



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

Detección y transporte inteligente de objetos utilizando el RC Rover®

Por:
Glebys González

Realizado con la asesoría de:
Carolina Chang

PROYECTO DE GRADO
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero en Computación

Sartenejas, Septiembre de 2012



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

ACTA FINAL PROYECTO DE GRADO

Detección y transporte inteligente de objetos utilizando el RC Rover®

Presentado por:
Glebys González

Este Proyecto de Grado ha sido aprobado por el siguiente jurado examinador:

Carolina Chang

Carolina Martínez

José Cappelletto

Sartenejas, (17/09/12)

Resumen

Este trabajo plantea el desarrollo de un sistema que integre el control de un robot, la captura y procesamiento de su video y la integración de sensores adicionales. El sistema es aplicado al problema de recolección de objetos.

El desarrollo de este proyecto se puede dividir en dos partes. La primera es la creación de un conjunto de librerías que permitieran el controlar de manera sencilla el robot RC Rover y facilitaran el uso de sensores adicionales.

Adicionalmente se requería hacer la captura del video enviado por el robot. Para realizar esta tarea se utilizó la librería de visión de computadoras OpenCV.

La segunda parte del proyecto consiste en el seguimiento de objetos. Con la intención inicial de que el robot siguiera un objeto genérico, se desarrollo un sistema con flujo óptico que le permitiera al usuario señalar cualquier elemento, para que el robot procediera a buscarlo.

Posteriormente este sistema se combinó con el de aprendizaje de máquinas para reconocer botellas automáticamente. Para lograr este objetivo se utilizaron redes neurales. Se obtuvo un 89 % de clasificación correcta.

Finalmente se logró producir un sistema que integrara todos los elementos mencionados anteriormente. Se posee un flujo óptico que es resistente hasta el 60 % de la interferencia que se produce de manera constante en la transmisión de la señal. Al combinar el flujo óptico con la detección de botellas el área de búsqueda para la red neural se puede reducirse hasta un 88 %, mejorando considerablemente el desempeño del algoritmo.

Adicionalmente se plantea un sistema de recolección semi-automática de datos de entrenamiento para la red neural, que reduce notablemente los tiempos de obtención de los mismos.

Palabras clave: Robótica inteligente, Visión de Computadoras, Aprendizaje de Máquinas, Flujo Óptico, Seguimiento de Objetos.

Al Grupo de Inteligencia Artificial,
por convertirse en más que un segundo hogar.

Agradecimientos

Durante la realización de este proyecto, conté con el fuerte apoyo de personas y entes, cuyos esfuerzos son parte importante de este trabajo. La menciones a realizar a continuación no están planteadas en orden de importancia.

En primer lugar, quiero agradecer a la Profesora Carolina Chang, por guiarme, mantenerse firme y apoyarme más de lo que se pensaría posible, a lo largo de todo el desarrollo del proyecto.

En segundo lugar quiero agradecer a la Universidad Simón Bolívar, por ser el motor de un aprendizaje verdaderamente integral, que me acompañará en las siguientes etapas de mi vida.

Agradezco también a Kelwin Fernández, que sentado a mi lado en el desarrollo de este proyecto, dió excelentes ideas que ahora se ven reflejadas a lo largo de todo este trabajo.

Además, quisiera agradecer a María Leonor Pacheco, por convertirse en una ayuda crucial en la etapa final de este proyecto.

Por último, agradezco a mi familia, por brindarme su apoyo y soporte durante toda mi vida académica.

Índice general

Índice general	vii
Índice de cuadros	x
Índice de figuras	xi
1. Introducción	1
2. Marco Teórico	5
2.1. Robótica Inteligente	5
2.1.1. Paradigmas de la Robótica Inteligente	6
2.1.1.1. Paradigma Jerárquico	6
2.1.1.2. Paradigma Reactivo	7
2.1.1.3. Paradigma Híbrido	8
2.1.2. Teleoperación	8
2.2. Visión de Computadoras	9
2.2.1. Filtrado de Imágenes	9
2.2.1.1. Difuminado	10
2.2.1.2. Ecualización de Histogramas	11
2.2.1.3. Detección de Bordes	12
2.2.2. Flujo Óptico	14
2.3. Conexiones Seriales	18
2.3.1. RS-232	18
2.3.2. Transmisión de la Señal	19
2.3.3. UART o LLT	19
2.4. Aprendizaje de Máquinas	19

2.4.1. Aprendizaje Supervisado	20
2.4.2. Redes Neurales Artificiales	21
2.4.2.1. Neurona Artificial	21
2.4.2.2. Redes Multicapa	21
2.4.2.3. Propagación Hacia Atrás	22
3. Interfaz del Robot RCRover	23
3.1. Captura de Video	25
3.2. Control del Robot	29
3.2.1. Librería de Comunicación Serial para Windows	31
3.2.2. Librería de Comunicación Serial para Unix	32
3.3. Sensores Adicionales	33
4. Seguimiento de Objetos	36
4.1. Lucas-Kanade contra Flujos Densos	37
4.2. Metodología	40
4.2.1. Algoritmo	40
4.2.2. Experimentos para el Flujo Óptico	43
4.3. Filtrado	46
4.3.1. Filtros utilizados	46
4.3.2. Experimentos con filtros	47
4.4. Detección de Interferencias	49
4.5. Detección de Botellas con Redes Neurales	51
4.5.1. Sistema Semi-Automático de Recolección de Datos	51
4.5.2. Entrenamiento y Resultados	51
4.5.3. Redes Neurales y Flujo Óptico	53
5. Conclusiones y recomendaciones	54
Bibliografía	57
A. Detalles de la Clasificación de Píxeles Borde	61
B. Características de la Comunicación Serial	63
C. Detalles Técnicos del RCRover 9-18	65

D. Equipos de Captura de video Revisados	67
E. Librerías de Captura Revisadas	71
F. Documentacion de la Librería RoverSerial	74

Índice de cuadros

3.1.	Componentes básicos del Robot RCRover	24
3.2.	Funcionalidades ofrecidas por <i>OpenCV1</i> , <i>OpenCV2</i> y <i>Processing</i> en 32bits para un Sistema en 64bits	28
3.3.	Error del ultrasonido	35
4.1.	Porcentaje de píxeles escogidos del fondo	40
4.2.	Promedio de veces que se pierde el flujo óptico debido a la interferencia o a debilidades propias del algoritmo, en un recorrido de cuatro metros, variando el valor de α en $\alpha * \sigma$	45
4.3.	Promedio de veces que se pierde el flujo óptico debido a la interferencia o a debilidades propias del algoritmo, en un recorrido de cuatro metros, variando la cantidad de píxeles en el flujo.	45
4.4.	TMI y PFO de los distintos filtros utilizados	48
4.5.	Porcentaje de clasificación correcta de las redes neurales. NN : Número de Neuronas. SF : imagen Sin Filtro. M : imagen con filtro de Mediana, G : imagen con filtro Gaussiano. EQ+M : Ecualización de Histogramas y filtro de Mediana. EQ+G : Ecualización de Histogramas y filtro Gaussiano Todos los filtros utilizados tienen ventana de tamaño tres.	52
B.1.	Tipos de Bits estándar en el envío de mensajes seriales	63
B.2.	Pines del estándar RS-232	64
C.1.	Detalles Técnicos del RCRover 9-18	66

Índice de figuras

2.1.	Paradigmas de la robótica inteligente	7
2.2.	Desventajas del paradigma reactivo	8
2.3.	Filtro Gaussiano y de Mediana	11
2.4.	Ecualización de Histogramas	13
2.5.	Flujo Óptico	14
2.6.	Lucas-Kanade piramidal	17
3.1.	Kit del Robot modelo REV9-18	23
3.2.	Hardware utilizado en la recepción de video	26
3.3.	Hardware utilizado en la recepción de video	27
3.4.	Hardware utilizado en la recepción de video	29
3.5.	Paquete de datos seriales	30
3.6.	Algoritmo de comunicación serial	31
3.7.	Pseudocódigo de recepción para el Arduino	34
4.1.	Interfaz Robot-Humano	37
4.2.	Píxeles en el Flujo Óptico	38
4.3.	Píxeles en el Flujo Óptico	39
4.4.	Método de Seguimiento del Objeto	42
4.5.	Posiciones del Robot en los Experimentos de Flujo	44
4.6.	Tipos de interferencia	50
A.1.	Clasificación de Píxeles	62

Capítulo 1

Introducción

Con el paso del tiempo, la robótica desempeña un papel más importante en la sociedad. Desde la industria, hasta la asistencia humana, cada vez existen más áreas en las que la robótica está tomando terreno.

Los robots están pensados para realizar trabajos que cumplan con alguna de las tres *D* (proveniente del inglés): *Dull, Dirty and Dangerous* [Mur00]. Esto se traduce en actividades que sean sucias, repetitivas, agotadoras o peligrosas. En el caso de que éstas requirieran de toma de decisiones, se utilizaban en su gran mayoría equipos teleoperados, delegando el control del robot a un humano.

Con el desarrollo de la inteligencia artificial, se han generado avances en la robótica inteligente [Mur00]. Los robots autónomos significan un alivio para muchas tareas humanas, donde debe haber una persona detrás de cámaras tomando las decisiones por el robot.

Existe un desarrollo importante en el área de los robots autónomos. Un ejemplo claro de esto, son los robots exploradores de la Nasa, que deben lidiar con terreno, ambiente y obstáculos desconocidos fuera de la tierra, sin posibilidades de recibir por radio instrucciones de los humanos [N1111]. Se piensa que en el futuro, la cantidad de tareas que serán desempeñadas por robots autónomos será cada vez mayor [Mur00].

La capacidad de un robot para hacer un trabajo está inicialmente restringida por su hardware. Por lo tanto, para entender el posible alcance de este proyecto, se debe comenzar describiendo físicamente el equipo de trabajo.

El modelo utilizado en el presente trabajo fue un RCRover9-18 [RC11]. Éste consiste en un robot todo terreno que cuenta con un sistema completo de teleoperación: un control remoto y un equipo de monitoreo a distancia, que permite recibir la señal de la imagen captada por la cámara del robot. El RCRover9-18 cuenta adicionalmente con una garra, que está colocada al final de un brazo con un grado de libertad.

Se ha probado que la teleoperación resulta una tarea difícil para el humano [CM07]. Esto se debe a que la percepción del espacio en el que trabaja el robot es limitada para su usuario. En particular, cuando se posee una sola cámara, que es el caso de este proyecto, se reduce drásticamente la capacidad de percepción de la profundidad. Empíricamente se determinó que esto dificulta principalmente dos tareas al manipular el RC Rover: detección y evasión de obstáculos y recolección de objetos.

Al percibir de manera incorrecta la distancia con los elementos del ambiente, se hace muy complicada manipulación un objeto con la garra. Esto se debe a que no se puede saber con precisión si el elemento se encuentra dentro del rango de agarre del robot.

Tomando en cuenta lo anterior, se propone la recolección de objetos como una posible aplicación de interés para el RCRover. Esta tarea puede ser clasificada como repetitiva o peligrosa, cuando se trabaja en un área contaminada, de difícil acceso o alto riesgo. La actividad también puede ser clasificada como peligrosa si el objeto presenta características que sean nocivas para el ser humano.

En particular, la recolección de basura reciclabla es una tarea de alta necesidad en costas y áreas de desperdicios contaminadas. En la actualidad, la cantidad basura generada no se equipara con el poder de reciclaje. Los problemas en esta área son un buen terreno para el

desarrollo de sistemas robóticos de recolección [Sal].

Este trabajo plantea la creación de un sistema que permita la recolección autónoma y asistida de objetos a través de un robot. Para esto se cuenta con un equipo que posee una única cámara como medio de percepción del ambiente. Se espera que este sistema sea adaptable a otros robots de características similares. Existen trabajos previos en el área de reconocimiento y recolección de objetos, desempeñados por un equipo robótico. Es usual que el proceso requerido para que el robot reconozca o aprenda a recoger los objetos deseados tiene a consumir una gran cantidad de tiempo y los resultados no siempre presentan buena precisión [FPD08, Mel11, SY08].

En [FPD08], se busca complementar los datos de entrenamiento con un conjunto de características basadas en los ángulos y las esquinas de los objetos. Utilizando esta información, el proceso de reconocimiento se hace mucho más exacto. Este proyecto tiene un requerimiento que representa una gran desventaja; el piso debe estar cuadriculado para poder generar las referencias de los ángulos.

Es muy común que en los trabajos de robótica el ambiente sea controlado y requiera de características especiales, como medidas predeterminadas, presencia de marcas o señalización en colores. En el caso de este desarrollo, se busca que el robot pueda trabajar en ambientes variados sin condiciones particulares.

En [Mel11], se trabaja con imágenes de los objetos tomadas desde ángulos conocidos. Aplicado al ambiente, el mismo concepto sirve para la navegación. El equipo que se posee en la Universidad no cuenta una cámara y sensores que tengan las características necesarias para su desarrollo.

Un trabajo innovador es [SY08]. Éste utiliza visión estereoscópica para reconocer las características volumétricas en virtualmente cualquier objeto, sin necesidad de poseer un modelo 3D previo. El robot analiza la estructura del objeto y decide cómo agarrarlo. El RC

Rover no cuenta con visión estereoscópica. El proceso de añadir una segunda cámara para permitirle esa capacidad se excede de las posibilidades del laboratorio.

Hasta el momento no existen trabajos publicados que permitan aplicar algoritmos de visión de computadoras o control autónomo al RC Rover R8-19. Aunque existen mejores equipos robóticos en el mercado para el desarrollo del proyecto, las restricciones de adquisición del laboratorio impiden su compra.

El objetivo general de este proyecto es crear un sistema central que integre el procesamiento de la imagen captada por el robot, el control del mismo y el uso de sensores extra. Se aplicará este sistema al problema de recolectar objetos. Con el fin de alcanzar el objetivo general de este proyecto, se plantean los siguientes objetivos específicos:

- Realizar captura de video.
- Utilizar técnicas de visión de computadoras para el procesamiento del video.
- Implementar la comunicación bidireccional entre la computadora y el sistema de sensores.
- Crear una librería para el control del robot y uso de los sensores.
- Recolectar objetos mediante el robot.
- Seguir objetos sin información previa.
- Seguir objetos con aprendizaje de máquinas.

El presente trabajo se encuentra dividido en 5 capítulos. El capítulo 1 introduce el problema. En el capítulo 2 se explican los conceptos y nociones bajo las cuales se soporta este desarrollo. El capítulo 3 describe la implementación de la interfaz entre la computadora, el robot y el sistema de sensores. Seguidamente, el capítulo 4 describe el proceso de seguimiento de objetos. Por último, en el capítulo 5 se despliegan las conclusiones y recomendaciones, así como las consideraciones para trabajos futuros.

Capítulo 2

Marco Teórico

En este capítulo se explican los conceptos teóricos que construyen la base del proyecto de investigación. En la sección inicial, se presentan las nociones fundamentales de la robótica inteligente, seguido de los problemas que se presentan cuando el usuario debe operar un robot a distancia. Posteriormente se trata el tema de visión de computadoras, donde se describen los filtros de imágenes necesarios para el pre-procesamiento del video que provee la cámara del robot utilizado. En la misma sección se presentan los conceptos de flujo óptico y sus variaciones. Finalmente se puntualizan las nociones de aprendizaje de máquinas que se usan en la detección de objetos.

2.1. Robótica Inteligente

Para poder hablar de robótica inteligente, primero es necesario definir la inteligencia artificial. Desde la aparición del concepto hasta la actualidad, existe un debate acerca de su significado [RN03]. Una rama busca definir la inteligencia en torno a los procesos cognitivos, mientras que la otra busca fundamentarse en el comportamiento. En ambos casos la meta es

producir máquinas capaces de resolver problemas que sean complejos para el ser humano o generar comportamientos y decisiones que sean considerados correctos.

Un robot inteligente es un agente electromecánico capaz de actuar de manera autónoma [Mur00]. Debe poder moverse en el mundo e interactuar con él, sin depender de la intervención humana.

2.1.1. Paradigmas de la Robótica Inteligente

Actualmente existen tres paradigmas bajo los cuales se diseñan robots inteligentes: jerárquico, reactivo e híbrido [Mur00]. Estos pueden ser descritos fácilmente en base a tres primitivas: *Percibir*, que se refiere a la acción de recolectar la información del ambiente a través de sensores. *Planificar*, que consiste en procesar la información percibida y decidir un conjunto de acciones a tomar. *Actuar* describe a todas aquellas funciones que produzcan movimiento motor en el robot. La figura 2.1 muestra un esquema de cada paradigma, basado en las primitivas.

Bajo este marco conceptual, la primitiva de *percibir* se refiere a la acción de recolectar la información del ambiente a través de sensores. *Planificar*

2.1.1.1. Paradigma Jerárquico

Fue el primero en desarrollarse. Siguiendo los principios de este paradigma, el mundo exterior se percibe a través de los sensores. Seguidamente, los datos obtenidos se usan como parámetros para un modelo global del espacio donde se desenvuelve el robot. En muchos casos el primer paso se omite y se le da al robot directamente un modelo del mundo. Posterior a esto, el sistema de toma de decisiones genera una serie de acciones a ejecutar. El proceso vuelve a repetirse una vez completada la maniobra planificada.

Las desventajas de este modelo se hacen más notables a medida que el número de elemen-

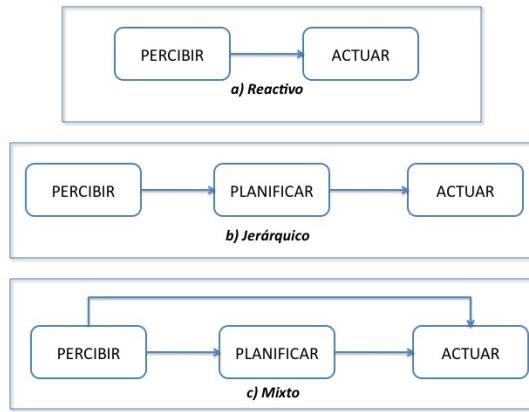


Figura 2.1: Esquema de los tres paradigmas de la robótica inteligente. Cada flujo de acciones se repite de manera cíclica en el robot.

tos y cambios en el ambiente aumenta, ya que el robot no puede ajustarse a sus alrededores una vez tomada la decisión.

2.1.1.2. Paradigma Reactivo

En respuesta al paradigma anterior, surge el modelo reactivo. Este carece de esquema global. Cada acción es una respuesta directa a la percepción de un sensor. En el diseño reactivo, la respuesta de cada dispositivo al exterior es independiente de las otras acciones. No existe un proceso que conozca lo que está ocurriendo en el robot completo. Dentro de sus desventajas existe el riesgo de generar conductas poco inteligentes al no poder llevar un registro de las acciones que se han tomado. La figura 2.2 muestra un ejemplo particular de la situación descrita anteriormente. Por otra parte, es posible que el robot no sea capaz de prever reacciones para el espectro de sensores.

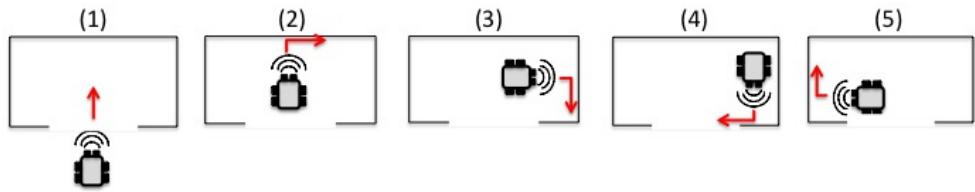


Figura 2.2: Instancia particular de un robot reactivo con un sensor de distancia delantero. Cada vez que encuentra con un obstáculo gira a la derecha. El robot entra en un ciclo sin poder salir de la caja. Sin planificación el agente no sabe que está atrapado.

2.1.1.3. Paradigma Híbrido

Como solución a los problemas derivados de los paradigmas descritos anteriormente, se propuso el modelo híbrido, que combina las características del jerárquico y el reactivo [Mur00]. La meta es utilizar una planificación y modelo global que le permita al robot conocer su estado y generar acciones adecuadas, mientras que las funciones que deben ser atendidas de inmediato (como evitar colisiones) son manejadas de manera reactiva. El planificador central divide las acciones en subtareas, lo que permite el manejo de distintas funciones bajo enfoques diferentes.

2.1.2. Teleoperación

El área de teleoperación es anterior a la robótica inteligente [Mur00]. En este modelo un individuo humano controla los movimientos del robot a través de una interfaz que le permite enviar comandos a distancia de la máquina.

En particular, el uso de la cámara para permitirle al usuario percibir los alrededores del robot, ha sido un elemento presente y constante en esta rama. Sin embargo, la percepción de la profundidad puede verse severamente alterada al interpretar distancias a partir de imágenes en dos dimensiones [CM07]. Por ejemplo, en el caso de tener una garra, la distancia entre ésta y los objetos es fácil de malinterpretar a través de la cámara.

Como solución a este problema se han propuesto distintos modelos que permiten agregarle información extra a la imagen acerca de las distancias y profundidades. La visión estéreo, por ejemplo, ha sido una técnica constantemente estudiada [ML00, CH96, CM95]. Esta consiste en colocar dos cámaras, donde la distancia y ángulo entre ellas es conocido. Sabiendo esta información es posible obtener datos en 3D, comparando el desplazamiento entre las imágenes. Sin embargo, esto sólo es posible cuando el robot posee dos cámaras, lo que coloca en desventaja a máquinas con limitaciones en sus dispositivos. Agregado a esto, la complejidad del procesamiento simultáneo de dos imágenes dificulta en muchos casos el uso en tiempo real.

2.2. Visión de Computadoras

La visión de computadoras es el área de la ciencias en computación que tiene como objetivo la extracción de información en una imagen y la interpretación de la misma, a través de un proceso algorítmico [Sze11, Pri12]. Gran parte de esta rama está ligada a la inteligencia artificial, debido a que el aprendizaje de máquinas es una herramienta importante para reconocer patrones o clasificar elementos.

2.2.1. Filtrado de Imágenes

Usualmente, para extraer información de una imagen, primero se preprocesa [SB08]. Con esta acción se busca eliminar datos que no sean importantes, destacar características de la imagen que se quieran tomar en cuenta y reducir el ruido, entre otros. La tarea de preprocesado se realiza en gran parte utilizando filtros de imágenes. A continuación se presentan los algoritmos de filtrado relevantes a este proyecto.

2.2.1.1. Difuminado

El difuminado es un filtro comúnmente usado para eliminar el ruido. En términos generales, el valor de un píxel se cambia por alguna aproximación al promedio de los vecinos, homogeneizando el valor de la imagen entre píxeles cercanos [SS01]. Como desventaja, el difuminado produce la pérdida de la nitidez de la imagen; sin embargo, conserva las formas generales de los objetos. Existen diversos tipos de filtros para difuminado. En esta sección se explican los dos más comunes.

El primero es el filtro gaussiano. Éste toma un grupo de píxeles y su centro. El valor de cada píxel será menos afectado por el valor del promedio, a medida que esté ubicado más alejado del centro. La siguiente fórmula describe el valor de un píxel:

$$\mathbf{g}(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{d^2}{2\sigma^2}} \quad (2.1)$$

La Ecuación 2.1 tiene la forma de la distribución normal. Donde x y y son las coordenadas del píxel, mientras que σ representa la desviación estándar de la distribución. La variable d es la distancia al píxel central y se calcula de la siguiente forma:

$$d = \sqrt{(x - x_c)^2 + (y - y_c)^2} \quad (2.2)$$

Donde (x_c, y_c) es el píxel que está en el centro del área seleccionada.

El segundo filtro es el de la mediana. Consiste en remplazar al píxel por el valor de la mediana de sus vecinos. La mediana es el valor que se ubica en el medio de una secuencia de valores ordenados. Como no toma los valores de los extremos, tiende a eliminar de manera efectiva los ruidos que contrasten con el valor promedio. Otra característica es que mantiene más definidos los bordes, debido a que no cambia el peso del píxel de manera gradual.

El costo de cómputo del filtro de la mediana es mayor, debido a que para calcular el valor

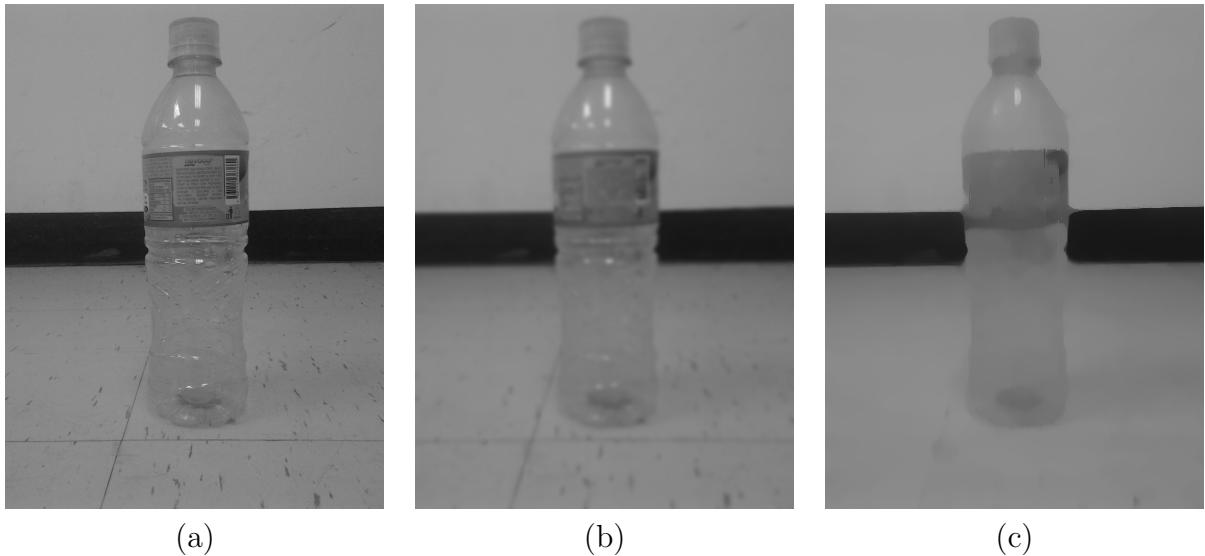


Figura 2.3: (a) Imagen original. (b) Efecto del filtro gaussiano con tamaño de cuadrado (kernel) de 11x11píxeles y $\sigma = 11$. (c) Filtro de mediana con kernel de 11x11píxeles

de un píxel necesita organizar secciones (parciales) de la matriz que representa la imagen. La figura 2.3 muestra el resultado de ambos filtros sobre la imagen de una botella.

2.2.1.2. Ecualización de Histogramas

La ecualización de histogramas es una técnica que ajusta los niveles de intensidad en los píxeles, con la meta de aumentar el contraste en una imagen [GW08]. Este método cambia el peso del píxel según la probabilidad acumulada (desde el valor cero hasta el valor en ese punto), de que algún píxel en la imagen posea la misma intensidad. La probabilidad se representa de la manera clásica; casos exitosos sobre casos totales, como se muestra en la Ecuación 2.3:

$$P_n = \frac{\text{Número de píxeles con intensidad } n}{\text{Número total de píxeles}} \quad (2.3)$$

Usualmente los valores de n varían entre 0 y 255. La ecualización de histogramas funciona como una transformada, donde cada píxel de intensidad i , toma el valor descrito por la función

T .

$$\mathbf{T}(i) = \lfloor (S - 1) \sum_{n=0}^i P_n \rfloor \quad (2.4)$$

En la Ecuación 2.4, la variable S representa el número total de posibles valores i , que un píxel puede tomar. Otra manera de interpretar la ecualización de histogramas, es tomando a las intensidades de los píxeles en la imagen original y ecualizada, como variables aleatorias X y Y continuas, respectivamente. Utilizando este enfoque, $\mathbf{g}(X)$ es la función de distribución acumulada de X multiplicada por $S - 1$, como se muestra en la Ecuación 2.5.

$$Y = \mathbf{T}(X) = (S - 1) \int_0^X P_x dx. \quad (2.5)$$

Esta técnica puede generar mejoras importantes en la calidad de la imagen. Sin embargo, en muchas instancias, puede aumentar la intensidad de datos no significativos y reducir la misma en los puntos de interés. La figura 2.4, muestra el resultado de la aplicación de ecualización de histogramas sobre la imagen utilizada anteriormente.

2.2.1.3. Detección de Bordes

Consiste en demarcar los límites de las figuras en una imagen, de modo que sólo se observen los bordes de los objetos. El algoritmo más utilizado para esta detección es Canny [Can86], llamado así por el nombre de su creador. A continuación se explican los pasos a seguir al aplicar este método.

Inicialmente, la imagen se preprocesa aplicando un filtro que la difumine, con el fin de eliminar el ruido y los bordes de los detalles que no son relevantes. Usualmente se utiliza el difuminado gaussiano.

Seguidamente, se obtienen los gradientes de la imagen en el eje x y y . Existen varios métodos para realizar este cálculo, entre ellos están: Roberts, Prewitt y Sobel [MA09]. A

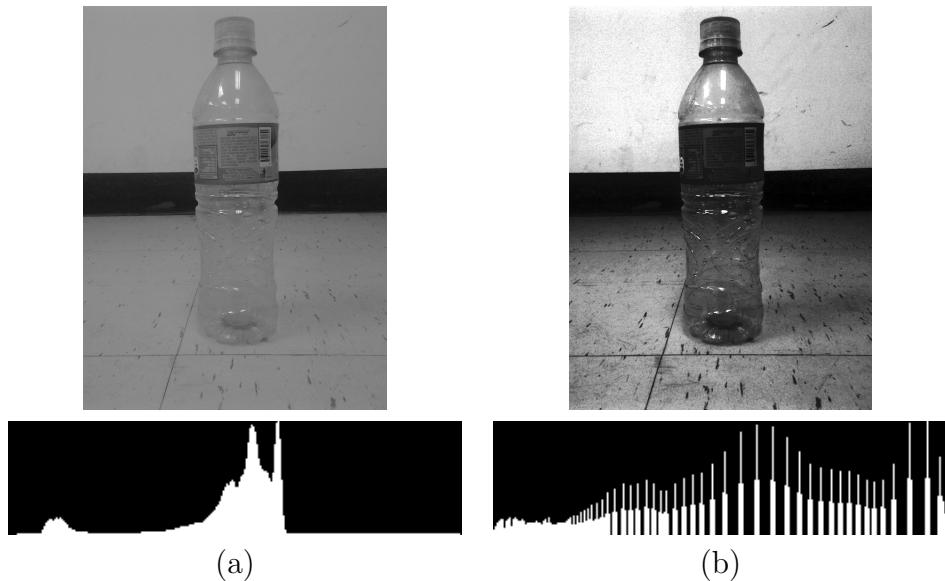


Figura 2.4: (a) Imagen original y su histograma. Nótese que las intensidades de los píxeles están distribuidas hacia el centro de manera poco uniforme (b) Resultado de aplicar el filtro de ecualización. Los valores de los píxeles ocupan un rango mucho mayor debido al aumento de contraste.

través de los gradientes se pueden obtener las direcciones θ de todos los posibles bordes:

Posteriormente, los ángulos se clasifican dentro de alguna de las 4 direcciones principales; vertical, horizontal y diagonales, de modo que al final de este proceso, cada valor de gradiente termina asociado a uno de sólo cuatro tipos de orientación.

Después de realizado el paso anterior, se procede a clasificar cada píxel como posible borde o no. Esto se hace comparando el gradiente en cada punto, contra el valor en sus puntos adyacentes. Los vecinos utilizados en la comparación son únicamente aquellos que estén clasificados en direcciones opuestas. Por ejemplo, si un punto es clasificado como vertical, los adyacentes utilizados deberán tener posición horizontal. Si el píxel en cuestión tiene un valor de gradiente mayor al de sus vecinos, es clasificado como posible borde.

Una vez obtenidos los candidatos iniciales, se comparan contra dos valores de umbral, uno alto y uno bajo. Primero se escogen los gradientes que superen el valor del umbral alto. Luego, se eligen los píxeles candidatos que queden cerca de las líneas que se dibujan entre

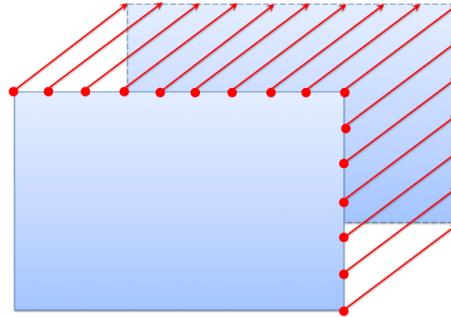


Figura 2.5: Cálculo del flujo óptico para los píxeles en los bordes superior y derecho de un rectángulo.

los puntos escogidos en la primera ronda. En esta segunda vuelta sólo se escogen píxeles cuyo gradiente sea mayor al umbral bajo. El Apéndice A describe en mayor detalle este proceso.

2.2.2. Flujo Óptico

El flujo óptico puede definirse como un campo vectorial que describe cómo una imagen cambia con el tiempo [Smi97]. La figura 2.5 muestra el cálculo del flujo óptico para dos bordes de un rectángulo.

Este cálculo está basado en la idea de que la intensidad de un píxel se mantiene a través del tiempo [FW09]. Si el píxel cambia de posición, debido al movimiento, puede determinarse su desplazamiento, al ubicarlo en la siguiente imagen. Si bien esta propiedad no se da de manera perfecta o exacta en la realidad, se puede generar una buena estimación del flujo óptico basándose en esas nociones.

Para poder explicar fácilmente el funcionamiento de los algoritmos, se debe definir previamente una notación. Sea p un píxel de coordenadas $p.x$ y $p.y$. Sea $\mathbf{I}(p, t)$, la intensidad del punto p en el momento t . Definamos v como el vector de velocidad $\in \mathbb{R}^2$ del píxel p .

La mayoría de los métodos para calcular el flujo óptico parten del mismo proceso de estimación de la velocidad v basado en el gradiente.

La estimación del flujo comienza suponiendo que se cumple la propiedad de la conservación de la intensidad de un píxel, que puede definirse de la siguiente manera:

$$I(p.x, p.y, t) = \mathbf{I}(p.x + v.x, p.y + v.y, t + 1) \quad (2.6)$$

Definamos Δx , Δy como el cambio producido por v en el píxel p entre dos imágenes. La propiedad de la conservación de la intensidad se redefiniría como:

$$\mathbf{I}(p.x, p.y, t) = \mathbf{I}(p.x + \Delta x, p.y + \Delta y, \Delta t) \quad (2.7)$$

Suponiendo que el movimiento es pequeño, el desplazamiento puede aproximarse con series de Taylor de primer orden:

$$\mathbf{I}(p.x, p.y, t) \approx \mathbf{I}(p.x, p.y, t) + \frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + \frac{\partial I}{\partial t} \Delta t \quad (2.8)$$

De esta aproximación podemos obtener la siguiente expresión:

$$\frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + \frac{\partial I}{\partial t} \Delta t = 0 \quad (2.9)$$

Multiplicando en ambos lados por el factor $\frac{1}{\Delta t}$, se obtiene la ecuación con las velocidades que define el flujo óptico:

$$\frac{\partial I}{\partial x} \frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y} \frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t} \frac{\Delta t}{\Delta t} = \frac{\partial I}{\partial x} v.x + \frac{\partial I}{\partial y} v.y + \frac{\partial I}{\partial t} = 0 \quad (2.10)$$

Finalmente, se tiene una ecuación con dos incógnitas. Este es el conocido problema de apertura del flujo óptico [FW09]. Los métodos que estiman el flujo buscar agregar una segunda ecuación que permita resolverlo. En el caso de este desarrollo se trabajó con el método Lucas-Kanade [LK81].

Para agregar más ecuaciones, el algoritmo Lucas-Kanade añade una nueva restricción. Si se tiene una ventana con n píxeles q_i , $i \in [1 \dots n]$, con centro en el píxel p , posible suponer que en un período de tiempo corto, el desplazamiento para todos los puntos de la ventana debe ser el mismo. Esta restricción genera el siguiente sistema de ecuaciones para $v.x$ y $v.y$:

$$\begin{aligned} F_x(q_1)v.x + F_y(q_1)v.y + F_t(q_1) &= 0 \\ F_x(q_2)v.x + F_y(q_2)v.y + F_t(q_2) &= 0 \\ &\vdots \\ F_x(q_n)v.x + F_y(q_n)v.y + F_t(q_n) &= 0 \end{aligned} \tag{2.11}$$

Donde F_x , F_y y F_t son las derivadas parciales de la imagen $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ y $\frac{\partial I}{\partial t}$ respectivamente.

Este sistema puede reorganizarse para tener la forma clásica $Av = b$, donde:

$$A = \begin{bmatrix} F_x(q_1) & F_y(q_1) \\ F_x(q_2) & F_y(q_2) \\ \vdots & \vdots \\ F_x(q_n) & F_y(q_n) \end{bmatrix}, v = \begin{bmatrix} v.x \\ v.y \end{bmatrix}, b = \begin{bmatrix} -F_t(q_1) \\ -F_t(q_2) \\ \vdots \\ -F_t(q_n) \end{bmatrix} \tag{2.12}$$

La solución a este sistema se obtiene a través del método de los mínimos cuadrados:

$$v = (A^T A)^{-1} A^T b \tag{2.13}$$

Al desarrollar la expresión anterior se obtiene:

$$\begin{bmatrix} v.x \\ v.y \end{bmatrix} = \begin{bmatrix} \sum_i F_x(q_i)^2 & \sum_i F_x(q_i)F_y(q_i) \\ \sum_i F_x(q_i)F_y(q_i) & \sum_i F_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i F_x(q_i)F_t(q_i) \\ -\sum_i F_y(q_i)F_t(q_i) \end{bmatrix} \tag{2.14}$$

Este sistema de ecuaciones permite determinar la velocidad con la que el píxel p se des-

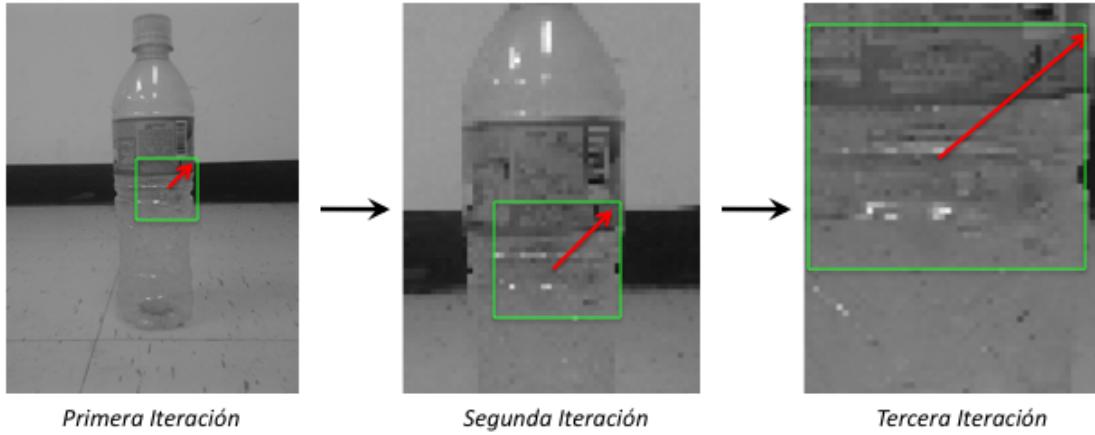


Figura 2.6: Implementación de tres iteraciones del algoritmo Lucas-Kanade piramidal. En verde se señala la ventana en cada iteración. El vector en rojo representa el desplazamiento calculado para el píxel central de la ventana.

plazó entre dos imágenes. En el caso de que el desplazamiento sea muy grande, el algoritmo original produce errores. Para atacar este problema, existe una implementación iterativa piramidal [Bou00].

El Lucas-Kanade piramidal comienza con la ventana de píxeles inicial. En cada iteración duplica su tamaño, a través de extrapolación bilineal, e intenta estimar el desplazamiento de la nueva ventana utilizando la velocidad calculada en el nivel anterior.

La figura 2.6, muestra el funcionamiento del algoritmo en una instancia con tres iteraciones. El rectángulo verde indica la ventana seleccionada para calcular el flujo. El vector rojo indica el cálculo del desplazamiento del píxel central. En cada iteración se amplía la imagen, generando una ventana más grande, que se utiliza para recalcular el desplazamiento. Al aplicar este algoritmo no se amplia la imagen completa, sólo el recuadro de interés. En la figura se muestra la ampliación completa por motivos de legibilidad. En el resto de este documento, cada vez que haga referencia al método Lucas-Kanade, se estará hablando de su implementación piramidal.

Otra mejora importante desarrollada para el método Lucas-Kanade es la detección previa

de las *Buenas Características a seguir*, desarrollada por Tomasi [ST94].

Este método tiene como meta buscar los autovalores más grandes, dentro del conjunto de autovalores mínimos para cada píxel. La combinación del algoritmo Lucas-Kanade piramidal con el método de Tomasi genera un comportamiento robusto ante los saltos grandes entre dos imágenes. Este punto es de especial importancia, ya que la cámara receptora está encima de un robot que tenderá a moverse bruscamente.

2.3. Conexiones Serials

La comunicación serial es una forma transmitir información bit a bit a través de un cable de envío y otro de recepción. Su protocolo de comunicación es regularmente usado en las áreas de computación y electrónica. La señal transmitida es de tipo lógica.

2.3.1. RS-232

El primer modelo de comunicación serial desarrollado que fue ampliamente aprovechado por la industria es el RS-232 [HC05]. Aunque actualmente es casi obsoleto, se siguen utilizando equipos que trabajan bajo este protocolo (Por ejemplo, impresoras).

La señal serial es digital, por lo tanto, utiliza dos valores de voltaje para indicar 0 o 1. En el caso del esquema RS-232, se utiliza voltaje negativo para el 0 y su equivalente positivo para el 1. Los valores debían mantenerse entre $[-25V, -3V]$ y $[3V, 25V]$, respectivamente. El estándar RS-232 tiene en sus conexiones físicas una serie de pines con distintas funciones de comunicación. En el Apéndice B, el Cuadro B.2 se muestra una especificación de cada uno de ellos.

2.3.2. Transmisión de la Señal

Para que la comunicación serial ocurra con éxito, se debe acordar una única velocidad de transmisión para los dispositivos que vayan a participar en la conexión.

La velocidad de transmisión se mide en Hertz (Hz) y puede variar, de forma discreta, desde 2400Hz a 115200Hz.

En el estándar RS-232 son utilizados cuatro conjuntos de bits de transmisión. En el Apéndice B, el Cuadro B.1 describe sus características y función.

2.3.3. UART o LLT

Su nombre proviene del inglés (universally asynchronous receiver/transmitter). Es el estándar moderno de la transmisión de señal serial. A nivel de software funciona igual que el protocolo RS-232 [RS110].

A nivel físico, la señal siempre tiene voltaje positivo. Uno bajo y uno alto, para el “0” y “1” respectivamente. Los dispositivos UART transmiten un bit a la vez, a la velocidad de transmisión designada para el protocolo de comunicación.

Los dispositivos UART tienden a usar 4 pines: Vin, Gnd, RX y TX. Los primeros dos suelen ser utilizados para proveer a los dispositivos de energía. Los últimos dos funcionan del mismo modo que el estándar RS-232.

En el caso de este proyecto de investigación, se trabajó con el estándar UART para la comunicación serial.

2.4. Aprendizaje de Máquinas

El aprendizaje de máquinas es una rama de la inteligencia artificial enfocada en el desarrollo de algoritmos que, a partir de datos empíricos, sean capaces de identificar patrones,

hacer predicciones o producir comportamientos inteligentes. En otras palabras, en vez de programar la solución de un problema de forma directa, se buscan métodos que le permitan al computador identificarla o generarla, basándose en los datos que les proveamos [Sch].

Se dice que un programa aprende de una tarea particular T , con medida de desempeño P y tipo de experiencia E , si el sistema mejora el desempeño P en una tarea T , a partir de la experiencia E . Dependiendo de cómo se especifiquen T , P y E , el tipo de aprendizaje varía [Mit97].

2.4.1. Aprendizaje Supervisado

El aprendizaje supervisado es un tipo de aprendizaje de máquina en el cual se infiere una función a partir de datos de entrenamiento. Los datos de entrenamiento se componen de pares de entrada-salida, y la función aprendida sirve luego para generar una salida correcta a nuevas entradas, siempre que sean válidas [RN03].

Dado un conjunto de entrenamiento de N pares entrada-salida $\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \dots, \langle x_n, y_n \rangle$, donde cada y_j fue generado con una función desconocida $f(x) = y$, la tarea del aprendizaje supervisado es descubrir una función h que se aproxime a la función verdadera f [RN03]. La función h es una hipótesis y el aprendizaje es una búsqueda en el espacio de hipótesis posibles. El desempeño de h suele medirse con un conjunto de datos de prueba, distintos al conjunto de datos de entrenamiento. La función inferida es llamada función de clasificación o regresión, dependiendo de si la salida corresponde a valores discretos o continuos. Técnicamente, en el caso de la función de regresión, se busca el valor esperado o un valor promedio de y , ya que la probabilidad de haber encontrado el valor exacto de y es 0 [RN03].

2.4.2. Redes Neurales Artificiales

Las Redes Neurales Artificiales (ANN) son un modelo de aprendizaje inspirado en el funcionamiento del sistema nervioso. Una red neural está compuesta por un grupo de unidades de procesamiento interconectadas, que responden en paralelo a un conjunto de señales de entrada y producen señales de salida. En las redes neurales, estas señales corresponden a valores reales, y la salida de una unidad forma parte de la entrada de otra unidad, como en el caso de las neuronas biológicas [Mit97].

2.4.2.1. Neurona Artificial

Una neurona artificial es una función matemática, definida como una abstracción de las neuronas biológicas, y son las unidades que componen a una Red Neural Artificial (ANN). Una neurona artificial recibe una o varias entradas (dendritas) y las utiliza para producir una salida (axón).

Cada neurona artificial recibe $m + 1$ entradas, a su vez cada una de esas entradas tiene un peso w_i asociado. Usualmente, a la entrada x_0 se le asigna el valor 1, el cual funciona como un valor de sesgo, lo que deja a la neurona con m valores de entrada reales. La neurona aplica una función de activación sobre el producto punto del vector de los valores de entrada por un vector de los pesos correspondientes. Las funciones de activación usualmente son de tipo sigmoide y se definen por la ecuación $\sigma(x) = \frac{1}{1 + e^{-t}}$.

2.4.2.2. Redes Multicapa

Las redes multicapa son redes formadas por una secuencia de capas, donde cada capa se compone de un conjunto de neuronas. Las redes multicapa pueden estar totalmente conectadas, en ese caso, cada una de las salidas de las neuronas de la capa i es entrada de *todas* las neuronas de la capa $i + 1$. También pueden estar parcialmente conectadas, caso en el que

cada salida de las neuronas de la capa i es entrada de una serie de neuronas de la capa $i + 1$, correspondientes a una región.

Existen tres tipos de capas; capas de entrada, formadas por las neuronas sólo que introducen los patrones en la red y que no realizan ningún procesamiento, capas ocultas, formadas por neuronas cuyas entradas provienen de capas posteriores y cuyas salidas son pasadas a capas posteriores, y capas de salida, formadas por neuronas que derivan los resultados definitivos de la red.

2.4.2.3. Propagación Hacia Atrás

La propagación hacia atrás (*Backpropagation* en inglés) es un algoritmo de aprendizaje supervisado que se utiliza para entrenar Redes Neurales Artificiales. Es un algoritmo que parte de una asignación inicial aleatoria de los pesos, y haciendo uso del algoritmo de descenso del gradiente, intenta minimizar el error cuadrático entre la salida de la red y la función objetivo [Mit97]. Es necesario que la función de activación de las neuronas sea diferenciable, condición que cumple la función sigmoidal. Esta función tiene también la particularidad de que su derivada se puede expresar en función de sí misma $\sigma(x) \cdot (1 - \sigma(x))$, lo que simplifica los cálculos en el algoritmo.

El algoritmo puede verse más claramente en dos fases. Una primera fase, en la que se realiza propagación hacia adelante de la entrada de un patrón de entrenamiento a través de la red, seguida de una propagación hacia atrás usando el objetivo del patrón de entrenamiento para generar el error δ de todas las neuronas de las capas ocultas y de salida. En la segunda fase, se actualizan los pesos; para cada neurona se obtiene el gradiente del peso y éste actualiza en la dirección opuesta al gradiente, sustrayéndole un porcentaje, denominado tasa de aprendizaje. Estas dos fases se repiten hasta que el desempeño de la red resulte satisfactorio.

Capítulo 3

Interfaz del Robot RCRover

En este capítulo se presenta el proceso de desarrollo de las distintas conexiones necesarias entre el robot y la computadora. El equipo robótico utilizado es el RCRover REV9-18 [RC11], que es un modelo todo terreno. Éste cuenta con: una cámara, una garra, un módulo de recepción de video y control remoto. La Figura 3.1, muestra el Kit completo del robot, asimismo el Apéndice ?? describe en mayor detalle sus características.



Figura 3.1: Kit del Robot modelo REV9-18. En rojo se señalan sus componentes principales.

El robot RCRover básico no posee un componente programable o de procesamiento, de

Componente	Descripción
Cámara	Ubicada en el frente y a la izquierda del robot. Permite una rotación horizontal y vertical de aproximadamente 160°. Su resolución es de 640x480 y trabaja con señal analógica de video.
Garra	Ubicada en el lado derecho del robot. Cuenta con un brazo de un solo eje, que puede rotar verticalmente hasta 210°aproximadamente. Al final del bazo se encuentra la garra, que puede abrirse hasta 90°.
Transmisión de Video	Consiste en una antena en la parte trasera del robot que emite información a una estación receptora. Ésta capta las señales de la cámara y las reproduce en un video analógico.
Control	Permite controlar el movimiento de ruedas, cámara, brazo y garra. Su duración en uso continuo es de una hora aproximadamente.
Ruedas	Consiste en dos conjuntos de orugas, que le dan una cualidad de <i>todo terreno</i> al equipo.

Cuadro 3.1: Componentes básicos del Robot RCRover

modo que no existe manera de producir comportamientos autónomos utilizando únicamente este equipo. Para poder generar conductas inteligentes, un sistema externo al robot es necesario. Éste sistema requeriría una serie de equipos de hardware e implementaciones en software para permitir la recepción de la imagen, su procesamiento y el envío de comandos al robot. Adicionalmente, es deseable la opción de poder añadir sensores adicionales, para complementar el video.

El objetivo para esta etapa de desarrollo consistía en crear el sistema mencionado. A continuación se despliega su descripción clasificada según los requerimientos funcionales. El equipo sobre el cual realizó la implementación es una MacBook Pro de 15 pulgadas, con sistema operativo OSX 10.6.

3.1. Captura de Video

El receptor de video del RCRover tiene una salida de tipo RCA como se muestra en la Figura 3.2. La captura de video requiere de un equipo especial. Para este proyecto en particular, se necesita que posea tres características:

- Compatible con hardware y software de Mac.
- Compatible con la interfaz IEEE 1394 [Ass08].
- Permitir la manipulación del video capturado.

En el mercado la mayoría de los equipos de captura trabajan con software propietario, que no permiten una manipulación posterior del video. También es difícil encontrar equipos con las características de compatibilidad anteriormente mencionadas. El Apéndice D despliega una lista de todos los hardware de captura que fueron considerados en esta etapa del desarrollo.

Se escogió equipo Canopus AVDC 110 [GVU12], debido a sus posibilidades esperadas de satisfacer los tres requerimientos. Este hardware tiene la capacidad de recibir la señal analógica y la procesa a digital; sus detalles técnicos se describen en el Apéndice D. La información del equipo es enviada por un cable de salida de tipo Firewire o IEEE 1394 de cuatro a seis pines. En el caso de este proyecto, la computadora utilizada trabaja con seis pines.

Seguidamente se describen, en orden cronológico, las acciones realizadas durante este proceso, limitaciones encontradas y soluciones escogidas. La figura 3.3, muestra una secuencia que resume el trabajo llevado a cabo.

En primera instancia, se decidió utilizar la librería *OpenCV2* [Bra12] para realizar la captura y el procesamiento del video. Ésta librería es la herramienta libre para visión de computadoras más completa y mejor documentada que existe en el área. Es importante

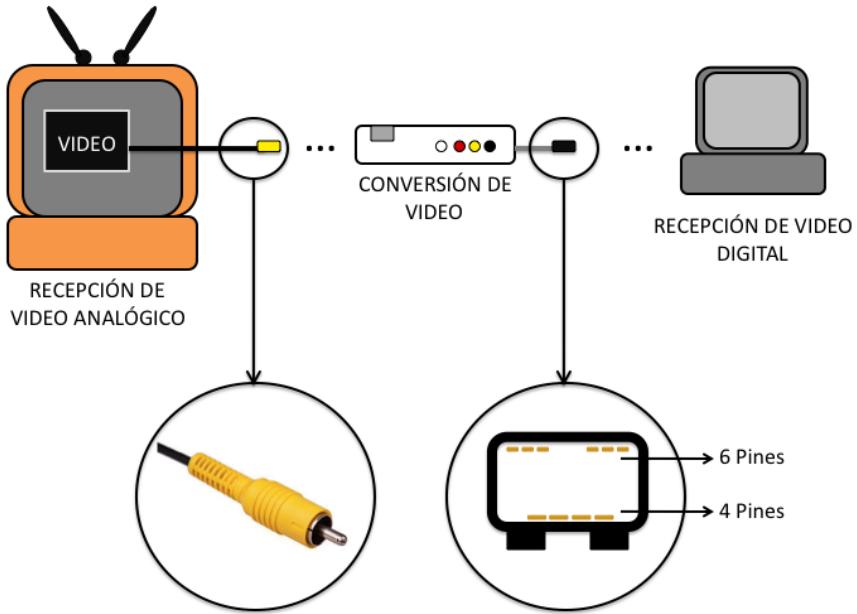


Figura 3.2: Esquema del funcionamiento del hardware para la recepción de video. La imagen del componente de video RGA fue tomada de [Amo10]

mencionar que anteriormente se habían realizado trabajos exitosos utilizando esta librería bajo el software de Mac OSX [VC09], de modo que se esperaba que la implementación de la captura del video fuera un proceso trivial. Sin embargo, el equipo de captura de video canopus® no era detectado por la librería.

Posteriormente se encontró que el ambiente de programación *Processing* [FR11] captaba exitosamente la señal por Firewire. Sin embargo, esta herramienta sólo provee funciones de dibujo 2D y 3D. En consecuencia, todo el procesamiento de imágenes debía implementarse separadamente. Otra desventaja que conlleva el uso este ambiente es la dificultad para integrar con otras librerías. Esto acarrea la aparición de limitaciones de peso al momento de añadir extensiones.

Debido a que las ventajas ofrecidas por *Processing* no hacían contrapeso suficiente contra sus limitaciones, se buscaron alternativas en librerías específicas para la captura de video con

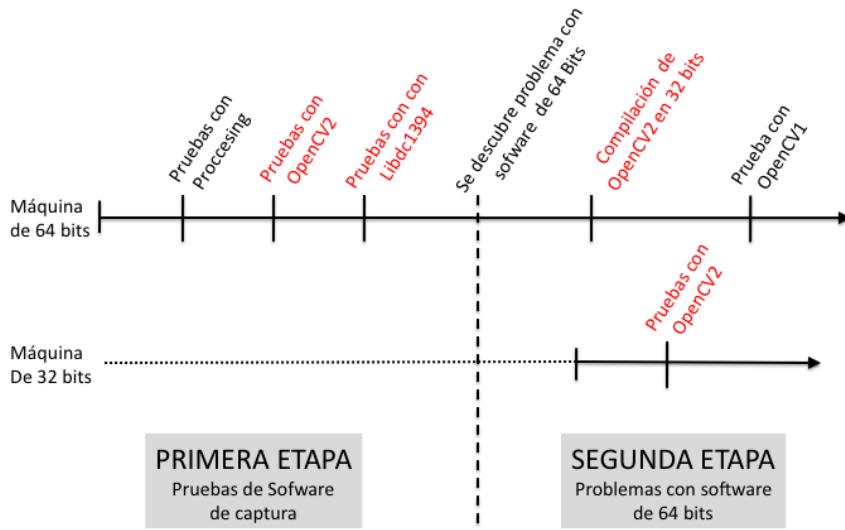


Figura 3.3: Esquema de las pruebas realizadas en orden cronológico. La línea de arriba muestra las pruebas hechas en el equipo de 64bits disponible. La línea de abajo muestra las pruebas hechas sobre la máquina de 32bits disponible. Las pruebas que no tuvieron resultado satisfactorio están marcadas en rojo.

Firewire (IEEE 1394).

Libdc1394 es un API para detección de cámaras y captura de video provenientes de conexiones del tipo IEEE 1394 [Dou11]. Esta es la única librería que funciona en Windows, Mac y Linux actualmente. El API usa como parte de su soporte a la librería libraw1394 . Por otro lado, otras librerías orientadas a la captura por Firewire como Camwire y Gstreamer están hechas sobre el API Libdc1394 [19311]. De modo que, al trabajar con esta librería se estaban probando 4 herramientas al mismo tiempo. El resultado de trabajar con Libdc1394 fue negativo; el equipo de captura, al igual que con OpenCV, no era detectado. El Apéndice E explica en mayor detalle las pruebas realizadas con las librerías.

Se detectó finalmente que los problemas encontrados son inherentes a los sistemas de 64bits. La implementación a bajo nivel aún no asegura el funcionamiento para la mayoría de los equipos de video. Para OpenCV2 en 64bits, por ejemplo, se posee una lista de los dispositivos que funcionan con la librería de manera segura [CVW11]. Hasta el momento sólo

existen 43 cámaras que funcionan con certeza. Ningún hardware de captura de video aparece listado aún.

La razón por la que *Processing* produce un funcionamiento correcto, se debe a que está implementado en arquitectura de 32 bits. Inicialmente se buscó compilar la librería OpenCV2 en 32 bits. Algunas de sus dependencias no podían ser recompiladas en esta arquitectura, de modo que no fue posible instalar OpenCV2. Seguidamente se encontró una versión precompilada de OpenCV1 en 32 bits. Esta logró capturar el video satisfactoriamente. El Cuadro 3.2 compara las tres herramientas de procesamiento de imágenes mencionadas. Debido a las ventajas ofrecidas por OpenCV1, se escogió esta librería para desarrollar el proyecto.

Funcionalidades	OpenCV2	OpenCV1	Processing
Captura de Video	No	Sí	Sí
Generar de Video en tiempo Real	Sí	Sí	Sí
Filtrado de Imágenes	Extenso y Completo	Extenso y Completo	No
Transformación de Imágenes	Extenso y Completo	Extenso y Completo	No
Flujo Óptico	Extenso y Completo	Sí	No
Aprendizaje de Máquinas	Extenso y Completo	Sí	No

Cuadro 3.2: Funcionalidades ofrecidas por *OpenCV1*, *OpenCV2* y *Processing* en 32bits para un Sistema en 64bits

Mientras se probaba la librería OpenCV1, otra alternativa estudiada fue trabajar con un equipo de 32bits que poseyera entrada de Firewire. Se utilizó una MacBook Pro con OSX 10.4. La cantidad de fotogramas o cuadros por segundo que se podían procesar era de aproximadamente 8Hz, sin pasar por algún filtro adicional, lo que la hace muy lenta para producir salida en tiempo real. Adicionalmente, la imagen se congelaba por algunos segundos en momentos aleatorios, de modo que se decidió seguir trabajando con el sistema de 64bits. De este punto en adelante se trabajó con la limitante de que cualquier librería a utilizar, debía

poder compilarse en 32bits, ya que es necesario para que pueda usarse junto con OpenCV. En los casos de librerías con pocas dependencias, esto no suele ser un problema.

3.2. Control del Robot

Para el control del robot se cuenta con un dispositivo creado en la Universidad Simón Bolívar por Tomás Chiesa, con la asesoría de la Prof. Claudia Pérez [Chi11]. Éste recibe, a través de un cable USB-Tipo A/B, paquetes seriales con la información de los comandos de control. La figura 3.4 muestra un esquema de la conexión física.

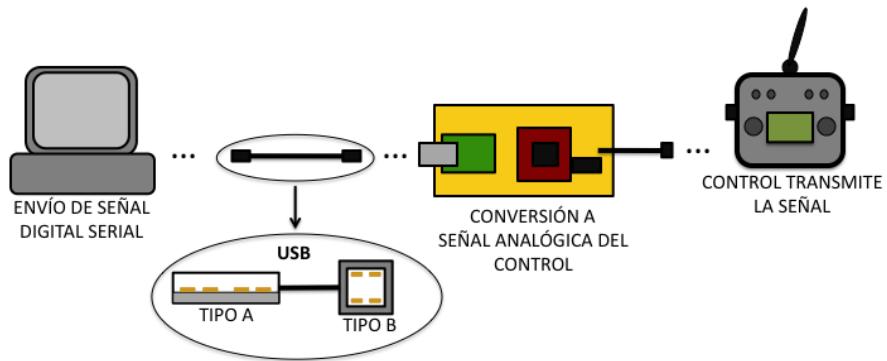


Figura 3.4: Esquema de las conexiones de hardware para el control del robot desde la computadora

Cada comando de movimiento se almacena en una estructura de 7 Bytes. En ese espacio se guarda la información para los siguientes atributos, cada uno con un rango de valores numéricos posibles:

- *Velocidad recta de las ruedas*: Indica la velocidad hacia adelante o hacia atrás del robot. Su rango es de $[0, 254]$. El valor 127 representa la velocidad cero, mientras que los extremos son la máxima velocidad hacia el frente y hacia atrás.
- *Velocidad de Giro de las ruedas*: Especifica la velocidad del giro hacia la izquierda o



Figura 3.5: Diagrama del paquete que se envía por conexión serial al dispositivo de comunicación.

hacia la derecha. Su rango es de [0, 254]. El valor 127 indica ausencia de giro, mientras que los extremos son la máxima velocidad hacia cada lado.

- *Ángulo del brazo*: Especifica la posición del brazo del robot. Su rango es de [0, 254].
- *Apertura de la Garra*: Indica el ángulo de separación entre las dos tenazas de la garra. Su valor se sitúa en el rango de [0, 127].
- *Ángulo de inclinación de la cámara*: Se maneja en el rango de valores de [0, 254]. Denota la posición de la cámara en eje vertical.
- *Ángulo panorámico de la cámara*: Denota la posición de la cámara en el eje horizontal. Su rango de valores es [0, 254].
- *Luz*: Booleano que indica si la linterna de la cámara está encendida o no. Posee dos únicos valores: 0 y 127.

Los dos primeros atributos generan, en combinación, el movimiento del robot. A mayor velocidad de giro y menor velocidad recta, el radio de giro se hace más pequeño. En el caso de no tener velocidad recta, el robot da vueltas sobre su eje. La figura 3.5 muestra cómo se distribuye la información dentro del paquete.

Seguidamente se expone el desarrollo de software que permite la comunicación entre el dispositivo creado en la Universidad Simón Bolívar y la computadora. Se presentan dos librerías. La primera, es la herramienta creada por el desarrollador del dispositivo de comunicación [Chi11]. La segunda, es la librería desarrollada en este proyecto.

Las funcionalidades principales de ambas librerías son: configuración del protocolo de comunicación, abrir puerto, cerrar puerto, escribir en el buffer del puerto, sistema de mensajes de error, creación del mensaje serial y modificación completa o parcial del mensaje. Particularmente, para el protocolo de comunicación de este proyecto se necesita poder configurar la velocidad de transmisión, bits de parada, opción de handshake, tamaño del mensaje y paridad.

La figura 3.6 muestra el pseudocódigo de la comunicación implementada en las dos librerías mencionadas para el dispositivo serial que controla el robot.

```

1 Se abre el puerto
2 While (No sea enviado el comando de salida)
3     Se establece el mensaje
4     Se envia el mensaje
5     if (se recibe el comando de salida)
6         Cerrar Puerto
7         break;
8     end if
9 end While

```

Figura 3.6: Algoritmo del esquema de comunicación serial.

3.2.1. Librería de Comunicación Serial para Windows

La librería desarrollada por Chiesa [Chi11] para el hardware de comunicación serial se implementó bajo el sistema operativo Windows VistaTM. El software está basado en la librería *windows.h* [Mic12]. Este encabezado da acceso al API completo de Win32. Debido a que la implementación de conexiones seriales requiere llamadas directas al sistema y trabajo a un

relativo bajo nivel, esta librería utiliza exhaustivamente las funciones, clases y tipos de datos que ofrece Win32. Esto trae una serie de desventajas:

- Para cambiar versión de sistema operativo Windows se deben cambiar los *macros* de la librería usados en la implementación.
- Funciona únicamente en Windows.
- Se trabaja sobre un ambiente de desarrollo que no es abierto.
- No existe un equivalente entre los tipos y clases utilizados. Las funciones de envío, recepción e incluso el tipo de dato usado para el paquete de mensaje (WindowsMessage), no poseen un equivalente en Unix.

Tomando en cuenta estas limitaciones, se decidió implementar una nueva librería que operara bajo sistemas Unix. Debido a que no existe una manera de hacer una traducción directa entre las librerías de Windows y Unix, se desarrolló una nueva librería.

3.2.2. Librería de Comunicación Serial para Unix

Esta librería está basada en un proyecto abierto y libre desarrollado por *Direcs* [DIR12]. El proyecto, a su vez, se apoya sobre la librería *termios* de *C* [Gro]. Esta librería permite implementación de la escritura y lectura sobre un buffer para un puerto serial. Sin embargo no permite un buen control de errores ni provee funciones que permitan enviar paquetes de tamaño variable de forma directa. La librería de *Direcs*, simplifica las opciones de configuración. De manera sencilla permite establecer las características del protocolo de comunicación: tamaño de mensajes, paridad y bits de parada. Por otro lado provee un mejor manejo de excepciones.

La librería de comunicación desarrollada (*RoverSerial*) está orientada a la transmisión de información hacia el dispositivo serial. El protocolo de comunicación particular para este

equipo establece tamaño de carácter en 8 bits, sin paridad, un bit de parada y una velocidad de transmisión de 9600.

Se provee una clase llamada RoverSerial que encapsula todas las funciones relacionadas con la comunicación serial. Una vez abierto el puerto, permite la creación de paquetes de comandos, y la modificación por separado de cada instrucción: velocidad recta, velocidad de giro, brazo, garra, luz, ángulo panorámico y ángulo de inclinación de la cámara. En el Apéndice F se despliega la documentación de la librería.

3.3. Sensores Adicionales

Para incrementar la capacidad perceptiva del robot se decidió añadirle un sistema de sensores de distancia. A nivel de hardware se implementó con ArduinoTM [Cua12].

Arduino es una plataforma electrónica abierta para diseño de prototipos. Posee su propio ambiente de programación, que permite descargar los programas a la tarjeta de manera directa. Su lenguaje está basado en *C*; posee una serie de librerías para el manejo de comunicación y dispositivos externos. Adicionalmente, posee una comunidad de desarrollo que publica librerías abiertas para utilización de una gran variedad de elementos (Motores, sensores, pantallas, y una gran diversidad de componentes electrónicos).

La comunicación entre el arduino y la computadora debía realizarse de manera inalámbrica. Se utilizó el adaptador de Xbee [XBE12] para cumplir con ese requerimiento. El adaptador se coloca directamente encima de la tarjeta y transmite la señal a un pequeño receptor que se conecta directamente a la computadora. Para que los dos módulos del Xbee puedan comunicarse, es necesario configurar los algunos parámetros. Ambos módulos deben tener la misma identificación para la red, estar en el mismo canal y tener la misma velocidad de transmisión. Adicionalmente se debe configurar la dirección destino para la comunicación inalámbrica.

En el modelo trabajado, la comunicación con la tarjeta se da de manera unidireccional. El

arduino recibe la información de los sensores y la envía constantemente. El mensaje consiste en una cadena de números, que representan distancias en centímetros, separados por un carácter “/”. Cada distancia proviene de un sensor distinto. El mensaje termina con el carácter ”%”.

La librería para la recepción de la información en el equipo también está basada en la librería *Direcs*. La apertura y cierre del puerto serial se realiza de la misma manera. La lectura del mensaje se realiza Byte a Byte. La Figura 3.7 muestra el pseudocódigo de la recepción del mensaje.

```

1 char caracter = nulo
2 char [] caracteres = nulo
3 int i = 0
4 Se abre el puerto
5 While (No se reciba el comando de salida)
6     While(caracter diferente de '\%') // Espera el caracter terminador
7         caracter = leer_un_Byte();
8     end while
9     caracter = leer_un_Byte();
10    While(caracter diferente de '%') // Hasta que no termine el mensaje
11        While(caracter diferente de '/') // Hasta que no termine el numero
12            caracteres[i] = leer_un_Byte();
13        end while
14        convertir el arreglo de caracteres a entero
15    end while
16    if (se recibe el comando de salida)
17        Cerrar Puerto
18        break
19    end if
20 end While

```

Figura 3.7: Pseudocódigo de recepción de la información de los sensores que el arduino envía.

Para este proyecto se trabajó con tres sensores ultrasonido modelo SRF04 [SRF]. Su rango de mayor precisión de medición está entre los 10cm y 150cm. Estos sensores detectan distancias en un cono de visión de aproximadamente 60°. Debido a que el ambiente es desconocido, es recomendable tener la menor cantidad posible de puntos ciegos. Esto le proporciona una ventaja al ultrasonido sobre los sensores infrarrojos, cuyo cono de visión se asemeja al de una línea recta.

Se colocaron tres sensores en el robot. Uno en la garra, para medir la distancia a los objetos al momento de recolectarlos. Otro en la cámara, que le da al robot un cono de visión total de aproximadamente 180° , al aprovechar el movimiento horizontal de la misma. El último sensor se colocó en la parte de atrás, para los movimientos en retroceso.

La frecuencia con la que ocurren errores en la medición del ultrasonido es relativamente baja, sin embargo, es una buena práctica reducir la cantidad de errores al mínimo para evitar comportamientos anómalos en el robot. Con este objetivo en mente, se implementaron dos métodos para minimizar el error. Ambos toman diez medidas del sensor, las procesan y devuelven una medida resultante. Se decidió utilizar diez porque es el número más grande que no afecta la velocidad de desempeño del algoritmo.

El primer método consiste en obtener el promedio de las diez medidas. El segundo método se denomina *eliminación de colas* y consiste en una mejora del anterior. Se parte del cálculo del promedio, luego se obtiene la desviación estándar, seguidamente se eliminan todas las medidas que se salgan del rango $|promedio| \pm desviacion$ y por último se vuelve a calcular el promedio de las restantes. El Cuadro 3.3 muestra el promedio de errores obtenidos en 600 mediciones. Debido a que el comportamiento de los sensores cambia si están estáticos o en movimiento, se separaron los resultados en esas dos categorías.

	Estático	Movimiento
Normal	0.0016	0.0083
Eliminación de Colas	0.0	0.0
Promedio	0.0033	0.0

Cuadro 3.3: Se muestra el promedio de medidas erróneas del ultrasonido cuando el robot se mueve y cuando está detenido. Se comparan tres distintos métodos. La medición normal sin modificaciones, el promedio de las medidas y el promedio de las medidas después de realizarse una eliminación de colas.

Debido a que el método de eliminación de las colas elimina el error por completo, se escogió como el procedimiento estándar para obtener mediciones en este proyecto.

Capítulo 4

Seguimiento de Objetos

En este capítulo se describe y presenta el proceso utilizado para lograr el seguimiento de objetos con y sin información previa a través de la cámara que posee el robot.

El desarrollo descrito en las primeras secciones tiene como objetivo principal el crear una interfaz que le permita al humano, inicialmente, indicarle al robot el objeto que desea recolectar, como se indica en la Figura 4.1; seguidamente que éste pueda ubicarse de manera autónoma frente al elemento meta, y finalmente que lo tome. Este objetivo se deriva del problema mencionado en el capítulo 2, sección 2.1. Al trabajar con un equipo que posee una sola cámara, teleoperar el robot a través de una imagen plana resulta complejo, difícil e incómodo para las personas.

Para poder recoger un objeto genérico, sin entrenamiento, en estas condiciones, el flujo óptico resulta ser una excelente opción. Debido a que el robot está en constante movimiento, ya sea de la cámara o de la estructura completa, existe un cambio continuo a través de cada cuadro del video transmitido. En este ambiente puede detectarse el flujo óptico de manera permanente, de modo que el uso de esta metodología permite aprovechar las circunstancias del problema. Por otro lado, el flujo óptico puede implementarse con costos algorítmicos que permiten interacción en tiempo real. Una vez escogido el método de flujo óptico, se plan-

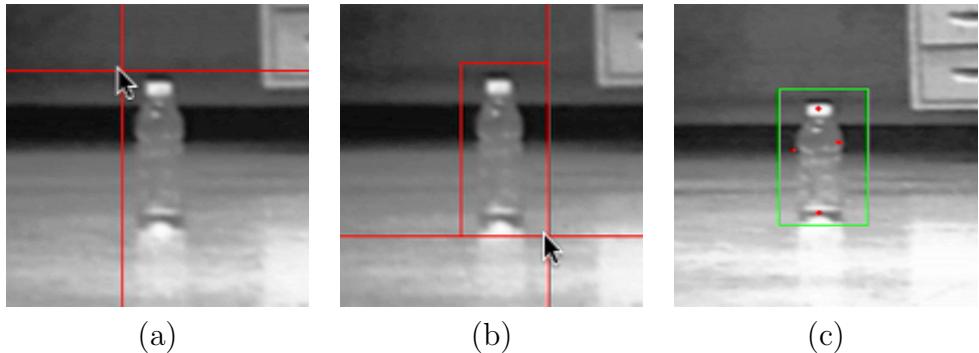


Figura 4.1: (a), (b) Selección de la imagen. (c) Método de seguimiento del objeto.

teó como meta principal, implementar el seguimiento de la menor cantidad de píxeles posibles sin afectar el desempeño. Por lo tanto, se creó un algoritmo de escogencia y eliminación de puntos que va de la mano con el flujo óptico.

4.1. Lucas-Kanade contra Flujos Densos

En términos generales, los algoritmos de flujo óptico trabajan sobre una cuadrícula densa de píxeles en la imagen, como se muestra en la parte izquierda de la Figura 4.2. Los métodos Horn-Schunck [HS81] y Farneback [Far01], son ejemplos de este caso. Dependiendo de la precisión buscada, la cuadrícula se hace más compacta.

En contraste a lo anterior, el algoritmo Lucas-Kanade [LK81], tiende a utilizarse con píxeles esparcidos de manera no uniforme sobre la imagen. Esto se debe a que éste obtiene resultados superiores si los puntos a seguir son esquinas o bordes, escogidos por el algoritmo de Mejores Características a Seguir [ST94]. El uso en conjunto de estos dos métodos es usualmente referido como el algoritmo Lucas-Kanade-Tomasi. Cuando se quiere detectar y seguir un objeto en particular, que no se encuentra en alguna posición predeterminada, el enfoque Lucas-Kanade provee una variedad de ventajas iniciales, que se mencionan a continuación.

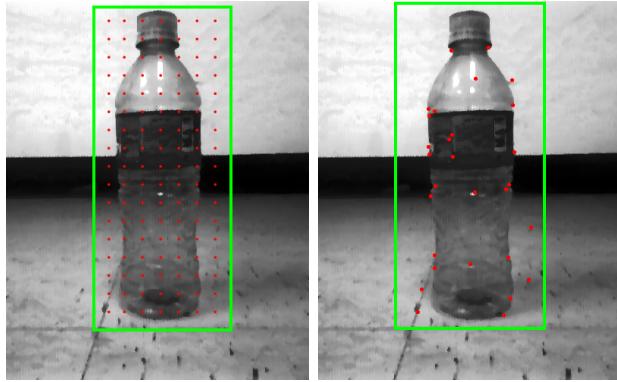


Figura 4.2: Los puntos rojos representan los píxeles cuyo desplazamiento será calculado. A la izquierda se muestra una cuadrícula común para detectar el flujo óptico. El 65 % de los píxeles son tangentes de la botella. A la derecha se muestran los puntos escogidos por el algoritmo de Mejores Características a Seguir. El 86 % de los píxeles son tangentes botella. Ambas imágenes están preprocesadas bajo un filtro de ecualización de histogramas, seguido de un filtro de mediana.

El en caso particular de este trabajo, el usuario inscribe al objeto dentro de un rectángulo. La nueva posición del objeto podría calcularse mediante el promedio de la desviación de los píxeles dentro del recuadro o de las velocidades del campo vectorial que forman la cuadrícula. Un problema inmediato que surge de estas condiciones es que algunos puntos pertenecerán al fondo y no al objeto de interés. Dependiendo del cambio de la perspectiva de la cámara y la distancia con los elementos en el fondo, los elementos alejados del punto de observación y del objetivo se trasponen a menor distancia que los píxeles del objeto meta. La figura 4.3 ejemplifica el fenómeno mencionado.

Al momento de promediar, la información proveniente de los píxeles del fondo agregará error al cálculo, que puede acumularse hasta ocasionar la pérdida eventual de la posición del objeto buscado. Para suavizar el efecto del error, se podría utilizar un estimador de la varianza que delimitara un rango en el desplazamiento de los píxeles. De modo que sólo aquellos puntos que pertenezcan a ese rango podrán ser tomados en cuenta para calcular el promedio de la próxima posición.

Aún así, la cantidad de puntos en el fondo podría afectar la media de tal manera, que

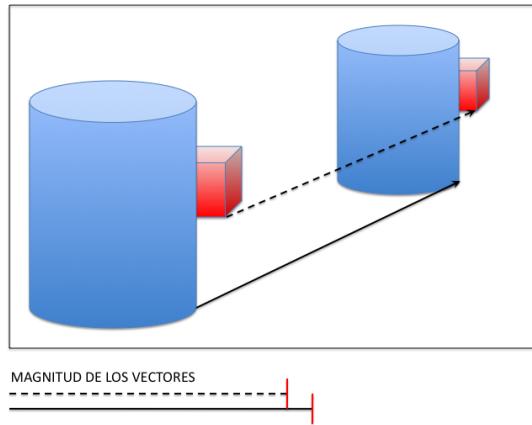


Figura 4.3: Dos objetos desde dos perspectivas distintas, el azul es el más cercano y el rojo el más alejado del observador. El desplazamiento del objeto azul es representado por la línea sólida y el del rojo por la línea pespunteada. A primera vista, los objetos dan la impresión de haberse desplazado de igual manera en la imagen. Sin embargo, el objeto rojo tiene una magnitud de cambio menor a la del objeto cercano.

sea imposible distinguir únicamente los píxeles de interés. Como muestra la figura 4.2, el método Lucas-Kanade-Tomasi, al tener la tendencia de detectar esquinas, puede reducir drásticamente la proporción de píxeles del fondo en muchos casos. De esta forma, una cantidad mucho más escasa de puntos en el fondo podría entrar dentro del rango basado en el estimado de la varianza. Para verificar esta inferencia, se tomó un total de 150 imágenes de botellas desde el robot, a uno, dos y tres metros de distancia. Seguidamente se enmarcó cada botella de manera tangencial (cada lado del rectángulo toca algún píxel del objeto) y no tangencial. Para el segundo caso, se amplió el cuadro tangencial en un 20 %. Posteriormente, se generaron puntos densos y no densos dentro para cada rectángulo separadamente.

Para generar una malla densa de puntos se utilizó una cuadrícula. En las imágenes donde la botella se encuentra separada a un metro, se trabajó con sub-rectángulos de 25×25 píxeles. En el caso de estar a 2 metros de distancia se utilizaron cuadrados de 20×20 y finalmente, para las botellas a tres metros se trabajó con rectángulos de 10×10 píxeles. Por otro lado, se usó el método de Tomasi para la producción de espacios poco densos. El cuadro 4.1, muestra

el promedio del porcentaje de píxeles escogidos que se encuentran fuera del objeto de interés. Es observable de manera clara que el algoritmo de Mejores Características a Seguir, beneficia naturalmente la selección de los píxeles que tocan la botella.

Método	No Tangencial			Tangencial		
	% a 1m	% a 2m	% a 3m	% a 1m	% a 2m	% a 3m
Tomasi	14	11	15	9	8	1
Cuadrícula	34	31	30	13	10	17

Cuadro 4.1: Porcentaje de píxeles escogidos del fondo

Por otro lado, los algoritmos de flujo óptico que trabajan sobre cuadrículas tienen la tendencia a detectar con error el flujo en trozos de la imagen que sean muy uniformes. Este sería otro elemento perjudicial para el cálculo del promedio. En contraposición, el algoritmo Lucas-Kanade-Tomasi utiliza esquinas, que suelen ser puntos cercanos a una zona de contraste. Para este trabajo se escogió el uso del método Lucas-Kanade-Tomasi, porque permite descartar los puntos que acumulan errores más fácilmente, debido a su naturaleza.

4.2. Metodología

La metodología para la eliminación de puntos con errores de cálculo en el desplazamiento está aplicada sobre imágenes previamente preprocesadas. Para entender las motivaciones bajo el filtrado escogido, es necesaria la exposición previa del proceso de seguimiento y detección. El preprocesamiento al que fue sometido cada cuadro es explicado en la sección 4.3, de este capítulo.

4.2.1. Algoritmo

El proceso de seguimiento de objetos se divide en cuatro etapas principales: 1. Se calcula el flujo óptico mediante el algoritmo Lucas-Kanade-Tomasi. 2. Se calcula la varianza del

desplazamiento entre los píxeles de interés de un cuadro al siguiente y se eliminan los píxeles que se salgan del rango establecido en base a el estimador. 3. En el caso de eliminarse más píxeles de lo permitido, se escogen nuevos píxeles provistos por el algoritmo de Tomasi. 4. Se reajusta el tamaño y la posición del cuadro.

Para poder desarrollar cada punto, es necesario definir formalmente algunos elementos. Sea p_i un píxel con componentes x y y denotadas como $p_i.x$ y $p_i.y$ respectivamente. Sean $P_{in}[N]$ y $P_{out}[N]$ dos arreglos de píxeles p de tamaño N . Se define a \mathbf{Dif}_{A-B} como el arreglo resultante de restar cada $A[i] - B[i]$, con $i \in [1..N]$. Sea $\mathbf{Mean}(A)$ y $\mathbf{Dev}(A)$, dos funciones definidas en $\mathbb{R}^N \rightarrow \mathbb{R}$, que reciben un arreglo y retornan el promedio aritmético de sus valores y la desviación estándar respectivamente.

Inicialmente se detectan las esquinas favorables con el método de Tomasi en el área demarcada por el usuario. Estos se almacenan en el arreglo P_{in} . Seguidamente se usa el algoritmo Lucas-Kanade para detectar esos los puntos en el siguiente cuadro del video. Los nuevos píxeles son almacenados en el arreglo P_{out} . Luego se procede a eliminar los píxeles cuyo desplazamiento no pertenezca al siguiente rango:

$$\mathbf{Mean}(\mathbf{Dif}_{P_{out}-P_{in}}) \pm \alpha \mathbf{Dev}(\mathbf{Dif}_{P_{out}-P_{in}}) \quad (4.1)$$

El elemento (a) de la Figura 4.4 muestra una instancia en donde dos píxeles (abajo y a la derecha) cumplen con las condiciones para ser eliminados, con $\alpha = 1$. El punto superior se excede por encima y el inferior por debajo. Este último píxel posee un módulo de desplazamiento parecido al del promedio, pero no una orientación diferente en el eje vertical. Debido a que todos los cálculos son realizados por componentes, el desplazamiento en el eje y de ese punto sería negativo mientras que el del resto es positivo.

Posteriormente en esta sección, se despliegan los experimentos realizados para determinar la escogencia de $\alpha = 1,8$ como el valor óptimo de esta constante.

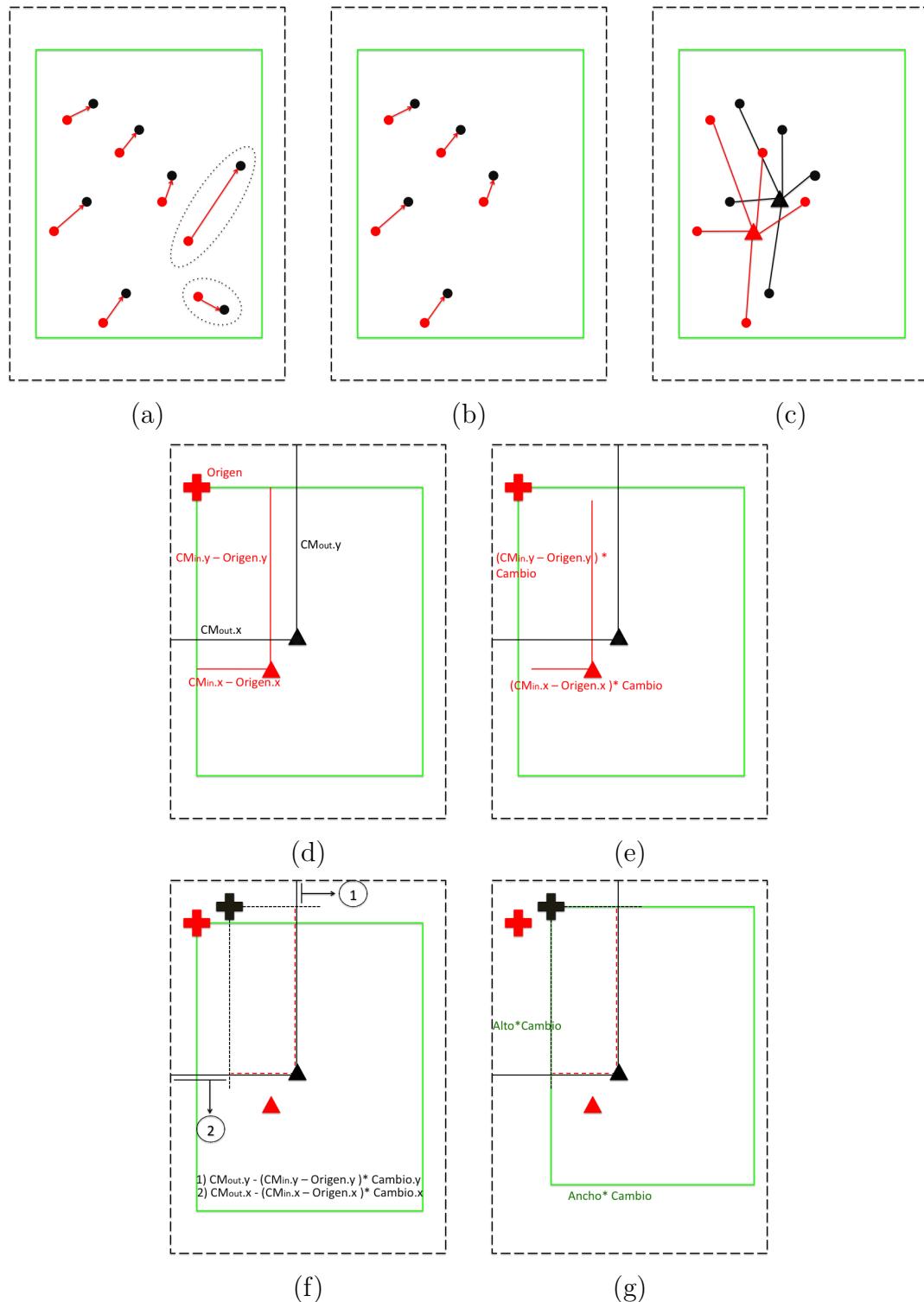


Figura 4.4: Esquema del algoritmo utilizado para mantener la posición y proporción del cuadro que enmarca el objetivo. En rojo se demarcan los objetos pertenecientes a la imagen inicial, mientras que en negro se denotan los elementos de la imagen que le sigue. Los píxeles son representados con círculos, los centros de masa con triángulos y el punto origen del cuadrado con una cruz.

Continuando con la metodología, el siguiente paso es calcular los centros de masa CM_{in} y CM_{out} , de los arreglos de píxeles P_{in} y P_{out} respectivamente. Seguidamente se obtienen las distancias desde cada punto del arreglo a sus centros de masa, como se muestra en la Figura 4.4 (c). Denotemos a los arreglos que contienen estas diferencias como Div_{in} y Div_{out} .

A partir de esta información es posible determinar el desplazamiento del cuadro y el cambio en sus proporciones. Para calcular el porcentaje de cambio en el tamaño del rectángulo, basta con tomar el cociente de los promedios de Div_{out} y Div_{in} :

$$PCambio = \frac{\text{Mean}(Div_{out})}{\text{Mean}(Div_{in})} \quad (4.2)$$

El cambio en la posición del origen está basado en el desplazamiento del centro de masa, tomando en cuenta el la variación en las proporciones del rectángulo ($PCambio$). La siguiente ecuación describe la nueva posición del origen:

$$Origen_{fin} = CM_{out} - (CM_{in} - Origen_{ini}) * PCambio. \quad (4.3)$$

Las Figuras 4.4 (d), (e), (f) y (g), muestran la secuencia de reajuste para el rectángulo. Las distancias desde el CM_{in} a los bordes de la figura multiplicados por $PCambio$, son utilizadas como las nuevas distancias desde CM_{out} a los bordes. La metodología explicada se realiza en cada iteración, produciendo una buena aproximación de la posición del objeto a seguir.

4.2.2. Experimentos para el Flujo Óptico

Los experimentos explicados a continuación se realizaron utilizando la misma metodología. Se colocó el robot a 4 metros del objetivo. Todas las pruebas son repetidas con el objeto en el centro, a la izquierda y derecha del campo de visión de la cámara, como se muestra en la figura 4.5.

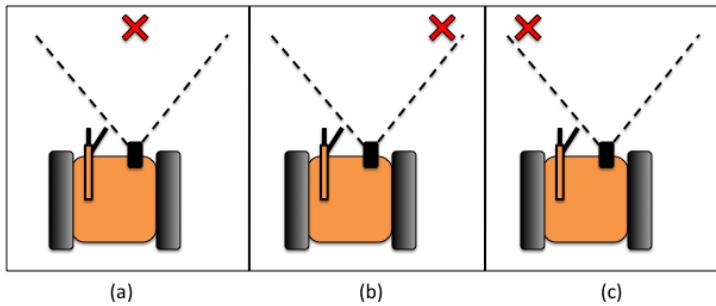


Figura 4.5: (a) El robot posee el objeto centrado en su campo de visión. (b) El robot tiene al objeto en el extremo derecho de su campo de visión. (c) El robot tiene al objeto en el extremo izquierdo de su campo de visión.

Para ajustar el valor de la constante α de la ecuación 4.1, se realizaron experimentos variándola de forma discreta dentro del rango $[1, 4]$. Se utilizó un total de 14 valores. Cada uno de ellos fue utilizado 5 veces en las tres posiciones mencionadas anteriormente, dando un total de 210 pruebas. Mediante análisis empíricos, se obtuvo que el flujo óptico se comporta de mejor manera cuando $\alpha \approx 2$. Debido a esto, el salto entre los valores escogidos disminuye, a medida que se acercan al 2.

Para las pruebas, se midió el número de veces que el flujo óptico se pierde en el recorrido de 4 metros hacia el objetivo. El flujo óptico se considera “perdido” en cualquiera de los siguientes casos: A) El objeto no es tangente con el centro del rectángulo. B) El cuadro es 70 % más grande que el rectángulo original. C) El número de píxeles para realizar el flujo óptico es menor a dos.

El Cuadro 4.2 muestra el promedio de las 15 pruebas para cada valor de α . Debido a que la interferencia, en muchos casos, altera la imagen hasta un punto donde es incluso ilegible para el humano, se colocan por separado las pérdidas del flujo ocasionadas por la interferencia y por los errores de cálculo. Esto permite evaluar de forma aislada el desempeño del flujo óptico.

Los resultados obtenidos indican que $\alpha = 1,9$ y $\alpha = 1,8$ son los valores que producen el

Valores de α	4	3	2.5	2.4	2.3	2.2	2.1	2.0	1.9	1.8	1.7	1.6	1.5	1.0
Interferencia	3.64	3.21	4.54	.4.79	4.13	4.46	5.21	4.73	4.53	4.66	4.26	3.50	3.14	5.0
Flujo Óptico	3.20	1.46	1.93	0.80	0.59	0.33	0.20	0.13	0.13	0.40	0.66	0.53	0.8	3.79

Cuadro 4.2: Promedio de veces que se pierde el flujo óptico debido a la interferencia o a debilidades propias del algoritmo, en un recorrido de cuatro metros, variando el valor de α en $\alpha * \sigma$.

mejor desempeño para el flujo óptico. Debido a que sólo se tomaron 15 muestras por cada uno, se calcularon los intervalos de confianza para los mejores α .

Para los resultados con promedio de 0,13, la media real está entre [0,0,0,32] con 95 % de confianza. Esto confirma que los valores de α aceptados generan un buen desempeño, debido a que, aún si el promedio estuviera en el extremo derecho, el resultado sigue siendo aproximable a cero.

Un indicador de mucho interés para la investigación, es el mínimo número de píxeles con el que funciona correctamente el flujo óptico. Con el objetivo de definir este indicador, se realizó un conjunto de pruebas con la misma metodología anterior, pero variando el número de píxeles iniciales utilizados por el flujo. Para estos experimentos se utilizó $\alpha = 1,9$. Los resultados se encuentran desplegados en el Cuadro 4.3. Se tomaron las medidas hasta 15 píxeles, porque el algoritmo de Mejores Características a Seguir no tiende a escoger más de esa cantidad.

Número de píxeles	15	14	13	12	11	10	9	8	7	6	5	4	3	2
Interferencia	3.07	3.21	3.57	4.66	4.33	3.35	4.93	4.26	4.59	4.46	5.06	4.79	3.78	3.28
Flujo Óptico	0.13	0.26	0.33	0.06	0.20	0.13	0.13	0.26	0.06	0.13	0.13	0.26	0.40	1.80

Cuadro 4.3: Promedio de veces que se pierde el flujo óptico debido a la interferencia o a debilidades propias del algoritmo, en un recorrido de cuatro metros, variando la cantidad de píxeles en el flujo.

Los datos revelan que a partir de cuatro píxeles, el cambio en su número no afecta el desempeño del algoritmo. Adicionalmente la velocidad del video se mantuvo en 24fps (Cua-

dros por segundo), durante todo el experimento, de modo que tampoco afectó los tiempos.

El flujo óptico puede trabajar hasta con cuatro píxeles sin comprometer los resultados. Cuando se utilizan dos píxeles el algoritmo se pierde más fácilmente. Al trabajar únicamente con un par de puntos, el flujo se declara perdido con la primera iteración. Por otro lado, se observó que el rectángulo muestra una tendencia mayor a perder sus proporciones. De esto puede concluirse que la información proporcionada por dos píxeles no es suficiente para determinar correctamente el valor de $PCambio$.

4.3. Filtrado

Esta sección explica el proceso de escogencia de la combinación de filtros utilizados en el preprocesamiento final para el flujo óptico. Los objetivos principales de este tratamiento a la imagen son: Eliminar el ruido inherente a la imagen, reducir al máximo el efecto de la interferencia sobre los resultados el algoritmo y mejorar el desempeño del flujo óptico.

Existen muchos algoritmos de preprocesamiento. Para elegir el conjunto inicial de filtros a probar, se utilizaron dos criterios. El primero es utilizar los algoritmos usualmente recomendados para trabajar con flujo óptico. El segundo es tomar en cuenta las características del objeto a seguir. En el caso de este proyecto se está trabajando con botellas de plástico, que en su mayoría están hechas de material transparente.

4.3.1. Filtros utilizados

Los filtros de difuminado, en especial el gaussiano y de mediana han probado ser excelentes algoritmos de preprocesamiento para el flujo óptico [SB12].

El filtro gaussiano es el más recomendado para el caso del algoritmo Lucas-Kanade y el que, en general, produce mejores resultados [Can86, SB12]. Sin embargo, al estar lidiando con un objeto de alta transparencia, es posible que el filtro de mediana genere un mejor

desempeño, debido a que conserva los bordes. Con el objetivo de producir una imagen en la que el objeto meta se distinga lo mejor posible del fondo, se probará utilizando la ecualización previa de la imagen antes de aplicar el difuminado.

Por último se utilizará el algoritmo Canny con un filtro gaussiano que elimine el ruido y los bordes menos importantes primero. Si el algoritmo detecta los bordes de la botella correctamente, en combinación con un difuminado fuerte, es posible que la Detección de Características a seguir obtenga una mayor cantidad de píxeles pertenecientes únicamente a la botella en contraste con los demás. El uso de Canny para el flujo óptico ha producido resultados exitosos en el pasado [CB98].

En resumen, se realizarán las pruebas bajo los siguientes procesos de filtrado: 1. Filtro gaussiano, 2. Filtro de medianas, 3. Ecualización de histogramas y difuminado gaussiano, 4. Ecualización de histogramas y difuminado de mediana y 5. Difuminado gaussiano y Canny.

4.3.2. Experimentos con filtros

Para medir la eficacia de los filtros se utilizó la misma metodología de experimentos mencionada en las secciones anteriores. Se colocó el robot a 4 metros del objetivo, desde tres posiciones diferentes y se tomaron 15 mediciones por cada valor a probar.

En estas pruebas se utilizaron las siguientes dos medidas:

- **Tasa de Manejo de Interferencias (TMI):** se calcula dividiendo el número promedio de veces que el flujo resistió una interferencia fuerte, entre el promedio número total de interferencias.
- **Pérdida del Flujo Óptico (PFO):** al igual que en las pruebas anteriores, es el número de veces que se pierde el flujo por los errores inherentes al cálculo.

El Cuadro 4.4, muestra los resultados de las pruebas de filtros. Con respecto al manejo de las interferencias, el filtro que proporciona mejores resultados promedio es el de mediana

Filtro	MED			GAU			EQ+M			EQ+G			CANNY		
	3	5	7	3	5	7	3	5	7	3	5	7	5	7	9
TMI	0.3	0.2	0	0.3	0.3	0.2	0.4	0.5	0.6	0.4	0.4	0.4	0	0	0
PFO	0.7	0.3	0.0	0.6	0.4	0.3	1.0	0.3	0.4	1.0	1.2	0.7	2.3	2.7	2.6

Cuadro 4.4: Tasa de manejo de interferencias y promedio de pérdidas del flujo óptico para distintas combinaciones de filtros. Los enteros debajo del nombre de cada filtro indican el radio o ventana del difuminado. La notación para los nombres es la siguiente: *MED*, filtro de mediana. *GAU*, filtro gaussiano. *EQ+M*, ecualización de histogramas y filtro de mediana. *EQ+G*, ecualización de histogramas y filtro gaussiano. *CANNY*, filtro gaussiano y Canny.

con previa ecualización de histogramas. Dentro de esta categoría, el difuminado más fuerte da los indicadores más deseables, manejándose el 60 % de la interferencia. Sin embargo, el filtro de mediana es pesado y cuando su ventana es mayor a 5, se reduce la velocidad de los cuadros en el video. Tomando esto en cuenta, se decidió utilizar el segundo mejor resultado: ecualización y filtro de mediana con radio 5, que maneja el 50 % de las interferencias. Con respecto al manejo del flujo óptico, el filtro de mediana con ventana 5 o 7 produjo buenos resultados con y sin ecualización.

La imagen trabajada en este proyecto es más ruidosa de lo acostumbrado. Esto se debe a que en la transmisión del video ocurren interferencias que distorsionan la imagen de manera suave casi constantemente, y de manera fuerte entre períodos cortos de tiempo. Tomando en cuenta este hecho y observando los resultados, se llega a una conclusión poco intuitiva: es mejor que el píxel a seguir esté rodeado de píxeles con la misma intensidad, aumentando la posibilidad de que el algoritmo confunda al punto buscado con alguno de sus puntos vecinos.

Debido a que la imagen da saltos constantes que generan errores acumulados en el flujo óptico, es conveniente que exista un radio delimitado de error, dentro del cual el píxel pueda ser desplazado sin perderse. El filtro de mediana hace precisamente eso, debido a que genera cúmulos de píxeles del mismo valor, donde el punto puede “saltar sin salirse de esa área. La ecualización de histogramas mejora el resultado porque resalta drásticamente los bordes de la botella y evita que se difumine con el fondo.

El filtro gaussiano probó ser bueno, pero no mejor para este problema que el de mediana. Los saltos constantes desplazan más fácilmente a los píxeles seguidos de su posición correcta. El filtro de mediana conserva mejor los bordes, característica que es sumamente importante en esta instancia particular. Canny produce malos resultados debido a que la interferencia es constantemente captada como borde, dañando el flujo óptico con más frecuencia que los otros filtros. El difuminado que logra eliminar la interferencia también hace a la botella indiferenciable de su fondo.

4.4. Detección de Interferencias

El RCRover transmite las señales de video por radio, que resultó ser un medio resultó ser bastante ruidoso en la práctica. En el caso particular de este proyecto de desarrollo, la interferencia ocurre constantemente. Ésta se clasificó en tres grupos, según el nivel de distorsión que genera en la imagen: en tres grupos:

- **Suave:** Ocurre de manera constante, pero es manejable si se usa el filtrado correcto.
- **Fuerte de Borde:** Ocurre frecuentemente. Daña los bordes del cuadro, por lo que no siempre afecta al flujo óptico.
- **Fuerte:** Crea un ruido muy fuerte que cambia por completo las intensidades de los píxeles. Produce un efecto parecido al de colocar cuadros negros entre las imágenes.

La Figura 4.6, muestra ejemplos de los tres tipos de interferencia. Ésta tiene dos efectos principales: La pérdida del flujo óptico y la pérdida del foco en el robot. El primer efecto es esperado, conociendo los resultados presentados en los Cuadros 4.2 y 4.3, en la Sección 4.2.2. El problema del segundo efecto es que la cámara apunta abruptamente a otro lado, debido al salto brusco que dan todos los píxeles seguidos por el algoritmo de flujo óptico. Este salto

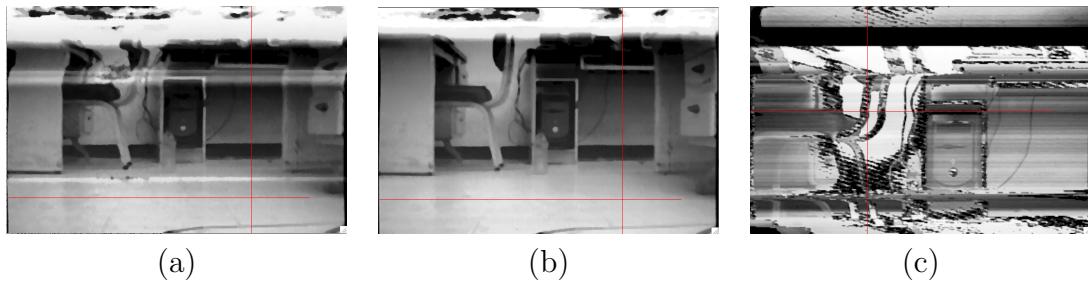


Figura 4.6: (a) Interferencia suave en el medio con interferencia fuerte de borde. (b) Interferencia fuerte de borde, (c) Interferencia Fuerte.

es interpretado como un movimiento en el cuadro que enmarca a la botella en el 90 % de los casos, sacando al objetivo del campo de visión del robot. La interferencia fuerte causa este efecto en la mayoría de los casos.

Para evitar que la cámara se mueva abruptamente cuando hay interferencia, se buscó una manera de detectarla que no agregara costos computacionales al programa. Aprovechando que el flujo óptico ya está siguiendo el movimiento en la imagen, se decidió utilizarlo para la detección de interferencias.

De manera empírica se hizo evidente que las interferencias, al cambiar drásticamente las posiciones de los píxeles, generan un cambio brusco en las proporciones del cuadro que sigue al objeto. Se realizaron 100 pruebas, en las cuales 98 % de las veces, la proporción del cuadro que enmarca la botella cambió en más de 25 %. Si se guarda la proporción del rectángulo de la imagen anterior y se compara con el rectángulo actual, se puede generar una excelente detección de interferencias sin agregar costo computacional.

Al verificar el cambio de proporción en el cuadro, se detecta la interferencia antes de que termine la iteración y puede impedirse a tiempo el movimiento de la cámara. De este modo se tiene un sistema que en el 98 % de los casos impedirá que el objeto a seguir se salga del campo de visión.

4.5. Detección de Botellas con Redes Neurales

Antes de exponer los detalles acerca del proceso llevado a cabo con aprendizaje de máquinas, es importante resaltar que en esta área es común que la recolección de datos de entrenamiento sea el paso que consume la mayor cantidad de tiempo. Tomando esto en cuenta se decidió implementar, como paso previo a la detección de botellas, un sistema semi-automático de recolección de datos, aprovechando la implementación del flujo óptico.

4.5.1. Sistema Semi-Automático de Recolección de Datos

El sistema se implementó agregándole una pequeña modificación a la metodología de flujo óptico explicada anteriormente. Al enmarcar un objeto con el ratón se activa la siguiente condición: mientras exista un cuadro de enmarcado en la pantalla, se toma una foto de esa región cada 60 cuadros.

Para tomar las imágenes de las botellas, se colocaron en distintos lugares del área de trabajo y se movió el robot manualmente alrededor de ellas, volviendo a enmarcar la botella cuando se perdía debido a la interferencia.

Para la recolección de imágenes que no son de botellas se implementó un pequeño programa donde se marca inicialmente un rectángulo en la pantalla y se toma una foto de esa área cada 60 cuadros. Al mover el robot, cambia la imagen que enmarca el rectángulo.

4.5.2. Entrenamiento y Resultados

Se decidió utilizar redes neurales debido a que han probado ser buenas para tratar con problemas de imágenes, son sencillas de configurar, entrenar y probar y finalmente, son rápidas en su ejecución. Mediante esto no se quiere decir que las redes neurales puedan producir mejores resultados que otras técnicas. El objetivo del proyecto es implementar algún mecanismo de aprendizaje y aplicarlo, mas allá de encontrar el que genere el mejor desempeño

La implementación de las redes neurales se realizó con la librería FANN (Fast Artificial Neural Network) [FAN] , por su facilidad de instalación, rapidez y buena documentación. Las redes utilizadas tienen únicamente tres capas, para simplificar su uso, debido a que va a hacer clasificación binaria (botella o no botella). Es de conocimiento general que una red de tres capas puede generar resultados correctos para cualquier problema de clasificación binaria [Roj03].

Para el entrenamiento y pruebas se tomaron 95 imágenes de botellas y 121 imágenes del laboratorio sin botella respectivamente, para un total de 432 imágenes. El tamaño de cada imagen se redujo a 45×95 píxeles para reducir los costos de cómputo. Cada píxel se utilizó como una entrada a la red, de modo que se tenían 4275 entradas. El número de neuronas en la capa oculta se varió discretamente de 5 a 100. El valor del error cuadrático medio deseado fue 0.001 y se colocó un máximo de 6000 iteraciones para reducir los tiempos de entrenamiento.

El Cuadro 4.5 muestra el porcentaje de clasificación correcta para la red, variando el número de neuronas (NN) y cambiando el filtrado inicial que se le dio a las imágenes. El mejor resultado de clasificación fue de 89 %. Lo produjo la red con 75 neuronas en la capa intermedia utilizando el filtro gaussiano.

NN	SF	M	G	EQ+M	EQ+G
5	82	85	83	86	82
10	82	82	79	79	86
15	83	78	80	80	80
25	85	79	86	80	84
50	86	84	82	82	82
75	87	81	81	80	89
100	85	80	81	84	85

Cuadro 4.5: Porcentaje de clasificación correcta de las redes neurales. **NN**: Número de Neuronas. **SF**: imagen Sin Filtro. **M**: imagen con filtro de Mediana, **G**: imagen con filtro Gaussiano. **EQ+M**: Ecualización de Histogramas y filtro de Mediana. **EQ+G**: Ecualización de Histogramas y filtro Gaussiano Todos los filtros utilizados tienen ventana de tamaño tres.

4.5.3. Redes Neurales y Flujo Óptico

Para poder clasificar una botella dentro de cada cuadro del video, se implementó un método iterativo donde se comienza con una ventana grande que se desliza por toda la imagen. En cada paso se pregunta a la red si lo que hay en la ventana es una botella o no. En el caso de no detectar una botella, se reduce el tamaño de la ventana y se vuelve a iterar.

Para reducir los tiempos se limitó el número de reducciones en el tamaño de la ventana a 7. Se comienza con un tamaño de ventana de 225×475 y en cada iteración se reduce en un 25 % hasta llegar a 52×112 . El tamaño de la imagen del video es de 640×480 píxeles.

Por otro lado se hizo el tamaño de salto en el proceso de deslizamiento por la imagen lo más grande posible. Cada salto es un tercio del tamaño de la imagen. Es decir, el salto en el *eje x* y en el *eje y* está dado por $\frac{\text{ancho}_{\text{ventana}}}{3}$ y $\frac{\text{alto}_{\text{ventana}}}{3}$ respectivamente. El problema de mover la ventana con pasos grandes es que se corre el riesgo en nunca centrar la botella en alguna de las ventanas y no poder clasificarla. Al colocar el tamaño de salto en un tercio de la proporción de la ventana se asegura que la mayor parte de la botella quede enmarcado en alguno de los saltos.

Aún haciendo estas reducciones, en el peor de los casos se puede llegar a hacer 577 clasificaciones en un solo cuadro del video. Para mejorar el desempeño, se propone utilizar el flujo óptico y clasificación con redes neurales en conjunto. Una vez detectada una botella, se enmarca con una segunda ventana 25 % más amplia que la primera (siempre que no se exceda del tamaño del video). Sobre esta segunda ventana se va a realizar flujo óptico. Hasta el momento en que se pierda el flujo óptico, la detección de la botella con la red no se hace sobre toda la imagen sino sobre la segunda ventana. De este modo se hace un máximo de siete clasificaciones por imagen. La aplicación de este algoritmo produce una reducción promedio en el número de consultas a la red en un 88 %, en los casos donde se detecta correctamente la botella. Esto se traduce en una mejora significativa del costo de cómputo.

Capítulo 5

Conclusiones y recomendaciones

Como resultado de este proyecto, se logró implementar un sistema que integrara el control del robot, el procesamiento de su imagen y el uso de sensores adicionales. Estos últimos dos puntos del sistema son utilizables con cualquier otro equipo debido a que son portables y autocontenidos.

En la aplicación para recoger botellas de plástico, para flujo óptico se logró desarrollar una variación del algoritmo para la escogencia de píxeles a seguir y un sistema de filtrado que resiste hasta el 60 % de las interferencias y reduce los errores inherentes al cálculo del flujo.

Al combinar el sistema de seguimiento de objetos genéricos con la detección de botellas de la red neural, se reduce la búsqueda del objeto al 88 % de la imagen, reduciendo los costos computacionales.

Finalmente se propone una forma sencilla de detección de interferencias con flujo óptico, que detecta correctamente en el 98 % de los casos.

Este trabajo posee ventajas que aportan beneficios en el uso del robot RCRover. En primer lugar, permite la recolección de objetos asistida sin necesidad de datos previos, por lo

que se puede seguir y recoger virtualmente cualquier objeto que el RCRover esté en capacidad de tomar y transportar.

La combinación del flujo óptico con aprendizaje de máquinas reduce los tiempos de desempeño de la detección. Esta característica es de crucial importancia cuando se trabaja con procesamiento de video en tiempo real.

Es importante mencionar que todos los resultados del desarrollo en el área de visión de computadoras son perfectamente aplicables a cualquier otro equipo que posea una sola cámara y tenga las posibilidades de transmitir sus datos de video, para que sean procesados en el computador.

Para mejorar los resultados expuestos en este trabajo, se presentan las siguientes recomendaciones:

- Estar atentos a la salida al mercado de un hardware de captura que sea reconocido por OpenCV2, para tener todo el poder de las optimizaciones de la nueva versión para procesado de video y aprendizaje de máquinas. Esto también eliminaría la restricción de trabajar con librerías compiladas en 32bits.
- Desarrollar un sistema de estabilización de la imagen, para evitar los efectos de la interferencia, lo que mejoraría los resultados del flujo notablemente.
- Evaluar otros filtros y combinaciones de los mismos, ya que es posible que exista un preprocesado distinto que produzca mejores resultados, tanto para el flujo óptico como para las redes neurales.
- Utilizar otros tipos de aprendizaje de máquinas, como árboles de decisión y SVM.
- Utilizar las distintas variaciones del algoritmo Lucas-Kanade.

Adicionalmente a las recomendaciones planteadas, se propone como trabajo futuro el utilizar aprendizaje de máquinas con el objetivo de que el robot detecte el lugar óptimo para agarrar los objetos. También se propone un sistema de clasificación que le permita al robot identificar cuándo un objeto es recolectable y cuándo no. Por último, se sugiere utilizar la información del movimiento de la cámara y del robot en el flujo óptico, ya que estos datos son conocidos cuando el robot está siendo controlado por el computador

Bibliografía

- [19311] <https://ieee1394.wiki.kernel.org/index.php/Libraries>, 2011.
- [Amo10] E. Amos. <http://commons.wikimedia.org/wiki/File:Composite-video-cable.jpg>, 2010.
- [Ass08] 1394 Trade Association. <http://www.1394ta.org/index.html>, 2008.
- [Bou00] J. Bouguet. Pyramidal implementation of the lucas kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.
- [Bra12] G. Bradski. <http://opencv.willowgarage.com/wiki/>, 2012.
- [Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), 1986.
- [CB98] Murray D. Scott G. Castelow, D. and B. Buxton. Matching canny edgels to compute the principal components of optic flow. *Image and Vision Computing*, 6(2):129–136, 1998.
- [CH96] R. Cipolla and N. Hollinghurst. Human-robot interface by pointing with uncalibrated stereo vision. *Image and Vision Computing*, 14(3):171–178, 1996.
- [Chi11] T. Chiesa. Diseño e implementación de una arquitectura de hardware para el control de un robot rc rover. 2011.
- [CM95] W. Chang and A.S. Morse. Human-robot interface by pointing with uncalibrated stereo vision. *Control Systems, IEEE*, 15(1):30–39, 1995.

- [CM07] Haas E. Chen, J. and Barnes M. Human performance issues and user interface design for teleoperated robots. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, page 1231–1245, 2007.
- [Cua12] D. Cuartielles. <http://arduino.cc/en/>, 2012.
- [CVW11] <http://opencv.willowgarage.com/wiki/FullOpenCVWiki>, 2011.
- [DIR12] http://www.direcs.de/doxygen/direcsSerial_8cpp.html, 2012.
- [Dou11] D. Douxchamps. <http://damien.douxchamps.net/ieee1394/libdc1394/>, 2011.
- [FAN] Fast artificial neural network library.
- [Far01] G. Farneback. Very high accuracy velocity estimation using orientation tensors, parametric motion models, and simultaneous segmentation of the motion field. *IEEE International Conference on Computer Vision*, 1:171–177, 2001.
- [FPD08] Lai K. Helmer S. Little J. Forssén P., Meger D. and Lowe D. Informed visual search: Combining attention and object recognition. In *IEEE International Conference on Robotics and Automation*, volume CFP08RAA-CDR, pages 935–942. IEEE Robotics and Automation Society, 2008.
- [FR11] B. Fry and C. Reas. <http://processing.org/>, 2011.
- [FW09] D. Fleet and Y. Weiss. Optical flow estimation. <http://www.cs.toronto.edu/~fleet/courses/cifarSchool09/flowChapter05.pdf>, 2009.
- [Gro] The Open Group. Termios.h.
- [GVU12] LLC. Grass Valley USA. <http://www.grassvalley.com/products/advc110>, 2012.
- [GW08] R. González and R. Woods. *Digital Image Processing*. Prentice Hall, 2008.
- [HC05] Diogenes H. Horning, D. and D. Cary. Wikibooks serial programming. http://upload.wikimedia.org/wikipedia/commons/1/1f/Serial_Programming.pdf, 2005.

- [HS81] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.
- [LK81] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.
- [MA09] R. Maini and H. Aggarwal. Study and comparison of various image edge detection techniques. *International Journal of Image Processing (IJIP)*, 3(1):1–11, 2009.
- [Mel11] J. Meltzer. *Vision for Robot Navigation, Localization, and Object Recognition*. PhD thesis, University of California, 2011.
- [Mic12] Microsoft. <http://msdn.microsoft.com/en-us/library/windows/desktop/aa383745%28v=vs.85%29.aspx>, 2012.
- [Mit97] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [ML00] D. Murray and James J. Little. Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*, 8(2):161–171, 2000.
- [Mur00] R. Murphy. *Introduction to AI Robotics*. MIT Press, 2000.
- [N1111] In-situ exploration and sample return: Autonomous planetary mobility. http://marsrover.nasa.gov/technology/is_autonomous_mobility.html, 2011.
- [Pri12] S. Prince. *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, 2012.
- [RC11] <http://www.reallycooltoys.com/robotics-rcrov/rev9-001.html>, 2011.
- [RN03] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Pearson Education, 2003.

- [Roj03] R. Rojas. Networks of width one are universal classifiers. *Proceedings of the International Joint Conference on Neural Networks*, 4:3124–3127, 2003.
- [RS110] Rs-232 vs. ttl serial communication. <http://www.sparkfun.com/tutorials/215>, 2010.
- [Sal] P. Salvini. The robot dustcart. *Robotics & Automation Magazine, IEEE*, pages 59–67.
- [SB08] Hlavac V. Sonka, M. and R. Boyle. *Image Processing, Analysis, and Machine Vision*. Tompson, 2008.
- [SB12] N. Sharmin and R. Brad. Optimal filter estimation for lucas-kanade optical flow. *Sensors*, 12(9):12694–12709, 2012.
- [Sch] R. Schapire. Foundations of machine learning.
- [Smi97] Stephen Smith. Reviews of optic flow, motion segmentation, edge finding and corner finding, 1997.
- [SRF] Srf04 - ultra-sonic ranger technical specification.
- [SS01] L. Shapiro and G Stockman. *Computer Vision*. Prentice Hall, 2001.
- [ST94] J. Shi and G. Tomasi. Good features to track. *9th IEEE Conference on Computer Vision and Pattern Recognition*, 1994.
- [SY08] Driemeyer J. Saxena, A. and Andrew Y. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 2:157–173, 2008.
- [Sze11] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2011.
- [VC09] M. Vázquez and Chang C. Real-time video smoothing for small rc helicopters. In *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, 2009.
- [XBE12] <http://arduino.cc/en/Main/ArduinoXBeeShield>, 2012.

Apéndice A

Detalles de la Clasificación de Píxeles Borde

En esta sección se describe con mayor detalle el proceso de clasificación de un píxel como borde utilizado por el algoritmo Canny [Can86]. Inicialmente para cada imagen I , se posee un Gradiente G , que se calcula de la siguiente forma:

$$G = \sqrt{G_x^2 + G_y^2} \quad (\text{A.1})$$

Donde G_x y G_y son las derivadas de la imagen según los ejes x y y . En otras palabras, son las derivadas horizontales y verticales de la imagen. Seguidamente se obtiene la dirección de cada píxel θ , a través de los datos calculados anteriormente:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \quad (\text{A.2})$$

Una vez obtenido el ángulo, se procede a clasificar los píxeles en una de cinco categorías, ilustradas en la Figura .

1. **A:** Entran en esta categoría, todos aquellos píxeles, cuyo ángulo θ esté dentro del rango

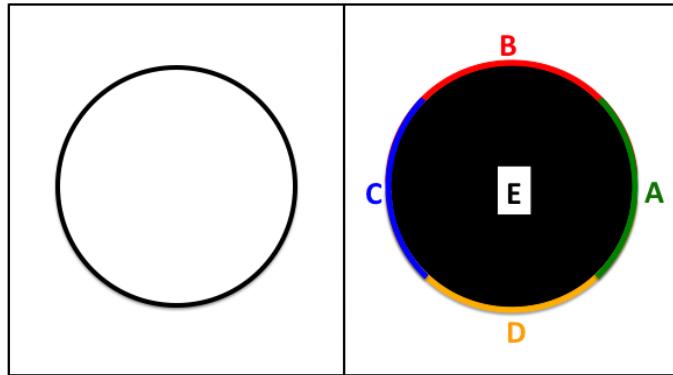


Figura A.1: La imagen muestra cómo serían clasificados los píxeles en una imagen de un círculo. La figura de la izquierda muestra el dibujo original. La figura de la derecha muestra la imagen con los píxeles clasificados

$(1\text{deg}, 45\text{deg}), (316\text{deg}, 359\text{deg})$.

2. **B:** Entran en esta categoría, todos aquellos píxeles, cuyo ángulo θ esté dentro del rango $(46\text{deg}, 135\text{deg})$.
3. **C:** Entran en esta categoría, todos aquellos píxeles, cuyo ángulo θ esté dentro del rango $(36\text{deg}, 225\text{deg})$.
4. **D:** Entran en esta categoría, todos aquellos píxeles, cuyo ángulo θ esté dentro del rango $(226\text{deg}, 315\text{deg})$.
5. **E:** Todos aquellos píxeles cuyo gradiente sea cero grados.

Una vez clasificados los píxeles, se trabaja con dos umbrales, uno alto y uno bajo, que llamaremos T_a y T_b . Todos los píxeles cuyo gradiente sea mayor al del umbral T_a serán considerados bordes definitivos. Una vez obtenidos estos puntos, se buscan todos los píxeles cercanos que posean su misma clasificación direccional. De este grupo se descartan todos aquellos que no superen el umbral bajo T_b . Los píxeles que no fueron eliminados durante este proceso se consideran bordes.

Apéndice B

Características de la Comunicación Serial

En este apéndice se presentan detalles adicionales acerca de las conexiones seriales. El Cuadro B.2 muestra el significado de cada bit en un mensaje serial. Por otro lado, el Cuadro B.1 describe el uso y de los pines más importantes en el hardware de la conexión serial.

BIT	Descripción
Bit de Inicio	Mientras no se estén enviando datos, este bit permanece en el estado lógico “1”. Cuando se envía un dato, este bit se coloca en cero y es prefijo de todo el mensaje.
Bits de Datos	Son el propósito principal de las comunicaciones seriales. El número de bits que se envía en serie para un mensaje puede variar, pero se usan 8 en general.
Paridad	Diseñado para soporte en la detección de errores. Existen dos tipos de paridad : par e impar. En el primer caso, se cuenta el número de bits con valor “1” en el mensaje. El bit de paridad debe afirmar si la secuencia recibida es tiene una cantidad par de “1”. En el segundo caso, se mide si el mensaje tiene una cantidad impar de “1”.
Bit de Parada	Esto, más que un bit, es un acuerdo de retornar la señal lógica “1”, cuando se termina de enviar un mensaje.

Cuadro B.1: Tipos de Bits estándar en el envío de mensajes seriales

PIN	Descripción
DCD	Encendido mientras los dispositivos que se estén comunicando permanezcan conectados.
RX	Junto con el pin TX, son el corazón de la comunicación serial. A través de este cableado se recibirá, en serie, un conjunto de señales de voltaje positivo y negativo que describirán el mensaje que se está recibiendo.
TX	Es el pin de transmisión de datos. Es el reverso del RX
DTR	Indica si el terminal está listo para enviar y recibir datos.
GND	Pin utilizado para generar una tierra común entre los dispositivos que van a transmitir datos seriales.
DSR	Es la contraparte del DTR. Una vez que se envía el DTR, se debería activar el DSR. Estos dos pines son utilizados para hacer <i>handshake</i> en muchas oportunidades.
RTS	Este pin forma parte del control de flujo a nivel de hardware. Cuando está encendido, el dispositivo indica que está listo para recibir más datos.
CTS	Cuando está encendido, se le permite al terminal enviar datos. ES la contraparte del pin RTS.

Cuadro B.2: Pines del estándar RS-232

Apéndice C

Detalles Técnicos del RCRover 9-18

El robot RCRover 9-18 es un modelo del tipo UGV [RC11] (*Unmanned Ground Vehicle*), cuyas siglas significan Vehículo Terrestre no Tripulado. Como lo dice su nombre, los equipos UGV funcionan sin un operador dentro del vehículo. Estos están diseñados con el propósito de proteger al humano de cualquier tarea peligrosa que deba realizarse con ellos. El RCRover 9-18, tiene las siguientes aplicaciones sugeridas:

- Vigilancia y seguridad.
- Búsqueda y rescate.
- Estudios geológicos y estructurales.
- Inspección de superficies contaminadas
- Desarrollo robótico académico.
- Automatización de instalaciones.
- Asistencia personal.

En en Cuadro C.1, se presenta en detalle las características del modelo RCRover 9-18.

Característica	Descripción
Material	Chasis de aluminio cortado con láser y soldado con tecnología aeroespacial.
Dimensiones del Chasis	Largo: 18 pulgadas. Ancho: 12 pulgadas. Alto: 4 pulgadas.
Resistencia	Puede resistir momentáneamente a: impactos, nieve, lluvia, picos de voltaje externos y solventes. Adicionalmente trabaja bien en ambientes inhóspitos y con materiales peligrosos.
Movimiento	Rotación de 50°en adelante. Puede manejar escalones de hasta 6 pulgadas.
Motores y Ruedas	Motores del tipo <i>Large Super Torque Planetary Precision</i> . Tracción en las cuatro ruedas. Sistema de manejo con dos orugas de 6× 4 pulgadas con superficie anticorrosiva y antirresbalante.
Velocidad	De pocas pulgadas por minuto a 7 Millas por hora.
Fuerza	Puede llevar hasta 200 libras de peso.
Brazo	Brazo de 14 pulgadas con garra de 4 pulgadas. Puede levantar de 3 a 8 Libras.
Teleoperación	Viene con un sistema de control RFO (<i>Ready for Operation</i>), listo para ser operado, con sistema de video.
Cámara	Cámara CCD de alta resolución, en color e infrarrojo (Día/Noche) con movimiento horizontal y vertical.

Cuadro C.1: Detalles Técnicos del RC Rover 9-18

Apéndice D

Equipos de Captura de video Revisados

En este apéndice se describen los equipos que fueron considerados para capturar el video analógico recibido por el Kit de control del RCRover. A continuación se exponen sus características y las desventajas encontradas.

Nombre: Turtle Beach Video Advantage USB

Características:

- Entrada de video por RCA.
- Salida del video por USB 2.0.
- Transforma el video a formato DV.
- Permite captura de video en tiempo real.

Desventajas:

- No permite la manipulación del video hasta que se detenga la grabación.
- Necesita de una computadora poderosa para evitar la pérdida de algún cuadro.

Nombre: Pinnacle Studio MovieBox

Características:

- Entrada de video por RCA
- Salida de video por USB
- Transforma el video a formato DVD o VCD

Desventajas:

- El video sólo puede ser manipulado mediante software privativo: *Pinnacle Studio 8.6*.
- Sólo puede utilizarse con los sistemas operativos *Windows*.
- El video no puede verse en tiempo real.

Nombre: Pinnacle Studio MovieBox 2

Características:

- Entrada de video por RCA, S-video y Firewire (IEEE 1934).
- Salida de video por Firewire
- Transforma el video a formato DV, HDV, Digital8, AVCHD, HD-DVD, AVI, DivX, MPEG-2 y MPEG-4.
- Permite captura de video en tiempo real.

Desventajas:

- Sólo puede utilizarse con los sistemas operativos *Windows*.
- El software de captura es privativo.

Nombre: HKWorld USB Analog Video Capture Device

Características:

- Entrada de video por RCA y S-Video.
- Salida de video por USB 2.0.
- Transforma el video a formato DVD .

Desventajas:

- Sólo puede utilizarse con los sistemas operativos *Windows*.

- El video no puede verse en tiempo real.

Nombre: AVID Pinnacle Platinum HD USB Video Capture Device

Características:

- Entrada de video por RCA
- Salida de video por USB
- Transforma el video a formato DVD, AVCHD, DVD, MPEG-4, 3GP, AVI, HDV, MPEG-1, SVCD, WMV, DivX, DV y MPEG-.2

Desventajas:

- Sólo puede utilizarse con el sistemas operativos *Windows*.
- Requiere tarjetas de video compatibles con DirectX9 o 10.

Nombre: Pinnacle Video Capture Device Studio MovieBox DV IEEE 1394 Interface

Características:

- Entrada de video por RCA, S-video y Firewire (IEEE 1934).
- Salida de video por RCA, S-video y Firewire.
- Permite captura y edición de video.
- Captura a 30 y 60 cuadros por segundo.

Desventajas:

- El video sólo puede ser manipulado mediante software privativo: *Studio 8*.
- Sólo puede utilizarse con el sistemas operativos *Windows*.

Nombre: ADS Tech PYRO A/V Link Video Device API-558-EFS IEEE 1394 Interface

Características:

- Entrada de video por RCA, S-VHS, Componente y Firewire (IEEE 1934).

- Salida de video por RCA, S-Video y Firewire de seis pines.
- Permite captura y edición de videos.
- Funciona en sistemas operativos *Mac* y *Windows*

Desventajas:

- Requiere tarjetas de video compatibles con DirectX9 o 10.
- Sólo puede ser capturado por software privativo: Final Cut pro, iMovie y PYRO A/V.

Nombre: Diamond USB HD Video Capture Device

Características:

- Entrada de video por Componente, RCA y S-video.
- Salida de video por USB.
- Transforma el video a formato MPEG-2, AAC-LC y DVD.

Desventajas:

- Sólo puede ser capturado por software privativo.
- Sólo puede utilizarse con el sistemas operativos *Windows*.

Nombre: Canopus ADVC 110

Características:

- Entrada de video por Firewire de 4 y 6 pines, RCA y S-video.
- Salida de video Firewire de 4 y 6 pines.
- No necesita de drivers para ser instalado.
- Puede ser capturado por software privativo y libre.
- Funciona con los sistemas operativos *Windows*, *Linux* y *Mac OsX*.

Desventajas:

- Está probado sólo en algunas versiones de Linux.

Apéndice E

Librerías de Captura Revisadas

En este apéndice se presenta una lista de las librerías y ambientes que fueron considerados como software de captura y procesamiento del video que transmite el RCRover. A continuación se presentan sus procesos de prueba y los resultados obtenidos. Estos están divididos en dos etapas: antes y después de descubrir que el problema era inherente a los sistemas de 64bits. Las librerías con resultados positivos se muestran subrayadas.

Previo al descubrimiento del problema con la Arquitectura de 64 bits

Nombre: Gstreamer

Descripción: Disponible para sistemas operativos Unix. Es un *Framework* que permite la manipulación de elementos multimedia. Posee un plugin que permite la detección de dispositivos de video conectados por Firewire.

Prueba: Se implementó un programa que listara los dispositivos conectados por Firewire.

Resultado: El software no fue capaz de detectar el equipo de captura de video

Nombre: Libdc1394

Descripción: Api de captura de video para dispositivos conectados por Firewire, que permite la manipulación de la entrada a bajo nivel. Funciona en *Windows, Mac OsX* y *Linux*.

Prueba: Se utilizó el programa de ejemplo que provee la documentación. Este debe listar todos los elementos conectados por firewire.

Resultado: La librería no fue capaz de detectar el equipo de captura de video

Nombre: PyDc1394

Descripción: Api de captura de video para dispositivos conectados por Firewire, hecho en python, basado en la librería Libdc1934

Prueba: Se implementó un programa que listara todos los dispositivos detectados por Firewire y mostrara la imagen recibida

Resultado: La librería no fue capaz de detectar el equipo de captura de video.

Nombre: Libraw1394

Descripción: Librería de bajo nivel, especial para captura de videos transmitidos por la conexión con Firewire.

Prueba: Debido a la falta de documentación y ejemplos, no se pudo implementar un programa con las funciones de captura que no produjera errores.

Resultado: No se pudo implementar un programa en esta librería

Nombre: ROS

Descripción: Librería especial para la programación de robótica. La implementación se hace por módulos que se comunican entre sí.

Prueba: Se creó un módulo que capturara el video proveniente del robot.

Resultado: El video recibido consistía en cuadros negros.

Nombre: OpenCV2

Descripción: Es la librería de visión de computadoras más completa dentro del software libre. Posee funciones de captura, filtrado y detección de patrones, entre muchas otras características.

Prueba: Se implementó un programa que capturara cualquier dispositivo adicional a la *WebCam* de la computadora, y mostrara la imagen recibida por pantalla

Resultado: El programa no detectaba dispositivos conectados.

Nombre: Processing

Descripción: Ambiente de programación especialmente diseñado para trabajar con contextos visuales, de modo que posee funciones de captura, generación de imágenes e interacción.

Prueba: Se implementó un programa que capturara cualquier dispositivo adicional a la *WebCam* de la computadora, y mostrara la imagen recibida por pantalla.

Resultado: El programa detectaba y mostraba satisfactoriamente el video capturado. Posteriormente se descubrió que esto se debe a que Processing está implementado bajo una arquitectura de 32bits.

Posterior al descubrimiento del problema con la Arquitectura de 64 bits

Nombre: OpenCV1

Descripción: Es la primera versión de librería de visión de computadoras OPENCV. Posee funciones de captura, filtrado y un tipo de aprendizaje de máquinas propio de la librería llamado Clasificador *Haar Cascade*.

Prueba: Se implementó un programa que capturara cualquier dispositivo adicional a la *WebCam* de la computadora, y mostrara la imagen recibida por pantalla. Se trabajó con una versión precompilada de la librería en 32 bits.

Resultado: El programa detectaba y mostraba satisfactoriamente el video capturado.

Nombre: OpenCV2 y ROS (en 32 bits)

Prueba: Se buscó re-compilar ambas librerías en 32bits, en una máquina con hardware de 64bits.

Resultado: Ambas librerías poseen dependencias que son inherentes a la arquitectura, de modo que falló la compilación al no encontrarse las herramientas en 32 bits requeridas. Estas últimas no eran instalables en una máquina de 64 bits.

Apéndice F

Documentacion de la Librería RoverSerial

Miembros Privados de la Clase

data

Tipo: unsigned char*.

Descripción: Estructura de siete bytes de tamaño en la que se almacena el conjunto completo de comandos al robot.

light

Tipo: boolean.

Descripción: Bandera que indica si la linterna del robot va encendida o no.

serial_port

Tipo: DirecsSerial.

Descripción: Puerto de la Clase DirecsSerial.

portDescriptor

Tipo: int.

Descripción: numero que sirve como identificador del buffer del puerto.

Miembros Públicos de la Clase

int openPORT()

Parámetros: Ninguno.

Descripción: abre el puerto miembro de la clase *port*.

Salida: Devuelve un número negativo en el caso de un error en la apertura.

int sendDATA()

Parámetros: Ninguno.

Descripción: envía por el puerto un mensaje con la información guardada en *data*.

Salida: Ninguna.

void setDATA(unsigned char move, unsigned char rotate, unsigned char arm, unsigned char claw, unsigned char pan, unsigned char tilt, bool light)

Parámetros:

move: velocidad del movimiento hacia adelante.

rotate: velocidad de rotación,

arm: posición del brazo.

claw: apertura de la garra.

pan: posición horizontal de la cámara.

tilt: posición vertical de la cámara.

Descripción: Permite modificar todos los datos del mensaje almacenados en *data*.

Salida: Ninguna.

void setMOVE(unsigned char move)

Parámetros:

move: velocidad del movimiento hacia adelante.

Descripción: coloca en *data* la información de la velocidad del movimiento hacia adelante.

Salida: Ninguna.

void setTURN(unsigned char turn)

Parámetros:

turn: velocidad de giro.

Descripción: coloca en *data* la información de la velocidad del giro.

Salida: Ninguna.

void setARM(unsigned char arm)

Parámetros:

arm: posición del brazo.

Descripción: coloca en *data* la información de posición del brazo.

Salida: Ninguna.

void setHAND(unsigned char hand)

Parámetros:

hand: Apertura de la garra.

Descripción: coloca en *data* la información de la apertura de la garra.

Salida: Ninguna.

void setPAN(unsigned char pan)

Parámetros:

pan: Posición horizontal de la cámara.

Descripción: coloca en *data* la información de posición horizontal de la cámara.

Salida: Ninguna.

void setTILT(unsigned char tilt)

Parámetros:

tilt: Posición vertical de la cámara.

Descripción: coloca en *data* la información de posición vertical de la cámara.

Salida: Ninguna.

void setLIGHT(bool tilt)

Parámetros:

light: booleano que indica si la luz de la linterna está encendida o no.

Descripción: coloca en *data* la información de la linterna.

Salida: Ninguna.