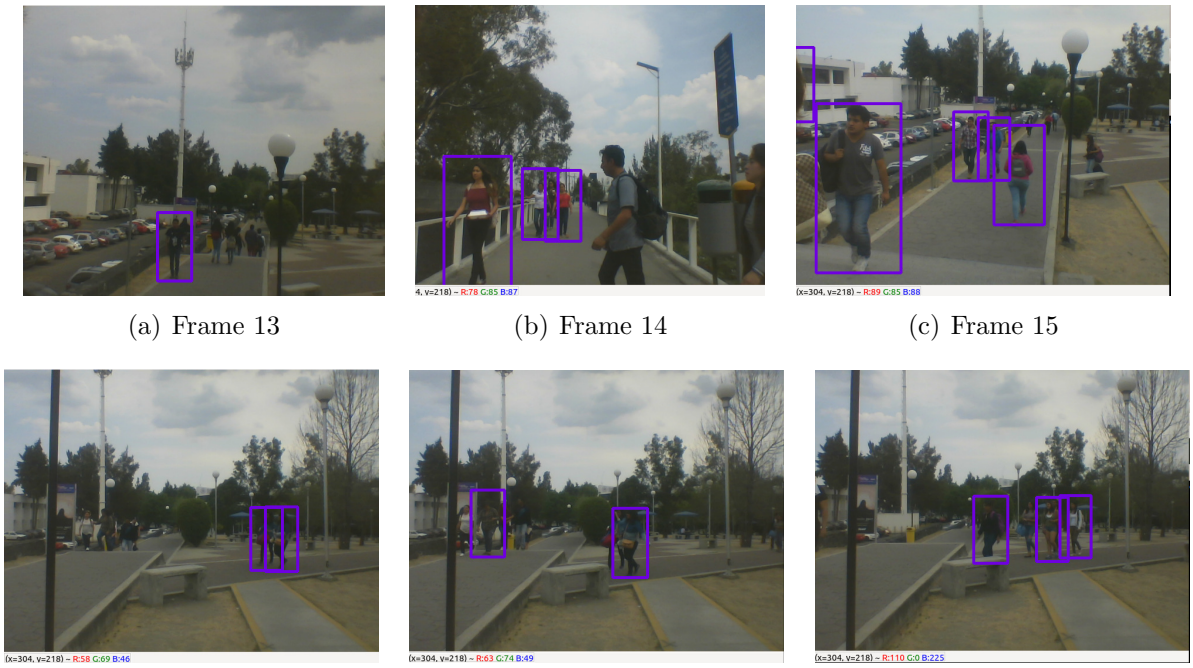
(d) Frame 4 (e) Frame 5 (f) Frame 6

Figura 4-14: Detector de personas en buenas condiciones de luz.

(a) Frame 7 (b) Frame 8 (c) Frame 9

(d) Frame 10 (e) Frame 11 (f) Frame 12

Figura 4-15: Detector de personas en condiciones de luz medianamente buenas.



(a) Frame 13 (b) Frame 14 (c) Frame 15

(d) Frame 16 (e) Frame 17 (f) Frame 18

Figura 4-16: Detector de personas con baja iluminación.

Frame	Aciertos	No detecciones	Errores	Personas totales
1	5	0	0	5
2	3	0	0	3
3	4	1	0	5
4	2	1	0	3
5	2	1	0	3
6	5	3	0	8
7	3	1	0	4
8	3	0	0	3
9	2	0	0	2
10	2	0	0	2
11	3	1	0	4
12	3	0	0	3
13	1	2	0	3
14	3	1	0	4
15	4	0	1	4
16	2	3	0	5
17	2	4	0	6
18	3	1	0	4
Totales	52	19	1	71

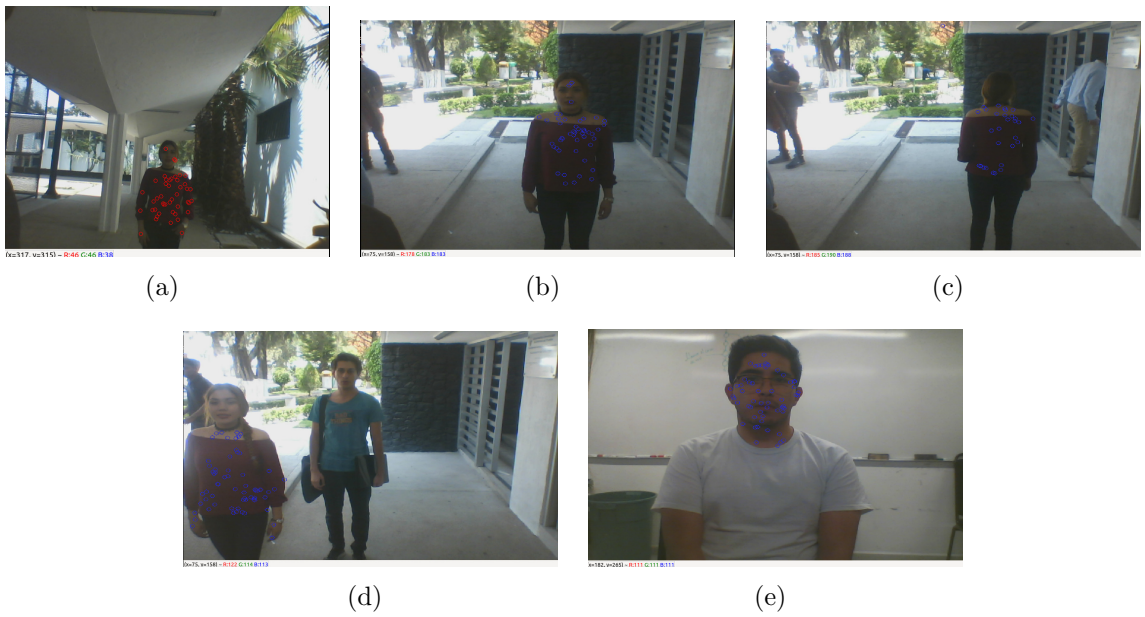
Tabla 4-3: Parámetros del detector.

La Tabla **4-3** muestra la información referente al comportamiento del detector en tiempo real basado en las pruebas realizadas, para ello se generó un conteo manual contemplando el porcentaje de aciertos y de errores para tener una noción de la eficacia del detector.

En análisis del detector de acuerdo a las detecciones proporcionadas por las pruebas realizadas nos permite saber los porcentajes de aciertos y errores en una fracción de las muestras totales, los valores que se obtienen son: 73.23 % de aciertos, 26.76 % de no detecciones y 1 una detección incorrecta por cada 18 frames.
Se puede observar que los resultados no son tan óptimos como se desearía, sin embargo cabe aclarar que el mayor número de no detecciones se presenta del frame 13 al 18, los cuales corresponden a las pruebas con poca luz, se puede afirmar entonces que la eficacia del detector está directamente relacionado con las condiciones de iluminación del escenario en el que se aplique. Los porcentajes que se obtienen sin tomar en cuenta los frames con poca luz muestran 83 % de aciertos y 23 % de errores, esto nos comprueba la afirmación antes realizada.
Finalmente, también es necesario comentar que una porción de las personas no detectadas corresponde a personas que se encontraban muy alejados del rango de visión de la cámara, por lo que es importante tener en cuenta que entre má alejada se encuentre una persona de la cámara la probabilidad de ser detectada correctamente disminuye.

4.4. Seguidor de personas

Una vez que se realizó la detección de personas, se aplicó el filtro de partículas para hacer el seguimiento de la persona. Se inició probando el generador de partículas, en la Figura **4-17** se muestran los resultados obtenidos para la primera fase del seguidor, que es calcular las partículas. Se puede apreciar claramente la forma en que dichas partículas se generan en la persona detectada previamente. También se puede observar comparando los incisos a) con c) que la iluminación es determinante para la generación de partículas en la persona a seguir, y muestra su eficacia al pertenecer con la persona que se esta siguiendo y no cambiar a otra que se encuentre en la escena.



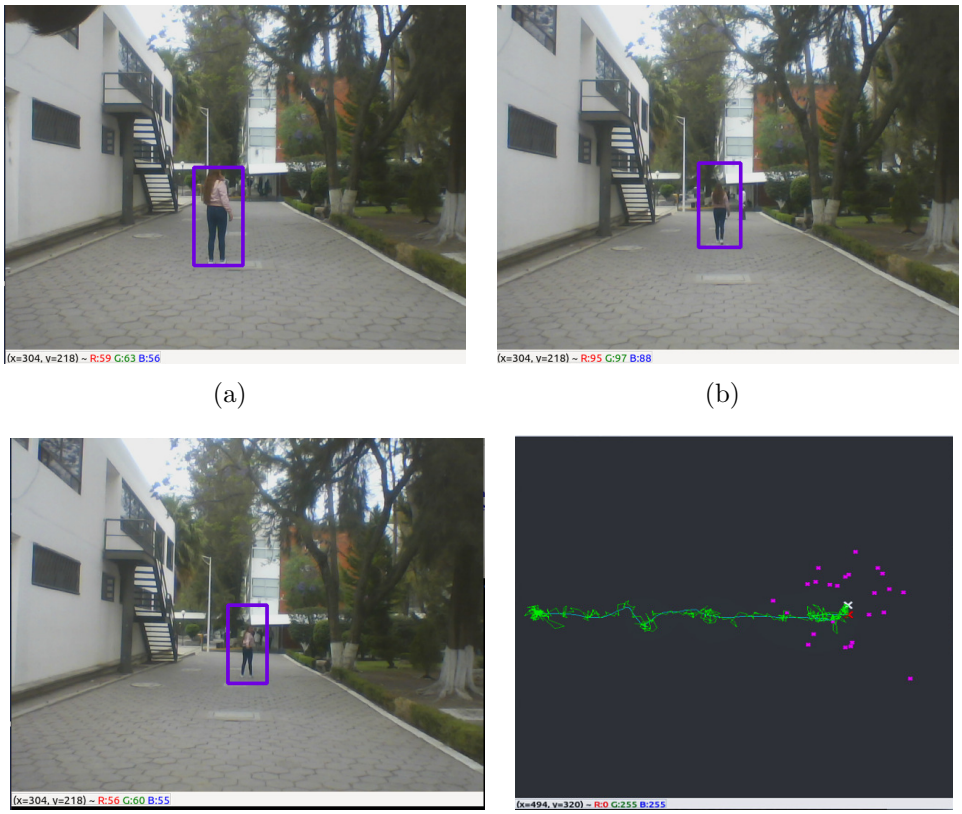
(a) (b) (c)

(d) (e)

Figura 4-17: Cálculo de partículas.

Después se integró el algoritmo seguidor final. Se puede observar en la Figura **4-18** el filtro de partículas, utilizando una cámara web, cuando va siguiendo a una persona que es reconocida por el HOG. En color fucsia se ven las partículas, en color verde se plasma una línea que recopila los estados que han sucedido en el tiempo que se realiza el seguimiento, con una cruz roja se observa el estado actual de la persona y con una cruz blanca se observa el estado actual que el filtro de partículas predice.

Después de la varias pruebas realizadas en distintos lugares (exteriores), se pudo observar que los resultados que se obtienen son óptimos, puesto que la persona es seguida un 100 % del tiempo. La Figura **4-18** muestra una secuencia de imágenes de una de las experiencias donde se puede observar la calidad de los resultados del seguimiento.



(a) (b)

(c) (d)

Figura 4-18: Pruebas del filtro de partículas (vistas de la cámara y la consola).

4.5. Inferencias de los parámetros

Como se mencionó al principio de este capítulo, la finalidad de realizar pruebas y análisis independientes para cada uno de los algoritmos implementados fue poder llevar acabo la selección de los mejores parámetros correspondientes a cada algoritmo. Un ejemplo de esta selección es el caso del tamaño de las imágenes de entrada del HOG o de la cantidad de partículas necesarias para un correcto seguimiento. Dados los resultados obtenidos y con base en el análisis realizado para cada uno de ellos podemos inferir que los parámetros propuestos son los más óptimos para realizar la tarea respectiva de cada algoritmo. Este paso fue de vital importancia en el desarrollo de la propuesta planteada en este trabajo de tesis, debido a que nos da pauta a la integración de los algoritmos para tener un seguidor de personas en tiempo real. Es por ello que al finalizar las pruebas aquí desarrolladas se procedió a crear un algoritmo robusto que engloba a todos los antes mencionados. El algoritmo final de seguimiento será también integrado con un algoritmo de planificación de movimientos para la realización del seguimiento de personas en tiempo real a través de un vuelo autónomo por parte del AR.Drone.

Capítulo 5

Seguimiento en tiempo real

Finalmente se realizó la integración de los algoritmos desarrollados anteriormente para resolver nuestro objetivo principal. El equipo de pruebas utilizado fue el cuadricóptero AR.Drone 2.0 cuyas características fueron descritas en el Capítulo 4. La intervención del drone para la parte del vuelo se realizó a través de un planificador de movimientos, lo que genera un vuelo autónomo, esta tarea se completó usando el framework TUM_Ardrone. Para la parte de detección y seguimiento de personas, se utilizó la cámara principal del Ar.Drone la cuál utiliza los puertos establecidos del MAV para comunicarse mediante la red local de WiFi y hacer streaming del video capturado.

5.1. Vuelo autónomo del AR.Drone 2.0

Para el vuelo autónomo del MAV es necesario realizar diversas tareas previas, entre ellas la planificación de movimientos. Se inicia con el modelado del escenario donde el Ar.Drone llevará a cabo su vuelo. Posteriormente a través del software OMPL App se realiza la planificación de movimientos bajo dos esquemas que son: Cuerpos rígidos 3D y cuadricóptero, la diferencia de estos radica en la sinemática dada al objeto. Se utilizaron diversos algoritmos de la rama de los RRT (Rapidly-exploring Random Trees), los algoritmos utilizados fueron RRT, RRT Star, RRT Conect, TRRT, SyclopRRT.

Una vez dado los resultados de los diversos algoritmos se evalúa cual de ellos es el óptimo para el vuelvo del drone, ésta medición se fija mediante de 3 resultados, el número de nodos creados, el tiempo y el porcentaje de eficiencia dada. Una vez obtenida la planificación de las coordenadas para el vuelo, mediante un software desarrollado en C++ se interpreta esta solución y se genera un nuevo código bajo el lenguaje de programación utilizado por el framework TUM_Ardrone, además de respetar las cabeceras necesarias donde se definen los parámetros para el vuelo del cuadricóptero y las escalas para el vuelo real. Una vez generado el código final, el framework nos permite la intervención del AR.Drone para llevar a cabo el vuelo autónomo. Éste recibe las instrucciones por parte de nuestro algoritmo y envía la secuencia de información para realizar el vuelo bajo la planificación dada. La implementación

de la planificación de movimientos en tiempo real para lograr el vuelo autómo del AR.Drone 2.0, utilizado en este trabajo de tesis, se encuentra detallada en [31].

5.2. Streaming de video

La dirección ip por la cuál fue realizado el video streaming es: $tcp://192.168.1.1:5555/$ Para recibir información del puerto 5555, que en este caso se refiere a los frames captados por la cámara del drone, únicamente es necesario conectarse a dicho puerto y automáticamente la información será enviada al receptor. AR.Drone únicamente es capaz de enviar datos de una de las cámaras, por tanto sólo una de ellas estará activa a la vez. Se utiliza la cámara frontal del AR.Drone puesto que su ángulo de visión es el correcto para realizar la detección, es decir, permite captar la silueta completa de una persona, la cámara de abajo no puede ser usada para este trabajo debido a que la vista de las personas no es la correcta. Además, la cámara frontal tiene mejor resolución por lo tanto la captación de la iluminación es mejor.

5.3. Escenario de pruebas

Las pruebas en tiempo real se realizaron en un escenario que previamente se modeló a escala en la parte del planificador de vuelos y que además fue el primer escenario en las pruebas de los algoritmos de clasificación y seguimiento. El escenario corresponde al pasillo principal que conecta a la explanada de la Facultad de Ciencias de la computación de la BUAP. La condiciones bajo las que se realizaron las pruebas del seguimiento de personas por parte del MAV son: nula afluencia de personas durante la ejecución, buenas condiciones de iluminación y cambios bajos y ligeros de corrientes de aire.

Figura 5-1: Escenario de las pruebas finales.

El resultado de las pruebas finales en tiempo real fue monitoreado y controlado a través una computadora con las características mencionadas en el Capítulo 4, las ventanas que se muestran en la Figura **5-2** corresponden a la ejecución de los algoritmos, dichas ventanas han sido delimitadas en diferentes colores para poder observar los recursos necesarios trabajando en conjunto para poder llegar al seguimiento de la persona.

Funciones de las ventanas:

- La ventana de color amarillo corresponde a la vista de control para el vuelo del AR.Drone y que pertenece a la herramienta tum_ardrone, en esta ventana es donde se cargan y controlan las instrucciones de vuelo, tal como se menciona en [31].

- La ventana en color verde corresponde a la vista "Map view"del tum_ardrone donde se puede observar la secuencia de vuelo que esté presentando en el momento el AR.Drone.

- La ventana en color rosa corresponde a la salida de la consola donde se están ejecutando las diferentes herramientas y donde se pueden detener los procesos o inciarlos.

- Finalmente la ventana en color azul corresponde a la vista de la secuencia de video en streaming proporcinada por la cámara frontal del AR.Drone y donde se puede apreciar la detección y seguimiento de la persona.

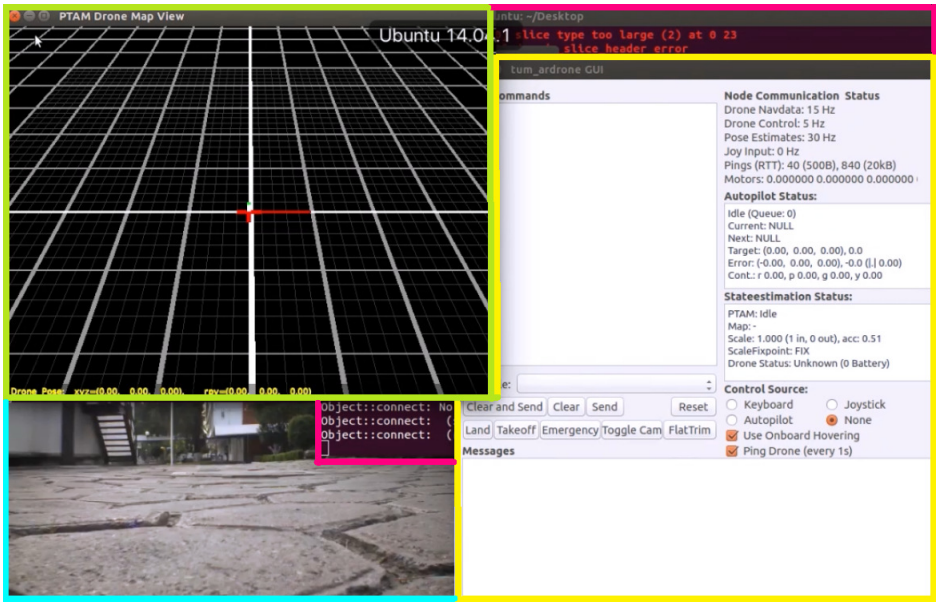
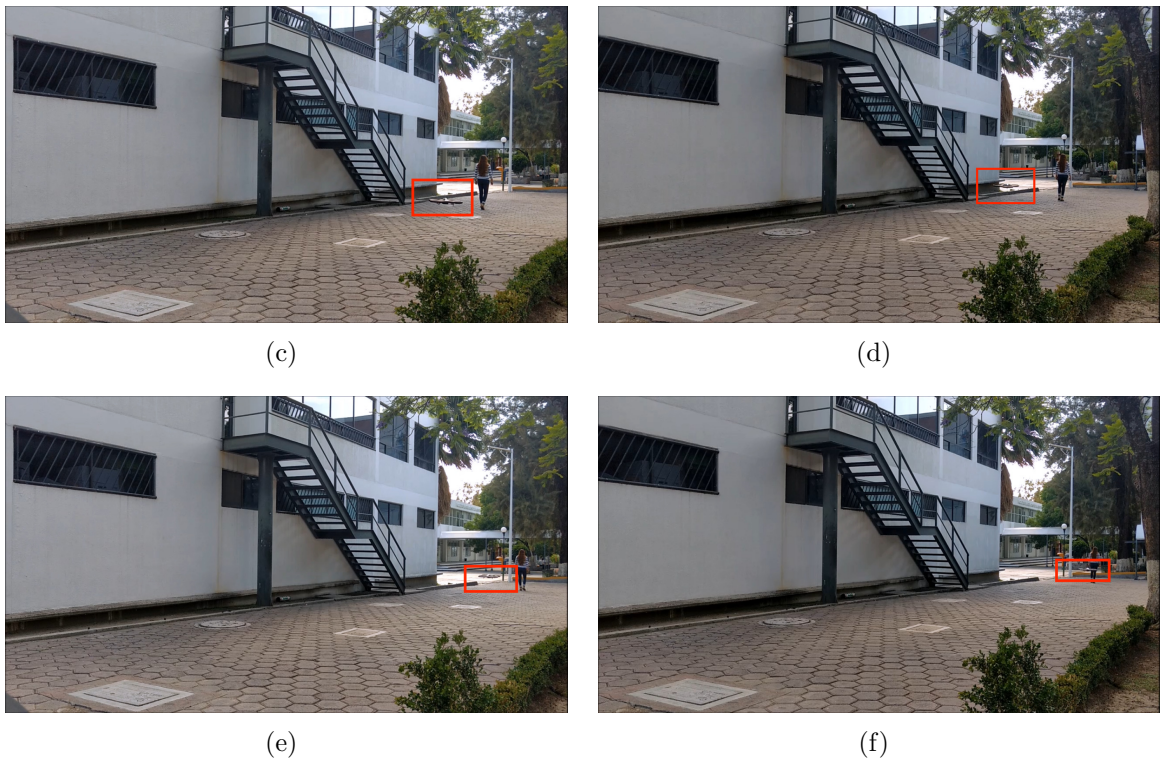


Figura 5-2: Ventanas utilizadas para las pruebas finales.

5.4. Seguimiento autónomo

La Figura **5-3** muestra un conjunto de imágenes tomadas de la secuencia del video que se grabó con una cámara externa para poder apreciar el vuelo y seguimiento de la persona en el escenario de pruebas. Se puede observar en estas imágenes que el AR.Drone conserva una distancia considerable respecto a la persona, lo que es importante para la detección puesto que esto le permite no perder de vista a la persona parcial o totalmente.

(a) (b)



(c) (d)

(e) (f)

Figura 5-3: Seguimiento de una persona con el AR.Drone 2.0.

La tarea de detección y seguimiento de personas es pesada computacionalmente, ya que entre mayor es la imagen que se recibe, más tiempo tarda en obtener resultados adecuados. Para imágenes muy pequeñas, del orden de 64x128 píxeles, el reconocimiento de una persona tarda en promedio 40ms. Para la imagen que se procesa desde el MAV, el tiempo que tarda en hacer el reconocimiento es de 200ms en promedio, por ello se tomaron medidas que ayudaran a la correcta percepción del seguimiento.

La Figura **5-4**, es una imagen tomada de la secuecia de video de las pruebas finales, en ella se pueden observar algunas consolas de la ejecución de los algoritmos. Para tener una vista mejorada de la detección de personas, el resto de las imágenes fueron recortadas al tamaño de la ventana “Detección de personas”.

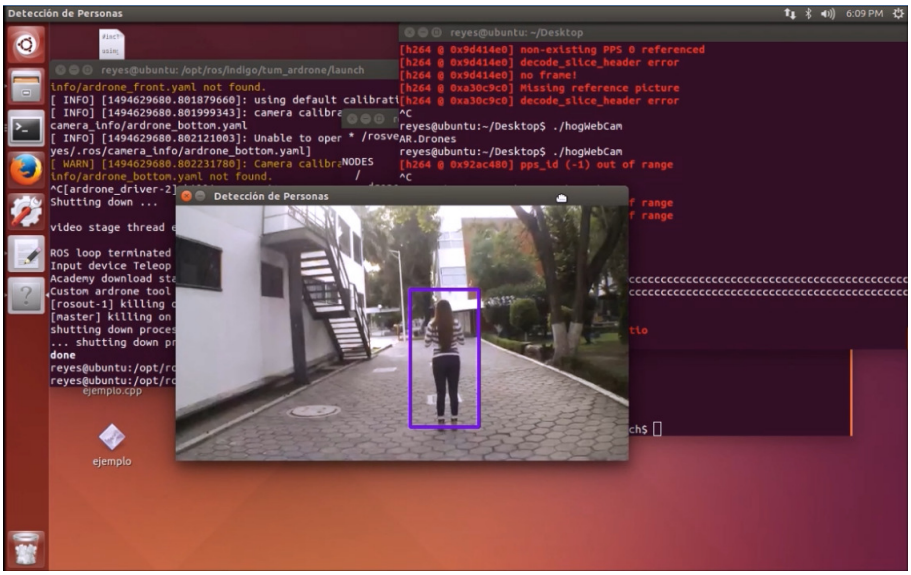
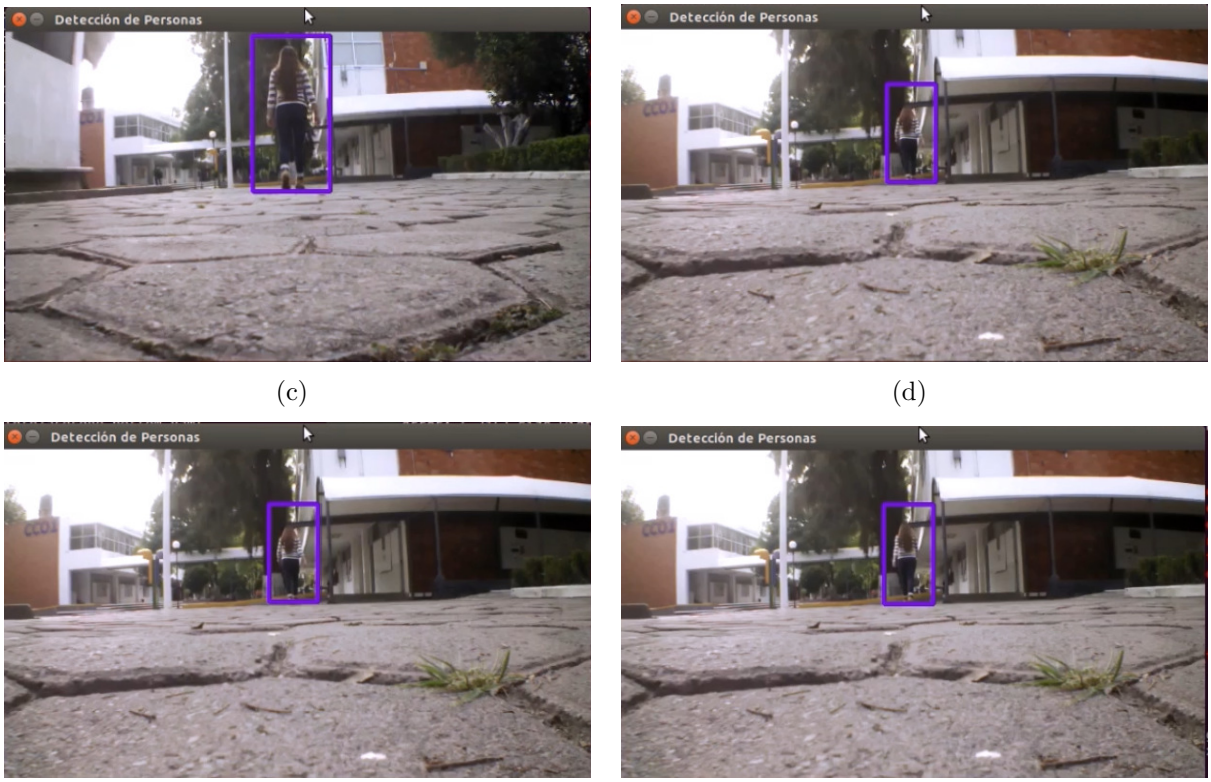


Figura 5-4: Pantalla completa de la ventana de detección y seguimiento.

La Figura **5-5** muestra la ventana de donde se está ejecutando el algoritmo de seguimiento, es decir, se observa lo que está grabando la cámara del AR.Drone y cómo se está aplicando el algoritmo, a la vez que se realiza un vuelo autónomo.

(a) (b)



(c) (d)

(e) (f)

Figura 5-5: Seguimiento de una persona con el AR.Drone 2.0.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Se realizó un sistema cuyo objetivo principal es el desarrollo de estrategias para el seguimiento de personas con vehículos aéreos no tripulados, dicho sistema lleva implícito diferentes técnicas como la extracción de características basada en Histogramas de Gradientes Orientados, la clasificación basada en Máquinas de Soporte Vectorial y el seguimiento utilizando el filtro de partículas.

Se desarrollaron diferentes técnicas para mejorar el funcionamiento de cada algoritmo y se probaron los algoritmos implementados, con lo que se demostró que el Histograma de Gradientes Orientados hace un reconocimiento muy bueno de varias siluetas de personas en la escena, sin embargo es importante tomar en cuenta las oclusiones y condiciones de iluminación de los escenarios, ya que se pudo apreciar que fueron estas condiciones las que disminuyeron el rendimieto neto del algortimo. Además de tomar en cuenta la distancia a la que se encuentra la persona del rango de visión de la cámara, si cada una de estas condiciones se cumple para la ejecución de la estrategia de detección de personas, los resultados son indudablemente eficaces.

El entrenamiento que se realizó a la MSV utilizando los datasets creados tuvieron un excelente rendimiento tomando en cuenta que se realizaron diferentes pruebas hasta obtener los mejores valores de salida, cabe resaltar en este punto que con base en las pruebas que se realizaron se pudo observar que los mejores resultados en los valores positivos, negativos, falsos positivos y falsos negativos se presentaron con un conjunto de 7025 imágenes negativas y 4218 imágenes negativas, es decir, que la razón de tamaño de los conjuntos muestra para el entrenamiento funcionan mejor a razón de 100 % ejemplos negativos y 50 % positivos.

El filtro de partículas presentó un seguimiento eficiente de las personas con las que se desarrollaron las pruebas, éste fue un algoritmo que presentó buenos resultados desde el principio de su desarrollo, sin embargo hay aspectos como la cantidad de partículas utilizadas y las condiciones de luz que podrían mejorarse.

Se mostró el desempeño del sistema integrado a un planificador de vuelos basado en algoritmos RRT y se demostró que se utilizaron soluciones robustas. Es importante señalar que las órdenes de movimiento que realiza el MAV aún son un tanto complejas, debido a los factores

externos muy variables que inherentemente se encuentran en este tipo de vehículos, como lo es el viento.
Sin embargo, las estrategias sugeridas en este ámbito ha sido exitosas debido a que se ha logrado implantar un comportamiento inteligente en el vehículo que le permite valerse por sí mismo en la tarea de seguimiento de una persona.

6.2. Trabajo futuro

Tras haber realizado este trabajo de tesis, se proponen varias líneas futuras de desarrollo para la continuación del proyecto.

- Combinación de técnicas de procesamiento de imágenes. En este trabajo, la imagen se analizó únicamente utilizando los descriptores HOG. No obstante, es probable que combinando los descriptores HOG con alguna otra técnica de procesado de imagen se consigan mejores resultados.

- Realizar nuevos entrenamientos más completos. Con un entrenamiento más completo el sistema podría reducir su tasa de error e incluso poder reducir al mínimo los valores falsos positivos y falsos negativos, valdrá la pena desarrollar una gran cantidad de pruebas para poder establer porcentajes de relación entre los tamaños de los conjuntos muestra positivos y negativos. Cotejar los resultados obtenidos con los ficheros de "ground truth"que proporcionan los datasets y comparar con otros proyectos.

- Existe un amplio trabajo a futuro puesto que, por ejemplo, se puede utilizar un algoritmo RRT en tiempo real que permita generar nuevas trayectorias dependiendo de los obstáculos imprevistos que se puedan presentar, o bien mejorar la estabilidad del vuelo en corrientes fuertes de aire, de igual forma.

- En el seguimiento de personas es posible hacer un mejor entrenamiento para la obtención de características dónde se tomen factores claves como, diferentes tipos de iluminación, escenarios, tipos de segmentación, tamaño de los datasets, entre otros, además de mejorar las características del filtro de partículas para un rendimiento más eficiente del seguimiento.

- Uno de los aspectos más significativos que saltan a la vista es la dificultad de estabilidad en los movimientos del MAV, sobre todo en condiciones de viento fuerte. La plataforma permite al vehículo permanecer en su posición cuando no tiene órdenes de moverse, sin embargo, es necesario implementar un ciclo de control que al presentar cambios en su posición en la tercera dimensión por agentes externos, logre regresar a la posición que se encuentra, tomando provecho de los sensores con los que se cuentan como la cámara que apunta al piso y el sensor ultrasónico.

- Migrar a tecnologías más eficientes. Debido a que, como se ha visto, el proceso de extracción de descriptores HOG a diferentes escalas es fácilmente paralelizable, se considera que la utilización de la GPU para la ejecución en paralelo de esa etapa puede reducir de forma notable el tiempo de procesamiento, pudiéndose eficientar los tiempos de ejecución en el proceso de detección y seguimiento.

Por las razones anteriores se busca mejorar las estrategias propuestas en este trabajo. Para el seguimiento en condiciones más extremas, es necesaria la intervención sobre el algoritmo desarrollado, proponiendo mejoras en la continuidad del proyecto. Una de las tareas en la que conviene trabajar fuertemente es en el seguimiento del objeto cuando se encuentran muchas personas (siluetas) u objetos del mismo tipo en la escena, sin perder de vista el objetivo principal, además de centrarse en la variante de iluminación que resulta de mucho interés mejorar para contar con un sistema eficiente lo más cercano al 100 %.

Finalmente, sería interesante aplicar las estratégias desarrolladas en vehículos aéreos no tripulados de diferente escala que cuenten con las características de sensores similares a las del AR.Drone 2.0 y con ello implementar aplicaciones en diversos campos, como el científico o el del entretenimiento.

Bibliografía

[1] G. Bradski y Adrian Kaehler. Learning opencv: Computer vision with the opencv library. Technical report, O'Reilly Media, septiembre 2008.

[2] Bebop drone. `http://global.parrot.com/mx/productos/bebop-drone/`, Último acceso: Mayo 2017.

[3] Escolano F, Cazorla M, Alonso M, Colomina O, and Lozano M. *Inteligencia artificial - Modelos, técnicas y áreas de aplicación.* Paraninfo, 2003.

[4] Conlago G., Yunda Sangoluisa Cristian R., and Jhony A. *Sistema Automático de Detección y Reconociemiento de señales de tránsito en Intersecciones Viales para Aplicaciones en Vehículos Inteligentes.* Tesis de licenciatura, Universidad de las Fuerzas Armadas, 2016.

[5] Histogram of oriented gradients (hog) descriptor. `https://software.intel.com/en-us/node/529070`, Último acceso: Abril 2017.

[6] Detector basado en hog/svm. `https://es.coursera.org/learn/deteccion-objetos`, Último acceso: Abril 2017.

[7] Enrique J. Carmona Suárez. Tutorial sobre máquinas de vectores soporte (svm). Technical report, Dpto. de Inteligencia Artificial, ETS de Ingeniería Informática, Universidad Nacional de Educación a Distancia (UNED), C/Juan del Rosal, 16, 28040-Madrid (Spain), Julio 2014.

[8] Parrot. Ar.drone developer guide sdk 2.0. Technical report, 2012.

[9] D.A. Ballard and C.M. Brown. *Computer vision.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1982.

[10] Zhao Pietikäinen, Hadid and Ahonen T. *Computer vision Using Local Binary Patterns.* Prentice-Hall, Springer London Dordrecht Heidelberg New York, 2011.

[11] Ayax Aldemar García López. *Reconocimiento de Objetos Inmersos en Imágenes Estáticas mediante el Algoritmo HOG y RNA-MLP.* Tesis de maestría, Universidad Tecnológica de la Mixteca, Huajuapan de León, Oaxaca, Agosto, 2015.

[12] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. *International Conference on Computer Vision and Pattern Recognition*, 2(3 - 4):886 – 893, June 2005.

[13] Sahama Tony Jung Hoe-Kyung, Tae Kim Jung and Yang Chung-Huang (Eds.), editors. *Future Information Communication Technology and Applications: ICFICE 2013*, volume 1. Springer, Germany Berlin Heidelberg, 2013. Chapter 16, Tracking Multi-Objects in Web Camera Video Using Particle Filtering, Yang Weon Lee.

[14] González Marcos Ana... [et al.] (integrantes del Grupo de Investigación EDMANS). Técnicas y algoritmos básicos de visión artificial. Technical report, Universidad de La Rioja, Servicio de Publicaciones, 2006.

[15] Jean Serra Jesús Angulo. Segmentación de imágenes en color utilizando histogramas bivariables en espacios color polares luminancia/saturación/matiz. *Revista Computación y Sistemas*, 8(4), June 2005.

[16] Escamilla Nú nez Rafael. *Diseño, Construcción, Instrumentación y Control de un Vehículo Aéreo No Tripulado (UAV)*. Tesis de licenciatura, Instituto Politécnico Nacional, México, DF, Noviembre, 2010.

[17] Svm^{Light}. `http://svmlight.joachims.org/`, Último acceso: Abril 2017.

[18] Trainhog tutorial. `https://github.com/DaHoC/trainHOG/wiki/trainHOG-Tutorial`, Último acceso: Abril 2017.

[19] Mit pedestrian data. `http://cbcl.mit.edu/software-datasets/PedestrianData.html`, Último acceso: Abril 2017.

[20] Inria person dataset. `http://pascal.inrialpes.fr/data/human/`, Último acceso: Abril 2017.

[21] Peta dataset. `http://mmlab.ie.cuhk.edu.hk/projects/PETA.html`, Último acceso: Abril 2017.

[22] The oxford buildings dataset. `http://www.robots.ox.ac.uk/%7Evgg/data/oxbuildings/index.html`, Último acceso: Abril 2017.

[23] Ardrone open api platform. `http://developer.parrot.com/`, Último acceso: Mayo 2017.

[24] A basic c library for interfacing the ar.drone quadricopter. `https://github.com/IUT-Beziers/RT_ARDrone`, Último acceso: Mayo 2017.

[25] Ros.org — powering the world´s robots. `http://www.ros.org/about-ros/`, Último acceso: Mayo 2017.

[26] Ar drone open api platform. `//projects.ardrone.org/.`, Último acceso: Mayo 2017.

[27] Ar drone - parrot for developers. `http://developer.parrot.com/ar-drone.html`, Último acceso: Mayo 2017.

[28] Autonomy lab. `http://autonomylab.org`, Último acceso: Mayo 2017.

[29] Sfu.ca - simon fraser university. `http://www.sfu.ca/`, Último acceso: Mayo 2017.

[30] ardrone_autonomy ardrone_autonomy indigo-devel documentation. `http://ardrone-autonomy.readthedocs.org/`, Último acceso: Mayo 2017.

[31] A.A. Reyes Montero. *Planificación de movimientos para robots aéreos no tripulados.* Tesis de licenciatura, Benemérita Universidad Autónoma de Puebla, 2017.