



UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

INGENIERÍA ELECTRÓNICA INDUSTRIAL Y
AUTOMÁTICA

SURFACE MINES DETECTION USING
VISUAL INFORMATION

Autor: Daniel Fernández Delgado

Tutor: Abdulla Hussein Abdulrahman Al-Kaff

RESUMEN

Las guerras han dejado atrás más de 100 millones de minas terrestres activas por todo el mundo, causando aproximadamente una víctima cada 20 minutos. En este proyecto se busca una solución para este problema, aplicando tecnologías emergentes como son los drones autónomos y la visión artificial para la detección y neutralización de estas armas.

Se emplearán los métodos más utilizados en el entorno de la detección de objetos mediante visión artificial, como son las detecciones por color y la detección basada en la mezcla de histogramas de gradientes (HOG) y máquinas de soporte vectorial (SVM). Ambos métodos se combinan para obtener una detección fiable de las minas, optimizándose para el entorno en el que se realizarán las principales pruebas. Estos algoritmos serán implementados como un paquete en el “Sistema Operativo Robótico”, más conocido como ROS, para su posterior inclusión en un dron autónomo.

Los resultados de estas detecciones serán comprobados mediante una fusión de métodos técnicos, como las matrices de fusión, y con métodos más subjetivos como la propia observación visual de los videos obtenidos.

Finalmente, se concluirá indicando qué tipo de detectores son los más idóneos para el proceso detallado, y se proporcionarán archivos ya entrenados capaces de realizar las detecciones de una forma fiable.

ABSTRACT

Past wars have left behind more than a hundred million active landmines all around the world, causing approximately one victim every twenty minutes. This project aims to find a solution for that issue by using emergent technologies such as autonomous drones and computer vision for the detection and neutralization of those weapons.

This purpose will be approached by using the main methods used in computer vision, as detections based on colors and detections based on the mix of “Histogram of Gradients” (HOG) and “Support Vector Machines” (SVM). These two methods will be combined to obtain a trustworthy detection of the mines, being optimized for the environment in which the main tests will be set. These algorithms will be implemented in a “Robotic Operative System” (ROS) package, for its later incorporation in an autonomous drone.

The results of this detections will be verified by using a merge of technical methods, like confusion matrix, and more subjective methods as visual observation of the obtained videos.

Finally, this project will conclude by indicating the type of detectors that is most suitable for the detailed process, and, furthermore, trained detectors ready to detect mines reliably will be provided.

ÍNDICE DE CONTENIDO

1	INTRODUCCIÓN	13
1.1	Minas antipersona en el mundo	13
1.2	Movimiento antiminas	15
1.3	Competición Minesweepers.....	16
2	ESTADO DEL ARTE	19
2.1	Drones y su uso en el ámbito humanitario.....	19
3	METODOLOGÍA	24
3.1	Introducción	24
3.2	Descripción del software	25
3.2.1	OpenCV	25
3.2.2	ROS	26
4	DESARROLLO DEL ALGORITMO	28
4.1	Fundamentos teóricos	28
4.1.1	Histograma de gradients orientado (HOG).....	28
4.1.2	Máquinas de soporte vectorial (SVM).....	31
4.1.3	Espacios de color.....	34
4.1.3.1	RGB	34
4.1.3.2	HSV	35
4.1.3.3	YCrCb	37
4.1.3.4	Lab	38
4.1.4	Reducción de ruido.....	39
4.1.4.1	Filtro mediana	39
4.1.4.2	Transformaciones morfológicas.....	39
4.1.4.2.1	Erosión	40
4.1.4.2.2	Dilatación	41
4.1.4.2.3	Opening	42
4.1.5	Detector de bordes de Canny.....	42
4.2	Descripción del algoritmo.....	43
4.2.1	Detección mediante SVM+HOG.....	44
4.2.1.1	Creación de la base de datos	44
4.2.1.2	Matriz de entrenamiento	48

4.2.1.2.1	Inicialización	49
4.2.1.2.2	Entrenamiento	50
4.2.1.3	Detección	55
4.2.1.3.1	Carga del SVM.....	55
4.2.1.3.2	Algoritmo de detección	57
4.2.2	Detección mediante color	61
4.2.2.1	Conversión del espacio de color	61
4.2.2.2	Thresholding	65
4.2.2.3	Reducción de ruido	66
4.2.2.4	Canny	68
4.2.2.5	Detección de contornos.....	68
4.2.2.6	Puntos medios	69
4.2.3	Detecciones finales	69
4.2.4	Implementación en ROS.....	70
4.2.4.1	Configuración inicial	70
4.2.4.2	Compilación y ejecución	72
5	RESULTADOS.....	74
5.1	Matriz de confusión	74
5.2	Análisis de resultados	76
5.2.1	Pruebas 1-6	77
5.2.1.1	Prueba 1	77
5.2.1.2	Prueba 2	78
5.2.1.3	Prueba 3	79
5.2.1.4	Prueba 4	80
5.2.1.5	Prueba 5	81
5.2.1.6	Prueba 6	82
5.2.2	Pruebas 7 – 10	83
5.2.2.1	Prueba 7	83
5.2.2.2	Prueba 8	84
5.2.2.3	Prueba 9	85
5.2.2.4	Prueba 10	86
5.2.3	Pruebas finales.....	87
5.2.3.1	Prueba 11	87

5.2.3.2	Prueba 12	88
6	CONCLUSIONES Y TRABAJOS FUTUROS	90
6.1	Conclusiones	90
6.2	Trabajos futuros	91
7	BIBLIOGRAFÍA	93

ÍNDICE DE FIGURAS

Fig 1.1: Mina Terrestre	13
Fig 1.2: Presencia de Minas en el mundo	14
Fig 1.3: Logotipo de la organización "Campaña internacional para la prohibición de minas terrestres"	15
Fig 1.4: Logotipo de la competición "Minesweepers"	16
Fig 1.5: Primera edición de la competición "Minesweepers"	17
Fig 1.6: Formato utilizado en la detección de minas de la competición "Minesweepers"	18
Fig 2.1: "Aerial Target", primer drón de la historia	19
Fig 2.2: Estructura física de un UAV (Wikipedia)	20
Fig 2.3: Estructura de control de un UAV	20
Fig 2.4: Fases de detección del dron "Mine-Kafon"	23
Fig 3.1: Estructura del proceso de detección de minas	24
Fig 3.2: Logotipo de la librería de visión artificial "OpenCV"	25
Fig 3.3: Logotipo del Sistema Operativo Robótico, ROS	26
Fig 3.4: Estructura de ROS	26
Fig 4.1: Estructura de las máscaras en un proceso de HOG: píxeles, celdas y bloques ..	28
Fig 4.2: Procesos del "Histogram of Gradients", HOG	29
Fig 4.3: Ejemplo gráfico del resultado de un HOG	30
Fig 4.4: Visualización de las opraciones realizadas en un HOG	30
Fig 4.5: Fórmula para obtener el tamaño final del vector obtenido tras realizar un HOG	31
Fig 4.6: Gráfico 2D representativo de una selección por SVM	32
Fig 4.7: Gráficos 2D y 3D representativos de una selección por SVM	32
Fig 4.8: Fórmula del hiperplano de un SVM	33
Fig 4.9: Fórmula del filtro kernel	33
Fig 4.10: Formula final de un SVM tras aplicar un filtro kernel para añadir un plano..	33
Fig 4.11: Ecuación kernel lineal	33
Fig 4.12: Ecuación kernel polinómico	33
Fig 4.13: Ecuación kernel gaussiano	33
Fig 4.14: Ecuación kernel sigmoidal	34
Fig 4.15: Representación en 3D del espacio de color RGB	34
Fig 4.16: Separación de los canales de color R-G-B en distintas condiciones de iluminación	35
Fig 4.17: Representación 3D del espacio de color HSV	35
Fig 4.18: Separación de los canales de color H-S-V en distintas condiciones de iluminación	36
Fig 4.19: Representación 2D del espacio de color YCrCb	37
Fig 4.20: Separación de los canales de color Y-Cr-Cb en distintas condiciones de iluminación	37

Fig 4.21: Representación en 3D del espacio de color Lab	38
Fig 4.22: Separación de los canales de color L-A-B en distintas condiciones de iluminación	38
Fig 4.23: Visualización matemática de un filtro de mediana (Clase 6 - Sistemas de percepción (UC3M)).....	39
Fig 4.24: Ejemplo de un proceso de erosión	40
Fig 4.25: Aspecto de un filtro kernel unitario de tamaño 3x3.....	40
Fig 4.26: Visualización matemática de un proceso de erosión.....	40
Fig 4.27: Ejemplo de un proceso de dilatación	41
Fig 4.28: Visualización matemática de un proceso de dilatación.....	41
Fig 4.29: Ejemplo de un proceso de opening	42
Fig 4.30: Representación gráfica de una selección de bordes mediante canny	43
Fig 4.31: Procesos de la detección con SVM y HOG	44
Fig 4.32: Estructura del archivo samplesCreation.cpp	45
Fig 4.33: Ejecución del archivo samplesCreation.cpp	46
Fig 4.34: Desglose de la función "selectROIs" de la librería OpenCV	46
Fig 4.35: Selección de una región de interés con la función "selectROIs"	46
Fig 4.36: Aspecto del código para la adquisición de ROIs a partir de un archivo de video	47
Fig 4.37: Recorte original	48
Fig 4.38: Recorte convertido a formato 32x32 en escala de grises	48
Fig 4.39: Estructura del archivo SVMTraining.cpp	49
Fig 4.40: Código en formato "CommandLineParser" para introducir los datos iniciales del entrenamiento	49
Fig 4.41: Desglose de la clase "HOGDescriptor" de la librería OpenCV	51
Fig 4.42: Desglose de la función "compute", dentro de la clase HOG de la librería OpenCV	51
Fig 4.43: Tipos de SVM (SVM - docs.opencv.org)	52
Fig 4.44: Tipos de kernel en un objeto SVM (SVM - docs.opencv.org).....	53
Fig 4.45: Llamada a la función "train" en el código	53
Fig 4.46: Output en la consola tras la ejecución del entrenamiento mediante la función "train".....	54
Fig 4.47: Código de ejemplo del primer entrenamiento	54
Fig 4.48: Código de ejemplo del segundo entrenamiento	54
Fig 4.49: Código de ejemplo del tercer entrenamiento	55
Fig 4.50: Procesos en la carga de un archivo SVM.....	55
Fig 4.51: Código de la función getSVM()	57
Fig 4.52: Desglose de la función "detectMultiScale" dentro de la clase HOGDescriptor de la librería OpenCV	57
Fig 4.53: Representación de un "scale"	58
Fig 4.54: Aplicación de "Non-maximum Supresión" en una detección por HOG	59
Fig 4.55: Resultado de una detección por SVM+HOG (1)	59
Fig 4.56: Resultado de una detección por SVM+HOG (2)	60
Fig 4.57: Resultado de una detección por SVM+HOG (3)	60

Fig 4.58: Procesos en una detección mediante color	61
Fig 4.59: Obtención de los valores RGB de un color	62
Fig 4.60: Código de la conversión de Vec3b a Mat3b	62
Fig 4.61: Código de las conversiones de color	63
Fig 4.62: Fórmula matemática de la conversión de RGB a HSV	63
Fig 4.63: Fórmula matemática de la conversión de RGB a YCbCr	64
Fig 4.64: Fórmula matemática de la conversión de RGB a LAB	64
Fig 4.65: Código de la conversión de Mat3b a Vec3b	65
Fig 4.66: Desglose de la función "inRange" de la librería OpenCV	65
Fig 4.67: Imagenes binarias. La superior es la obtenida a partir de una imagen HSV, la inferior a partir de YCrCb	66
Fig 4.68: Desglose de la función "medianBlur" de la librería OpenCV	67
Fig 4.69: Ejecución de un "opening" en código	67
Fig 4.70: Resultado del opening	67
Fig 4.71: Desglose de la función "Canny" de la librería OpenCV	68
Fig 4.72: Resultado del Canny	68
Fig 4.73: Desglose de la función "findContours" de la librería OpenCV	68
Fig 4.74: Procesos en la detección de centros	69
Fig 4.75: Desglose de la función "moments" de la librería OpenCV	69
Fig 4.76: Código de la detección de centros.....	69
Fig 4.77: Compilación y acceso al pkg de ROS	70
Fig 4.78: Directorio de los archivos .cpp.....	71
Fig 4.79: Directorio del archivo CMakeList.txt	71
Fig 4.80: Configuración del archivo CMakeList.txt	71
Fig 4.81: Inicialización de "roscore"	72
Fig 4.82: Código para la ejecución del programa de ROS	72
Fig 4.83: Resultado final de la detección utilizando ROS.....	73
Fig 5.1: Fórmula para obtener la exactitud.....	75
Fig 5.2: Fórmula para obtener el ratio de positivos	75
Fig 5.3: Fórmula para obtener el ratio de falsos positivos.....	75
Fig 5.4: Fórmula para obtener el ratio de negativos	75
Fig 5.5: Fórmula para obtener el ratio de falsos negativos.....	76
Fig 5.6: Fórmula para obtener la precisión.....	76
Fig 5.7: Detecciones para el detector 1	77
Fig 5.8: Imagen final detector 1	78
Fig 5.9: Resultados para el detector 2.....	78
Fig 5.10: Imagen final detector 2	79
Fig 5.11: Resultados para el detector 3.....	79
Fig 5.12: Imagen final detector 3	80
Fig 5.13: Resultados para el detector 4.....	80
Fig 5.14: Imagen final detector 4	81
Fig 5.15: Resultados para el detector 5.....	81
Fig 5.16: Resultados para el detector 6.....	82
Fig 5.17: Resultados para el detector 7.....	83

Fig 5.18: Imagen final detector 7	83
Fig 5.19: Resultados para el detector 8.....	84
Fig 5.20: Imagen final detector 8	84
Fig 5.21: Resultados para el detector 9.....	85
Fig 5.22: Imagen final detector 9	85
Fig 5.23: Resultados para el detector 10.....	86
Fig 5.24: Imagen final detector 10	86
Fig 5.25:Resultados para el detector 11.....	87
Fig 5.26: Imagen final, estándar y con HSV, para el detector 11.....	88
Fig 5.27: Resultados para el detector 12.....	89
Fig 5.28: Imagen final, estándar y HSV, para el detector 12	89

INDICE DE TABLAS

Tabla 1: Estructura de una matriz de confusión	74
Tabla 2: Detectores entrenados.....	77
Tabla 3: Precisión detector 1	78
Tabla 4: Precisión detector 2	79
Tabla 5: Precisión detector 3	80
Tabla 6: Precisión detector 4	81
Tabla 7: Precisión detector 5	82
Tabla 8: Precisión detector 6	82
Tabla 9: Precisión detector 7	84
Tabla 10: Precisión detector 8	85
Tabla 11: Precisión detector 9	86
Tabla 12: Precisión detector 10	87
Tabla 13: Precisión detector 11	88
Tabla 14: Precisión detector 12	89

1 INTRODUCCIÓN

En la actualidad, la tecnología de enfoque militar avanza a grandes pasos debido a las numerosas inversiones que reciben por sus propios países, interesados en ser pioneros en cualquier avance en el entorno para tomar la delantera frente a sus competidores y poder lucrarse de ello.

Sin embargo, esta inversión económica solo afecta a todo aquello relacionado con tener mayor potencia militar, dejando de lado cualquier tema que no tenga probabilidades de producir beneficios en un futuro.

Así, las tecnologías para la protección de civiles como pueden ser los sistemas de detección de minas, los sistemas de neutralización de estas, ... se han visto estancadas desde hace décadas. Por ello, numerosas organizaciones a favor de los derechos humanos se han visto obligadas a tomar parte en este asunto y fomentar ellos mismos el desarrollo de tecnologías que se encarguen de estos temas abandonados por los gobiernos mundiales.

La competición “Minesweepers: Towards a landmine-free world” surgió en 2012 con el objetivo de dar visibilidad y crear conciencia sobre el problema de las minas terrestres y las UXOs, o municiones no explotadas; fomentando la investigación en robótica y sus aplicaciones. Dentro de esta competición es posible utilizar vehículos terrestres o aéreos no tripulados con el objetivo de detectar, posicionar, ... minas terrestres e UXOs, tanto enterrados como superficiales [1].

1.1 Minas antipersona en el mundo

Las minas antipersona son minas terrestres diseñadas para asesinar o incapacitar a sus víctimas. Son utilizadas como armas en las guerras, siendo lanzadas desde aviones de manera casi aleatoria, provocando que su localización sea muy difícil incluso para los propios manipuladores.



Fig 1.1: Mina Terrestre

Uno de los grandes problemas de estas minas es que están diseñadas para causar heridas en los enemigos (mutilaciones, ...) y se activan con presiones de pesos muy ligeros, por lo que un mínimo contacto produce su detonación. La retirada de forma manual de estas minas es muy peligrosa, por ello es necesario el uso de robots y otros sistemas que aumentan mucho el coste de este proceso.

Juntando el arsenal de minas de países productores como Estados Unidos, Israel, Corea del Norte, India, China, Rusia, Sudáfrica, Singapur, ... se pueden contar entre 180 y 185 millones de minas en todo el mundo. Por otro lado, se calcula que existen más de 110 millones de minas repartidas en más de 64 países, la mayoría de ellos en África.

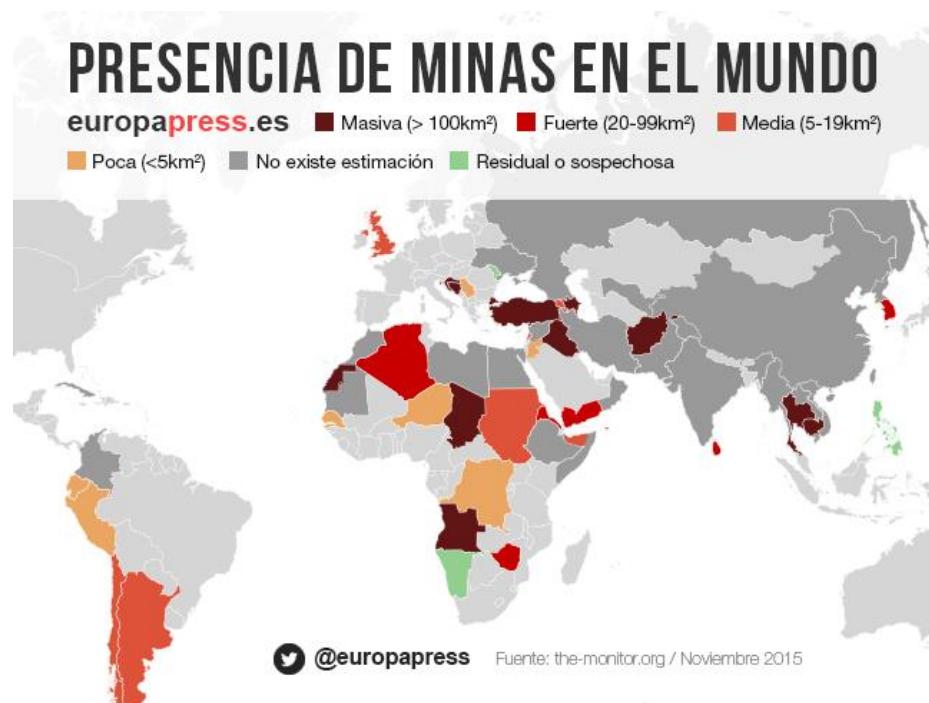


Fig 1.2: Presencia de Minas en el mundo

Los países con más campos minados son Camboya, con 10 millones de minas (uno de cada 236 ciudadanos ha sufrido una herida por mina), Angola con 9 millones de minas (uno de cada 470 habitantes mutilado), Bosnia-Herzegovina, Afganistán, Sudán, Irak, ... y también se ven afectados otros países más desarrollados como El Salvador, Perú, Nicaragua, y Colombia (en 2009 se reportaron 777 casos de víctimas por minas antipersona).

Anualmente, más de 26.000 personas mueren o sufren mutilaciones por detonaciones de minas por todo el mundo, aproximadamente una víctima cada 20 minutos, 10 diarias según los informes de la ICBL (Campaña Internacional para la Prohibición de las Minas

Antipersona). Las minas pueden permanecer activas durante décadas desde que son activadas, por lo que no es necesario que el conflicto bélico sea reciente para que exista peligro por la existencia de minas terrestres. Así, el 90 % de las víctimas por minas antipersona son civiles, un 39% niños; y a su vez su presencia en carreteras, caminos, campos de cultivo,... suele impedir la vuelta a la normalidad de un país tras un conflicto.

El precio de manufacturación tiene un coste inferior a 2€, sin embargo, el proceso de desactivación puede alcanzar un coste incluso superior a 700€. Muchas veces esto es usado por las empresas productoras para beneficiarse económicamente ofreciendo servicios de desminado para sus propios productos [2] [3].

1.2 Movimiento antiminas

En el año 1992, seis organizaciones por los derechos humanos se unieron para formar la “Campaña Internacional para la Prohibición de las Minas Antipersona”, o “International campaign to ban landmines” (ICBL) en inglés.



Fig 1.3: Logotipo de la organización "Campaña internacional para la prohibición de minas terrestres"

En el año 1997 esta organización, que ya estaba formada por más de 1.400 organizaciones, consiguió que tuviese lugar la “Convención sobre la prohibición de minas antipersonales”, o el Tratado de Ottawa, donde 156 países firmaron la prohibición de minas antipersonales, que entró en vigor en 1999. En este tratado cada estado miembro se compromete a:

- Nunca emplear, producir, adquirir, almacenar, ... minas antipersonales.
- Destruir todas las minas en su posesión en un máximo de 4 años desde la entrada en vigor del tratado.

- Destruir todas las minas presentes en zonas de su territorio los 10 años después de la entrada en vigor del tratado para ese estado miembro en concreto.
- Proveer asistencia para este fin a cualquier estado miembro en caso de ser posible.

Gracias a este tratado, la ICBL obtuvo el premio Nobel de la paz en 1997.

En España concretamente se eliminaron entre 1998 y 1999 un total de 820.000 minas antipersona.

El principal problema existente con este tratado es que, aunque se eliminan las minas no utilizadas, los campos minados son muy difíciles de limpiar y requieren de mucho apoyo económico. En 2012 se obtuvo la cifra récord con 681 millones de dólares, y desde entonces esta cifra no ha hecho nada más que reducirse.

Sin embargo, el número de kilómetros cuadrados limpiados de minas al año ha aumentado respecto a anteriores datos (en 2013 se limpiaron 185km cuadrados, en 2014 200km cuadrados). Uno de los principales motivos de esto es la innovación en los métodos de detección, con el uso de robots autónomos, etc.

Aunque se tomaron medidas importantes en contra de estas minas, en la actualidad siguen siendo la causa de miles de heridos y muertos por todo el mundo, mientras que el dinero invertido en su eliminación va disminuyendo.

Las nuevas tecnologías suponen el único futuro posible en esta lucha, facilitando las operaciones de limpieza y disminuyendo su coste [4] [5].

1.3 Competición Minesweepers



Fig 1.4: Logotipo de la competición "Minesweepers"

En el año 2012, Egipto era el país con más minas “activas” del mundo (23.000.000 de minas esparcidas por su territorio). Debido a intereses económicos, el gobierno egipcio declaró como una de sus principales prioridades la descontaminación de zonas como el Golfo de Suez, la costa norte, y las costas del Mar Rojo.

A modo de búsqueda de una solución para este problema, la “German University in Cairo” (GUC) creó la competición “MINESWEEPERS: Towards a Landmine-Free Egypt”. Esta competición comenzó siendo a nivel nacional, y su objetivo era fomentar la investigación y el desarrollo de la robótica para usos humanitarios en la eliminación de las minas de Egipto.



Fig 1.5: Primera edición de la competición "Minesweepers"

Debido al éxito de esta primera edición, la competición ha continuado celebrándose anualmente y pasó a ser de carácter internacional. La edición del año 2018 se celebrará en Madrid, España.

La competición cuenta con 3 categorías:

- Minesweepers Juniors: para estudiantes de primaria e institutos. La competición consistirá tan solo en la detección de objetos metálicos
- Minesweepers Academia: orientado para estudiantes de grado, postgrado e investigadores. Detección y mapeo de objetos metálicos.
- Minesweepers Industria: para compañías y entornos profesionales. Objetos metálicos y no metálicos con diferentes dimensiones y formas. Detección, posicionamiento en el entorno, y mapeado.

La categoría de estudiantes de grado, postgrado e investigadores cuenta con las siguientes características:

La localización será un campo de hierba con un área de 20x20m delimitados por 4 coordenadas de GPS donde existirán desniveles y algunas plantas. Las minas superficiales serán cubos metálicos negros con un tamaño de 10x10x10cm y no estará permitido el contacto con ellas.

El robot ha de ser teledirigido o autónomo, y debe de haber sido construido por los miembros del equipo. Los robots autónomos tendrán un bono de 40% en su puntuación respecto a los robots teledirigidos. Cuando se detecte una mina, el robot deberá reportar su presencia automáticamente mediante el uso de una alarma visual y sonora durante mínimo 2 segundos.

Se deberá introducir la posición de las minas en un mapa de 19x19 cuadrados (cada uno de ellos representando un área de 100x100cm) con la siguiente apariencia:

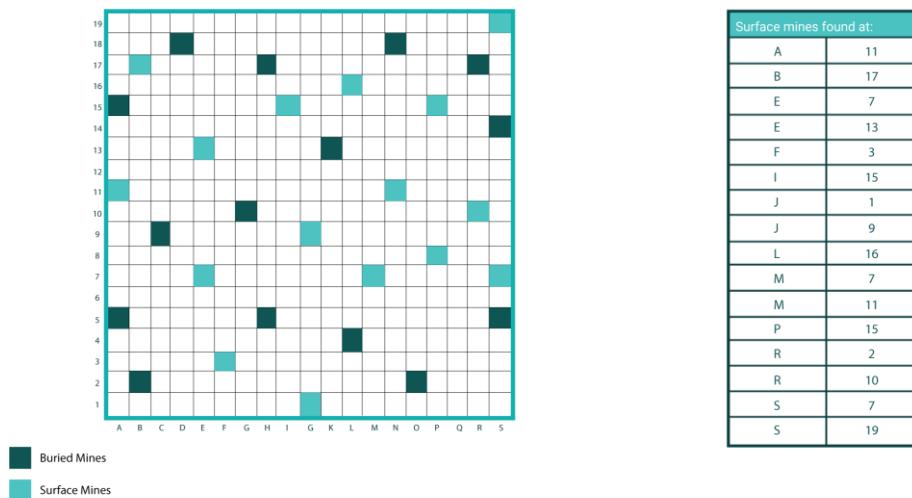


Fig 1.6: Formato utilizado en la detección de minas de la competición "Minesweepers"

Los robots han de comenzar desde la casilla A1, desde ahí se moverán por todo el área de forma libre y deberán detectar, registrar y avisar de la localización de cada mina que encuentre. Solo un miembro del equipo podrá estar dentro del área por si fuese necesario reparar el robot. En total, cada equipo dispondrá de 20 minutos incluyendo el tiempo de reparación, durante el cual se irá penalizando al equipo.

La competición terminará si el robot toca una mina superficial, si existen trampas, si el equipo se rinde, o si el tiempo total pasa.

Los equipos obtendrán un 20% de puntuación extra si se utiliza “ROS”, “Multi robot system”, o “Autonomous robotic system” en el dron [6].

2 ESTADO DEL ARTE

2.1 Drones y su uso en el ámbito humanitario

“Dron” es el nombre que se le da en el ámbito común a los técnicamente conocidos como “Unmanned Aircraft Vehicles” (UAV), o “Vehículo Aéreo no Tripulado” (VANT) en castellano. Estos son vehículos sin tripulación capaces de volar de forma autónoma de una forma controlada y sostenida.

Su aparición se produjo durante la primera guerra mundial y fueron inicialmente utilizados como blancos aéreos de entrenamiento.



Fig 2.1: "Aerial Target", primer dron de la historia

Son de nuevo utilizados como blancos aéreos en la segunda guerra mundial, pero no es hasta finales del siglo XX cuando comienzan a ser operados de forma autónoma mediante radio control. Su gran potencial se demostró de nuevo durante otras guerras, especialmente la guerra de Bosnia y la guerra del Golfo, donde fueron usados por la preocupación de las U.S. Air Force en lo respectivo a perder a sus pilotos sobre territorio enemigo.

Al igual que muchas otras tecnologías importantes, la innovación en el ámbito de los vehículos aéreos no tripulados se vio impulsada por su uso bélico, que hizo que los gobiernos de todo el mundo invirtiesen dinero en su investigación y desarrollo.

Una vez comprobada las grandes ventajas de este tipo de vehículos, se comenzó a explorar su uso en otro tipo de sectores no relacionados con el entorno bélico. Un ejemplo se puede ver en el proyecto “CAPECON” de la unión europea, un proyecto que buscaba el desarrollo de UAV para su uso en aplicaciones civiles [6].

Los UAV's comparten por lo general la misma estructura física:

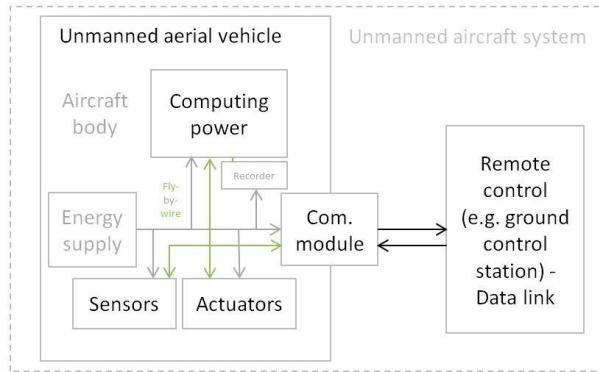


Fig 2.2: Estructura física de un UAV (Wikipedia)

- Energy Supply: por lo general, los UAV's de tamaño pequeño utilizan baterías Li-Po, mientras que otros de mayor tamaño utilizan las ingenierías convencionales usadas en aviones.
- Computing: el hardware utilizado por los vehículos aéreos no tripulados es conocido como "Flight controller" (FC), "Flight controller board" (FCB), o autopiloto.
- Sensors: sensores de posición, de movimiento, de distancia, sensores que proporcionan información sobre el estado interno del dron, etc. Dentro de esta categoría se incluyen otros sensores específicos, como pueden ser las cámaras utilizadas en el reconocimiento por imágenes, los sensores electromagnéticos para detección de minas, ...
- Actuators: desde controles digitales de velocidad, hasta LED's o armas.
- Software: son sistemas de tiempo real que requieren de respuestas rápidas, se usan desde Raspberry Pis, hasta otros procesadores o sistemas diseñados especialmente para controlar drones.

Los drones utilizan por lo general estructuras de control en bucle:

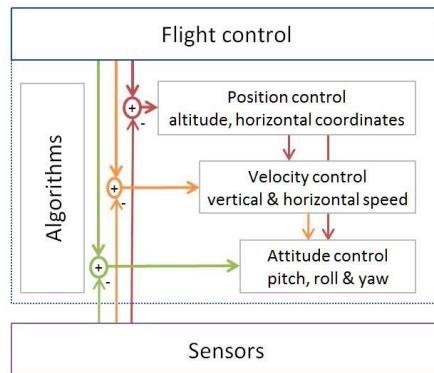


Fig 2.3: Estructura de control de un UAV

Dentro de estas estructuras es posible incluir acciones extra para el uso de drones en diversos ámbitos, como podría ser el uso de actuadores para entregar productos, etc.. [7].

Esta versatilidad ha llevado a que, en la actualidad, los drones sean frecuentemente usados como herramientas en muchos sectores aparte del militar [8]:

- Aeroespacial: los UAVs son utilizados como herramientas de inspección visual en el mantenimiento y creación de vehículos aéreos.
- Uso recreacional: han de cumplir distintas normas dependiendo del país en el que se usen, pero pueden ser usados para fotografía y grabación de videos, para carreras, etc.
- Investigación científica: se usan en áreas en las que el uso de vehículos aéreos tripulados puede suponer un gran riesgo para sus tripulantes, como pueden ser huracanes,...
- Control de especies: para la observación y el control de especies animales, ... protegidas o en peligro de extinción. Pueden servir para monitorearlos y conocer su estado, e incluso para luchar contra la caza furtiva.
- Control de contaminación: drones equipados con sensores de contaminación que facilitan su estudio.
- Arqueología: los drones son capaces de crear mapeados 3D en las que se ven representadas zonas mineras, y otro tipo de datos que pueden ayudar a los arqueólogos.
- Agricultura: aparte de para crear mapeados que ayuden a la expansión de los campos de cultivo, los drones pueden ser usados para la plantación autónoma de algunos tipos de semillas.
- Construcción: permiten reportar errores de construcción en zonas de difícil acceso, e incluso solucionar algunos de estos errores.
- Transporte de pasajeros.

Uno de los sectores en los que más se ha fomentado el uso de drones es el sector humanitario, dentro del cual se pueden distinguir distintos tipos:

- Búsqueda y rescate: UAVs de gran tamaño como los “Predators” fueron usados tras los huracanes ocurridos en Louisiana y Texas (EEUU) en 2008. Estos permitían gracias a sus sensores y radares la toma de imágenes a través de nubes, lluvia, niebla, ... dando a los investigadores datos reales sobre el estado de las zonas devastadas antes y después del huracán.

Otros radares de pequeño tamaño permiten realizar operaciones de este estilo a pequeña escala: en 2014 un hombre de 82 años desaparecido durante 3 días fue encontrado por un UAV. En algunas costas los drones han sido usados por

socorristas para buscar mediante cámaras térmicas a personas en peligro de ahogamiento y proporcionarles equipos salvavidas, su eficacia se comprobó en junio de 2018, cuando un dron salvó la vida a dos adolescentes en Australia.

En 2017 un estudio reveló que al menos 59 personas fueron rescatadas gracias al uso de UAVs.

- Ayuda en desastres: tras la catástrofe de la planta nuclear de Fukushima por el tsunami de marzo de 2011, fueron utilizados drones autónomos para obtener información del estado de la central. Por otra parte, se están realizando otros estudios para utilizar drones en los incendios forestales, etc.
- Transporte de ayudas: En 2013 la compañía DHL utilizó UAVs para enviar medicinas a regiones inaccesibles. Esta solo fue la primera operación “oficial” de este estilo, pero los UAVs se han seguido utilizando con este propósito en numerosas ocasiones.

El uso de UAVs destaca principalmente en operaciones que ponen en peligro la vida humana, un ejemplo claro sería la desactivación de minas.

En 2011 la organización “Find a Better Way” se unió a la Universidad de Bristol para desarrollar drones equipados con tecnología de imágenes hiperespectrales, un tipo de captura de imágenes espectrales capaces de recoger información de espectro electromagnético de la zona fotografiada. Esta tecnología permitía la localización de zonas con restos químicos absorbidos por plantas, etc, causando anomalías en la zona que ayudarían a conocer la localización de minas enterradas. Así, se obtenía un mapeado de la zona con las minas detectadas, facilitando la labor de los trabajadores encargados de su neutralización.

Otro ejemplo de los usos de drones para desminado se observa en la competición “Drones for Good” de 2015, donde una compañía española llamada CATUAV fue finalista con un dron equipado con sensores ópticos utilizado para escanear zonas minadas de Bosnia Herzegovina [9] [10].

En la actualidad, se está desarrollando el proyecto “Mine Kafon” en Holanda, un proyecto que busca la creación de un dron capaz de detectar y detonar rápidamente minas terrestres. El dron utiliza un proceso de tres pasos: mapeado de la zona, detección de las minas, y detonación de estas.

Su funcionamiento es el siguiente: sobrevuela zonas potencialmente peligrosas generando un mapeado en 3D, y utiliza un detector de metal para localizar las minas. Una vez localizadas, el dron deposita sobre las minas un detonador utilizando un brazo robótico, y posteriormente se aleja a una distancia en la que no se ve afectado por la

explosión. Su objetivo es ser 20 veces más rápido y 200 veces más barato que las tecnologías de desminado actuales, y declara que podrá eliminar las minas terrestres de todo el mundo en unos 10 años [11].

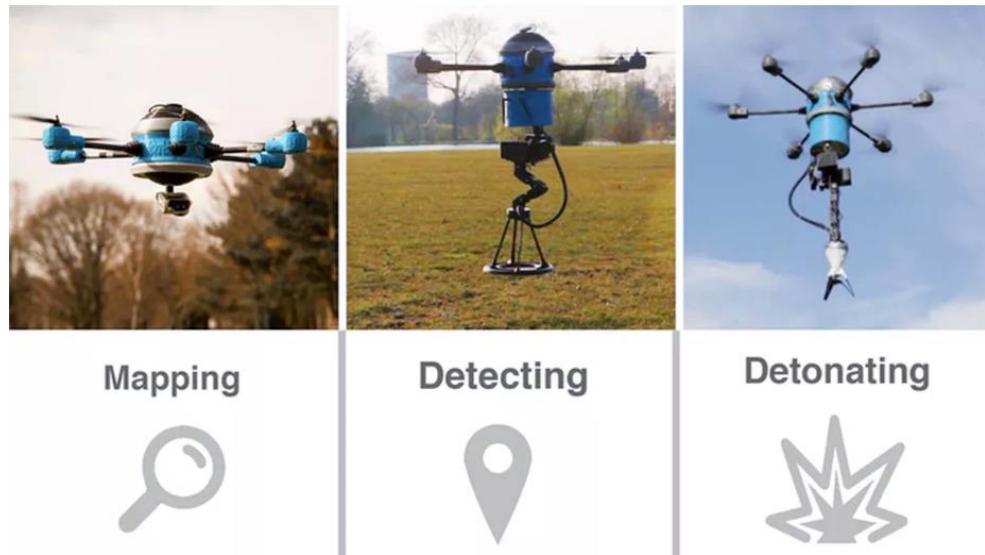


Fig 2.4: Fases de detección del dron "Mine-Kafon"

3 METODOLOGÍA

3.1 Introducción

En este proyecto se busca la realización de un software capaz de detectar, de manera fiable, minas superficiales de un tamaño de 10x10x10 cm y de color negro. Estas minas se situarán sobre una zona herbácea con pocos obstáculos, y podrán ser divisadas desde una posición vertical.

Así, la detección será llevada a cabo mediante el procesamiento de información visual obtenida por una cámara situada en un dron que se moverá de forma autónoma conforme el código determine. Para procesar esta información se utilizará la librería de visión artificial “OpenCV”, se programará en C++, y posteriormente se introducirá este código en ROS.

El software se dividirá en dos procesos iniciales, y posteriormente estos procesos se unificarán para obtener una solución conjunta.

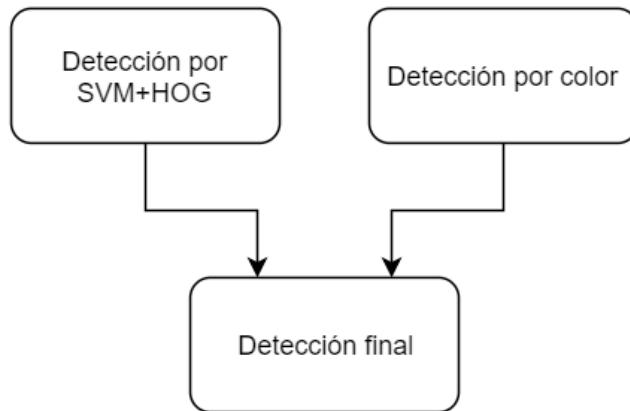


Fig 3.1: Estructura del proceso de detección de minas

– Detección por SVM+HOG

Se diseñará un programa para obtener ROI's de minas a partir de imágenes y videos. Este mismo programa se encargará de compatibilizar estas imágenes de ROI's con el formato necesario.

Una vez creada la base de datos, se desarrolla un programa para entrenar un detector de tipo HOG+SVM utilizando la base de datos creada.

Finalmente, se obtendrán en forma de rectángulos las detecciones hechas.

- Detección por color

El primer paso de este proceso consistirá en convertir la imagen obtenida por cámara a distintos espacios de color. Mediante una función de thresholding se convertirán las imágenes a tipo binario, adjudicando al color negro un valor “0” (blanco) y al resto de colores un valor “1” (negro), y se reducirá el ruido mediante algoritmos de suavizado.

Una vez obtenida la imagen binaria pre-procesada, se detectarán los bordes con un algoritmo de Canny, y posteriormente se almacenarán los distintos contornos detectados como color negro.

Para finalizar, se hallarán los centros de estos contornos.

- Detección final

Mediante una lógica que compruebe la superposición de las dos detecciones detalladas anteriormente, se decidirá de forma final si el objeto detectado es una mina o no.

El resultado será utilizado en la competición “Minesweeper: Towards a Landmine-free world” que tendrá lugar en Madrid del 2 al 5 de octubre de 2018.

3.2 Descripción del software

3.2.1 OpenCV

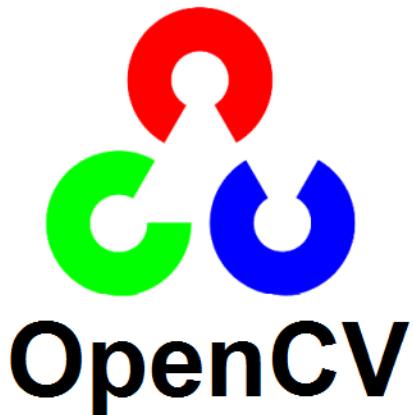


Fig 3.2: Logotipo de la librería de visión artificial "OpenCV"

OpenCV (Open Source Computer Vision) es una biblioteca de código abierto que ofrece funciones para el modelado de imágenes, la detección de objetos, y otra serie de procesos relacionados con la visión artificial.

Fue lanzada en 1999 por Intel, como un proyecto para simplificar los algoritmos de visión y fomentar su desarrollo, evitando que distintos fabricantes, investigadores, ... perdiessen tiempo reinventando códigos ya existentes, pero no disponibles.

Adquirió importancia en el año 2005 al ser utilizada por la universidad de Stanford para ganar el gran desafío DARPA de conducción autónoma. Tras ese punto de inflexión, esta librería se ha convertido en la más utilizada en el entorno de la visión artificial por su simpleza, constante actualización, carácter multiplataforma y compatibilidad con frameworks de Deep learning como TensorFlow, Torch y Caffe.

En la actualidad openCV se encuentra en su versión 3.4, recibiendo actualizaciones casi mensuales [12] [13].

3.2.2 ROS



Fig 3.3: Logotipo del Sistema Operativo Robótico, ROS.

ROS, siglas que corresponden al Sistema Operativo Robótico (Robot Operating System en inglés), es un entorno de trabajo o framework que provee las funcionalidades típicas de un sistema operativo, sin llegar a serlo, para facilitar el desarrollo de software enfocado en robots. Estas funcionalidades, entre muchas otras, pueden ser:

- Abstracción del hardware
- Control de dispositivos a bajo nivel
- Implementación de funcionalidades de uso común
- Mantenimiento de paquetes
- Paso de mensajes entre procesos



Fig 3.4: Estructura de ROS

ROS surgió en el año 2007 en el laboratorio de inteligencia artificial de Stanford, como fruto de la unión de varios entornos de software enfocados en robótica de uso libre. No fue hasta 2010 cuando apareció la primera versión estable de ROS (ROS 1.0), y este mismo año fue usado para volar por primera vez un dron.

Su arquitectura se basa en la teoría de grafos, localizándose en los nodos todos los procesos relacionados con recibir, mandar, multiplexar mensajes de sensores, ... ROS en sí mismo no es un sistema operativo en tiempo real (RTOS), pero es posible implementarlo con código en tiempo real.

Se pueden apreciar 3 grupos diferenciados dentro de ROS:

- Sistema Operativo en general, incluyendo el lenguaje, las herramientas,... para distribuir y compilar el software de ROS.
- Las librerías de implementación de ROS, como pueden ser “roscpp” (compatibilidad con C++), “rospy” (con Python), etc...
- Packages, una suite de paquetes creados por usuarios externos a ROS (por lo general) que aporta multitud de funciones al sistema operativo. Un ejemplo de estos “pkg” sería la librería OpenCV, que se puede utilizar con facilidad.

Por lo general, ROS es un software libre bajo la licencia BSD, así como lo son la mayoría de sus packages. Existen excepciones de estos últimos cuando se habla de robots y otras aplicaciones de uso comercial.

ROS está orientado para un sistema UNIX (Ubuntu,...), pero existen versiones experimentales con otros sistemas como Windows.

Las versiones de ROS adquieren distintos nombres (Jade Turtle, Kinetic Kame,...), y a su vez disponen de distintos períodos de mantenimiento dependiendo de la estabilidad de cada una. Desde 2013, ROS ha publicado versiones nuevas de forma anual [14] [15].

Las versiones que a día de hoy siguen actualizándose son las siguientes:

- Kinetic Kame (23 de Mayo de 2016 – 30 de Mayo de 2021)
- Lunar Loggerhead (23 de Mayo de 2017 – 30 de Mayo de 2019)
- Melodic Morenia (23 de Mayo de 2018 – 30 de Mayo de 2023)

4 DESARROLLO DEL ALGORITMO

4.1 Fundamentos teóricos

4.1.1 Histograma de gradientes orientado (HOG)

El histograma de gradientes orientados, también conocido como HOG, es un descriptor de objetos centrado en la apariencia de estos, frecuentemente utilizado en la detección y etiquetado de objetos mediante información visual.

Descriptor es el nombre que se le da a las aplicaciones o algoritmos que extraen información particular de un objeto, ya sea su textura, orientación, forma, etc. Dependiendo de lo exhaustivo que sea el descriptor, se distinguen 2 tipos diferentes:

- Descriptores generales: aquellos que dan información relativamente “simple” como puede ser el color, la forma,... de un objeto.
- Descriptores de dominio específico: son descriptores más complejos, obtienen características como pueden ser los gradientes en los bordes de las imágenes, los puntos de interés mediante la matriz hessiana, etc.

Como su propio nombre indica, HOG calcula la orientación del gradiente de cada píxel, que variará dependiendo de su cercanía a los bordes y otras diferencias con las características de los píxeles vecinos. Hallados estos gradientes, es posible describir y clasificar objetos del mismo tipo, siempre que sus formas sean parecidas.

Antes de entrar en el orden de procesos que sigue el descriptor, es importante saber la organización que se sigue a la hora de seleccionar píxeles:

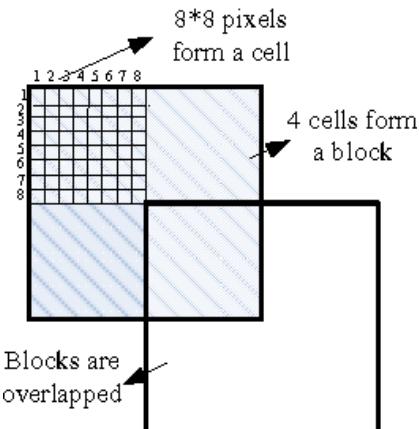


Fig 4.1: Estructura de las máscaras en un proceso de HOG: píxeles, celdas y bloques

La resolución de una imagen nos da su tamaño en matriz, por ejemplo, una imagen de 64x64 sería una matriz de 64 píxeles de altura por 64 píxeles de anchura. Estos píxeles han de dividirse en celdas, de 4x4 píxeles, 8x8,... a menor número de píxeles en cada celda, mayor es la sensibilidad del sistema, así como el coste computacional.

En formas simples es posible obtener mejores resultados mediante celdas de tamaños mayores, ya que al no existir cambios visibles en zonas pequeñas analizar los gradientes en estas puede llevar a que la sensibilidad sea tan grande que cualquier objeto parecido sea detectado como un igual.

Las celdas se agrupan en bloques, que han de ser múltiplos de estas, y que conforman la selección de la imagen sobre la que se realizarán las “comparaciones” entre imágenes. Así, una imagen de 64x64 dividida en bloques de 16x16 requerirá de 7 movimientos horizontales y 7 verticales, dando un total de 49 posiciones distintas.

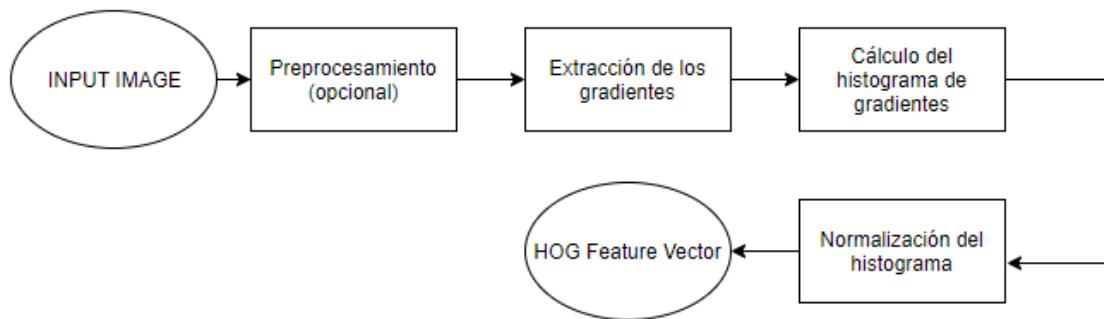
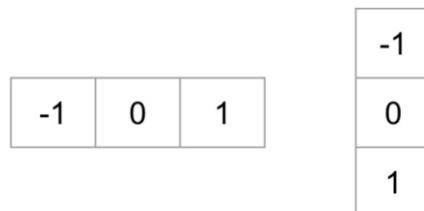


Fig 4.2: Procesos del "Histogram of Gradients", HOG.

El proceso de cálculo del vector de descriptores mediante HOG es el siguiente:

La imagen de entrada es procesada para eliminar ruidos no deseados, problemas con la iluminación, etc. Este proceso es opcional, ya que no se notan mejorías significativas tras la realización de esta normalización.

Para simplificar el cálculo de los gradientes, se aplican filtros kernel que derivan al resultado:



Es posible utilizar otro tipo de máscaras: Sobel, Prewitt, ...

El resultado sería algo parecido a lo representado en la siguiente imagen:

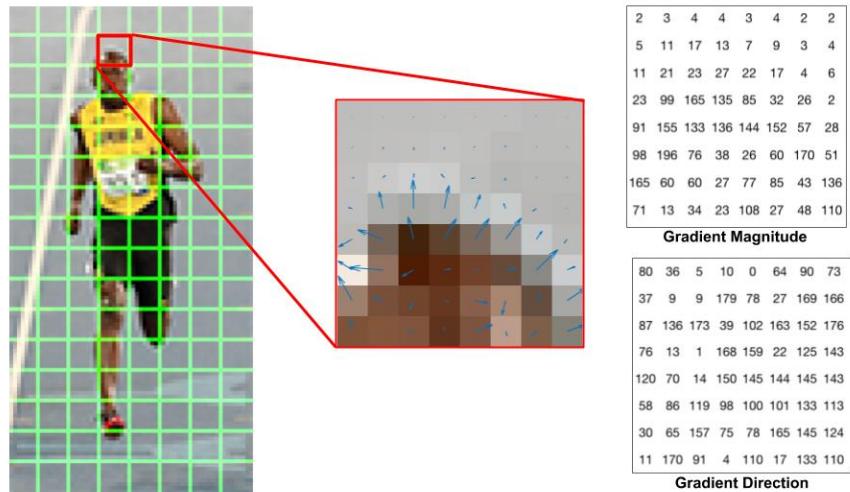


Fig 4.3: Ejemplo gráfico del resultado de un HOG

Tras obtener los gradientes, se realizan los histogramas que nos indicarán las direcciones predominantes en cada celda. Se suele dividir estos histogramas en 9 “bins”, correspondiendo a los ángulos: 0°, 20°, 40°, 60°, 80°, 100°, 120°, 140°, 160°.

Para la adjudicación de valores se tienen en cuenta tanto la dirección como la magnitud del gradiente de cada píxel: si un píxel tiene un valor de 80° y una magnitud de 2, se suma este 2 al bin representativo de esa dirección en el histograma. En caso de valores de dirección intermedios, se divide la magnitud entre los 2 ángulos más próximos de forma equitativa según el valor de la dirección. Se puede comprobar en la siguiente imagen:

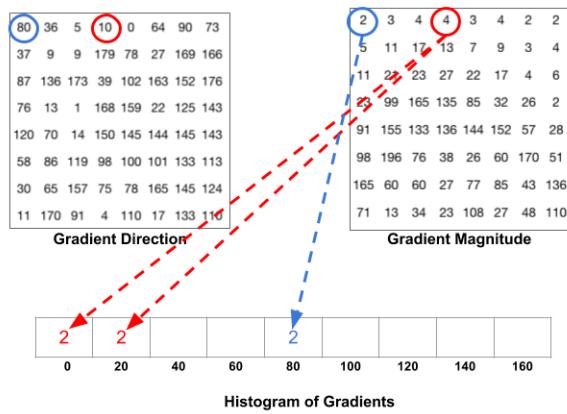


Fig 4.4: Visualización de las operaciones realizadas en un HOG

Una vez se ha hallado el histograma de todas las celdas, es necesario normalizar los resultados para eliminar errores derivados de cambios de iluminación. Esta normalización se realiza sobre los bloques, que dependiendo de su tamaño pueden agrupar 4, 16,... celdas distintas con sus respectivos histogramas [16] [17] [18] [19] [20] [21].

Normalizados los histogramas, los resultados son guardados en un solo vector con un tamaño igual a:

$$M \times \frac{B}{C} \times b$$

Fig 4.5: Fórmula para obtener el tamaño final del vector obtenido tras realizar un HOG

Siendo:

- M = Número de posiciones en las que se coloca un bloque. Ejemplo: imagen de 64x64 con un bloque de 16x16, es necesario mover el bloque 7 veces en horizontal y 7 en vertical, la M sería igual a 49.
- B = Tamaño del bloque
- C= Tamaño de la celda
- b = Número de bins por histograma.

4.1.2 Máquinas de soporte vectorial (SVM)

Las máquinas de soporte vectorial son un conjunto de algoritmos de aprendizaje supervisado para la resolución de problemas de clasificación y regresión.

Inicialmente, se pensaron para resolver problemas de clasificación binaria, es decir, diferenciar 2 clases, pero con el paso del tiempo se han optimizado y actualmente es posible utilizarlas en problemas de regresión, agrupamiento y multiclasiificación (varias clases).

Las SVM pertenecen a la categoría de clasificadores lineales, clasificadores que permiten la formación de una frontera de decisión, o hiperplano, alrededor del dominio de los datos de aprendizaje.

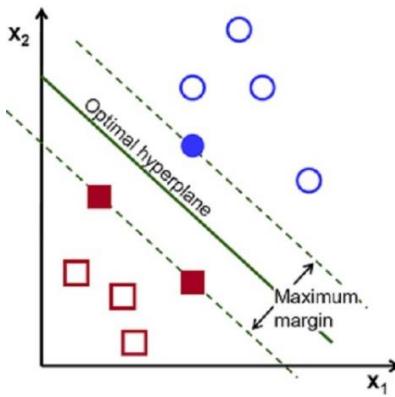


Fig 4.6: Gráfico 2D representativo de una selección por SVM

La mayoría de los métodos de aprendizaje se centran en minimizar errores basándose en los ejemplos de entrenamiento, error empírico, mientras que las SVMs buscan minimizar el riesgo estructural, o lo que es lo mismo, maximizar la distancia entre los 2 objetos de cada clase más cercanos entre sí.

A la hora de determinar el hiperplano, solo se tienen en cuenta los ejemplos de entrenamiento que caen justo en las fronteras de su clase. Son llamados “Support Vectors”.

En el caso de la visión por computador, las imágenes conforman una base de datos compleja y no separable linealmente. Por ello, es necesario crear espacios transformados de alta dimensionalidad en los que situar los hiperplanos capaces de actuar como frontera entre las clases. Por ejemplo, una base de datos de objetos de 2 dimensiones, deberán ser clasificado por un hiperplano situado en un espacio de 3 dimensiones.

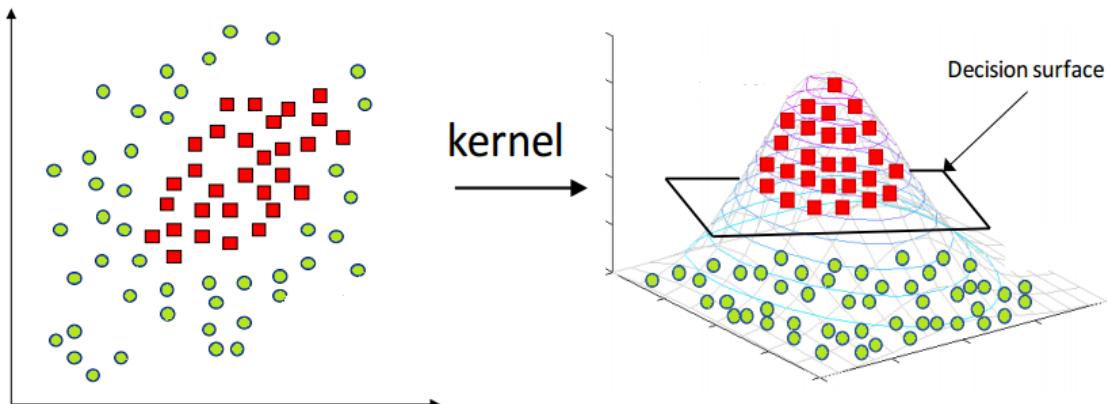


Fig 4.7: Gráficos 2D y 3D representativos de una selección por SVM

Estos espacios reciben el nombre de “espacio de características”, y su función de decisión viene dada por la siguiente fórmula:

$$D(\mathbf{x}) = (w_1\phi_1(\mathbf{x}) + \dots + w_m\phi_m(\mathbf{x})) = <\mathbf{w}, \Phi(\mathbf{x})>$$

Fig 4.8: Fórmula del hiperplano de un SVM

Pero esto da como resultado un hiperplano en la dimensión inicial, para transformarlo entran en juego los filtros kernel:

$$D(\mathbf{x}) = \sum_{i=1}^n \alpha_i^* y_i K(\mathbf{x}, \mathbf{x}_i)$$

Fig 4.9: Fórmula del filtro kernel

Dando como resultado:

$$K(\mathbf{x}, \mathbf{x}') = <\Phi(\mathbf{x}), \Phi(\mathbf{x}')> = (\phi_1(\mathbf{x})\phi_1(\mathbf{x}') + \dots + \phi_m(\mathbf{x})\phi_m(\mathbf{x}'))$$

Fig 4.10: Formula final de un SVM tras aplicar un filtro kernel para añadir un plano.

Así, es posible asignar a cada objeto de entrada una nueva posición en el nuevo plano mediante el uso de funciones kernel. Algunos ejemplos de estas funciones son:

- Kernel lineal:

$$K(\mathbf{x}, \mathbf{x}') = <\mathbf{x}, \mathbf{x}'>$$

Fig 4.11: Ecuación kernel lineal

- Kernel polinómico:

$$K_p(\mathbf{x}, \mathbf{x}') = [\gamma <\mathbf{x}, \mathbf{x}'> + \tau]^p$$

Fig 4.12: Ecuación kernel polinómico

- Kernel Gaussiano:

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2), \gamma > 0$$

Fig 4.13: Ecuación kernel gaussiano

- Kernel Sigmoidal:

$$K(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + \tau)$$

Fig 4.14: Ecuación kernel sigmoidal

En definitiva, dependiendo de la dificultad en la separación de los datos de distintos objetos, será necesario ir creando nuevos planos para la separación de los datos [22] [23].

4.1.3 Espacios de color

Dentro de los distintos descriptores que caracterizan una imagen, el color es uno de los que más destacan a la hora de diferenciar objetos. Inicialmente era poco utilizado debido a su alto gasto computacional, pero con las innovaciones en GPU's y otros dispositivos hardware, en la actualidad es posible su uso a la hora de realizar detecciones en tiempo real.

Sin embargo, el color de un objeto se ve fácilmente afectado por cambios en la iluminación, reflejos, etc., por lo que es conveniente utilizar distintos espacios de color para obtener mejores resultados. A continuación, se explicarán los espacios de color más utilizados [24].

4.1.3.1 RGB

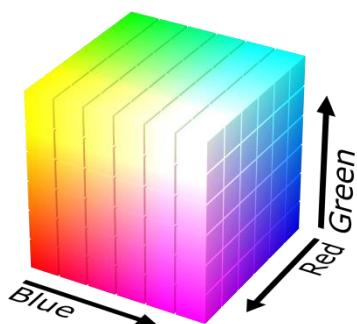


Fig 4.15: Representación en 3D del espacio de color RGB

Es el espacio de color más común, y el establecido como “estándar” en la librería openCV. Está basado en la síntesis aditiva: mediante mezclas lineales de los 3 colores primarios (rojo, verde y azul) es posible obtener cualquier otro color. Todos estos están correlacionados con la cantidad de luz que alcanza la superficie del objeto.

Su principal ventaja es que su uso resulta muy intuitivo, sin embargo, no resulta la mejor opción a la hora de analizar imágenes según su color, ya que no separa de forma correcta los cambios de iluminación respecto a los cambios cromáticos, produciendo distintos resultados en función de la claridad-oscuridad de una misma imagen.

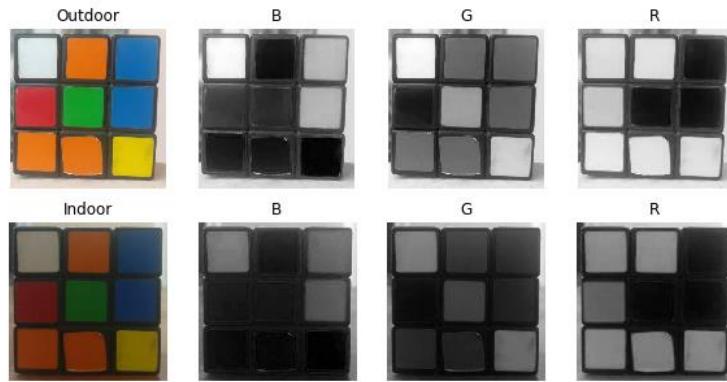


Fig 4.16: Separación de los canales de color R-G-B en distintas condiciones de iluminación

Podemos observar como en los ejemplos tomados con una buena iluminación es posible diferenciar correctamente los colores en función del canal elegido, pero ocurre lo contrario en la imagen de baja iluminación, donde los colores que no corresponden al canal elegido son indistinguibles. Por ejemplo, en la imagen del canal R en la iluminación “indoor”, los colores naranja, amarillo y blanco son exactamente iguales, y lo mismo ocurre con el azul y el verde.

4.1.3.2 HSV

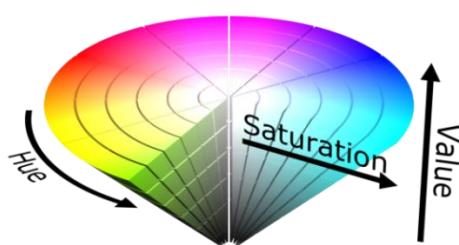


Fig 4.17: Representación 3D del espacio de color HSV

El modelo de color HSV proviene de una transformación no lineal del espacio de color RGB, y está compuesto por 3 componentes:

- H: “Hue”, o el matiz. Se representa como un grado de ángulo entre 0° y 360°, correspondiéndose cada valor a un color.
- S: “Saturation”, marca la tonalidad del color en rangos de 0 a 100%, siendo 0% un color muy claro (tonalidad blanca) y 100% un color puro o saturado (tonalidad negra). Permite una fácil diferenciación entre colores como el rojo o el rosa.
- V: “Value”, determina la intensidad del color. Los valores van del 0 (negro) al 100% (blanco o un color más o menos saturado).

Este modelo es de los más utilizados en la visión por computador debido a que sus canales están perfectamente separados mediante los 3 componentes nombrados anteriormente. El componente H se encarga de describir de manera totalmente independiente el matiz del color, permitiendo al usuario seleccionar de manera muy precisa los rangos de color sobre los que se quiere trabajar, obviando valores perjudiciales como puede ser una iluminación variable.

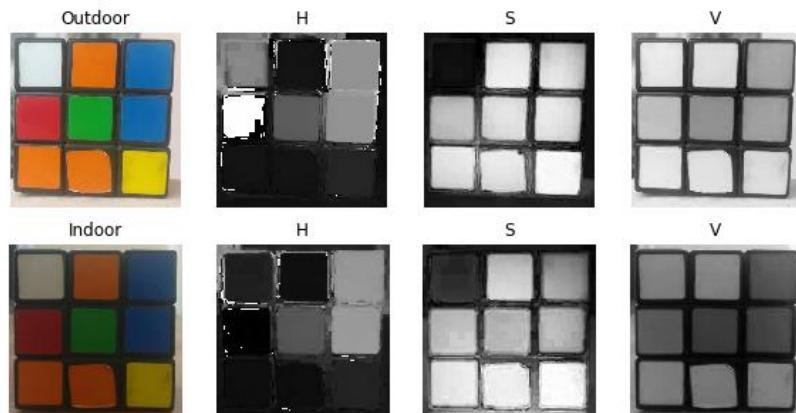


Fig 4.18: Separación de los canales de color H-S-V en distintas condiciones de iluminación

En la imagen superior se puede apreciar que las variables H y S son bastante similares en ambos entornos, lo que indica una alta independencia respecto a los cambios de iluminación. Por otro lado, la variable V si se ve afectada por estos cambios.

4.1.3.3 YCrCb

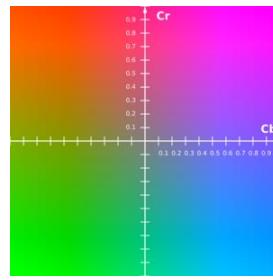


Fig 4.19: Representación 2D del espacio de color YCrCb

El formato YCrCb no es un espacio de color en sí, sino una forma de codificar información RGB. Viene determinado por 3 componentes:

- Y: Luminancia obtenida de los valores RGB tras realizar una corrección de gamma.
- Cr: R – Y, o la distancia entre el valor del componente rojo y la luminancia.
- Cb: B – Y, o la distancia entre el valor del componente azul y la luminancia.

Así, los valores cromáticos quedan representados por dos canales mientras que la iluminación se representa en tan solo uno. Se utiliza mucho en televisión.

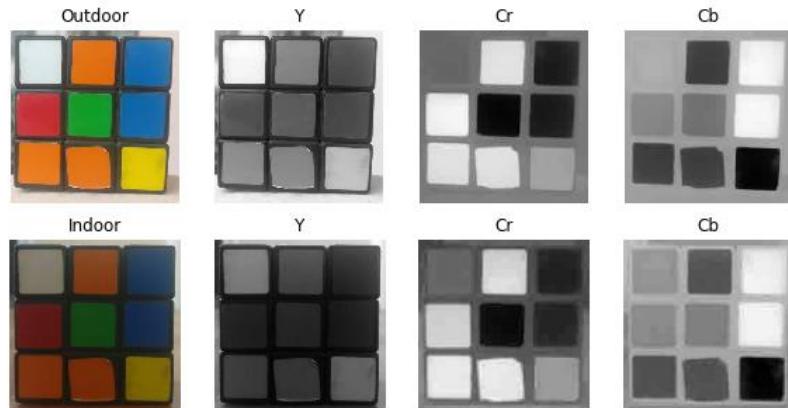


Fig 4.20: Separación de los canales de color Y-Cr-Cb en distintas condiciones de iluminación

En la imagen, el canal Y se ve afectado por el cambio de entorno, pero los otros dos canales permanecen casi invariables. Como desventaja, la diferencia entre colores como el rojo y el naranja es casi inapreciable [25].

4.1.3.4 Lab

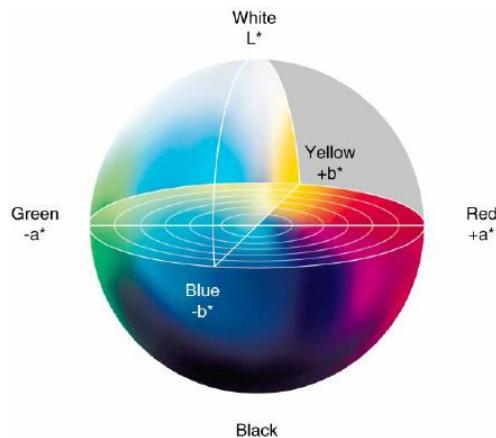


Fig 4.21: Representación en 3D del espacio de color Lab

Lab es el nombre abreviado de CIELAB y no sigue los patrones típicos del espacio RGB. En este, el canal L es independiente de la información cromática y engloba solo a la iluminación, mientras que los canales a y b se encargan solamente de indicar el color. Así, sus 3 canales serían los siguientes:

- L: “Lightness”, o intensidad de brillo.
- a: Componente de color, con rango desde verde hasta magenta.
- b: Componente de color, con rango desde azul hasta amarillo.

Su principal ventaja es la independencia del valor del brillo respecto a los valores del color, pero su composición es compleja desde el punto de vista del ojo humano. Se utiliza mucho en Photoshop.

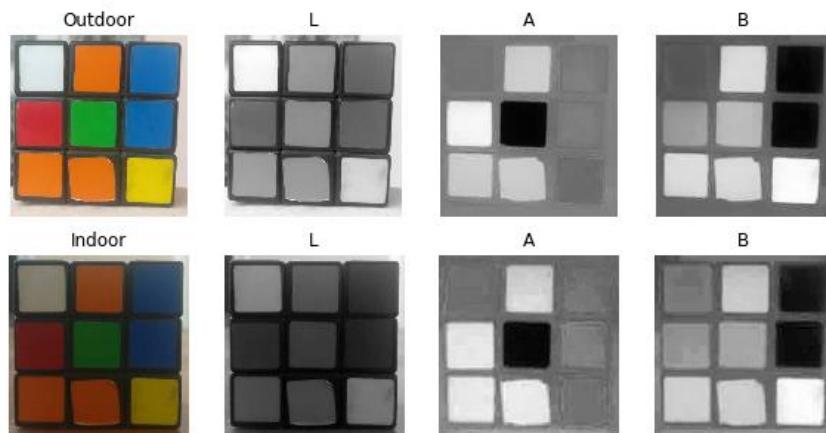


Fig 4.22: Separación de los canales de color L-A-B en distintas condiciones de iluminación

Como se ha dicho en la teoría, los cambios de brillo solo se aprecian en el canal L. Tanto en el canal A como en el B se puede observar que sus valores máximos y mínimos de rango están en negro y en blanco, por ejemplo, el verde en el canal A es negro y el rojo/magenta blanco, pero no se ven diferencias entre las imágenes de interior y exterior [26].

4.1.4 Reducción de ruido

4.1.4.1 Filtro mediana

El filtro de mediana es un filtro no lineal utilizado en el procesamiento de imágenes. Se puede utilizar para eliminar ruido de tipo gaussiano, pero es mucho más eficiente a la hora de eliminar ruido impulsional.

Estos filtros adjudican al píxel central el valor que tenga el mismo número de valores superiores que inferiores.

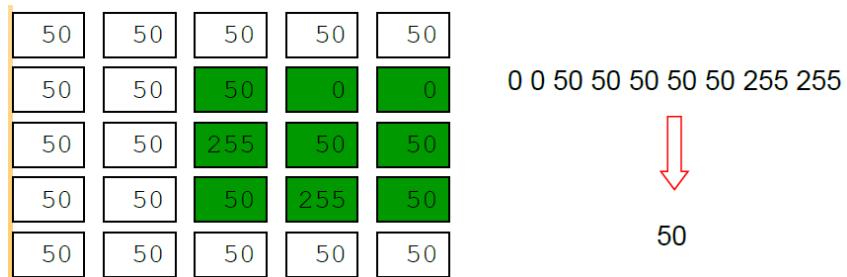


Fig 4.23: Visualización matemática de un filtro de mediana (Clase 6 - Sistemas de percepción (UC3M))

A mayor tamaño de la máscara, mayor es la eliminación de ruido impulsional; pero a su vez, la imagen se ve más deformada.

4.1.4.2 Transformaciones morfológicas

Las transformaciones morfológicas, dentro del ámbito de la visión artificial, son operaciones realizadas sobre imágenes binarias basadas en las formas de los objetos representados en dichas imágenes. Dentro de OpenCV, están compuestas por dos elementos: la imagen original sobre la que se realiza la operación, y el kernel, que decide el tipo de operación [27].

Dentro de las transformaciones morfológicas, la dilatación y la erosión son las operaciones más usuales y serán las utilizadas en este proyecto:

4.1.4.2.1 Erosión

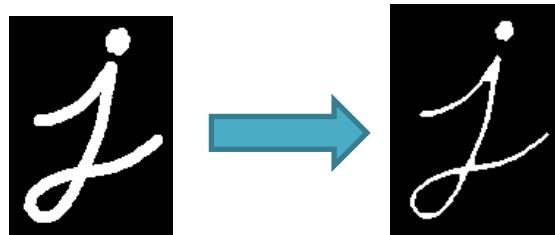


Fig 4.24: Ejemplo de un proceso de erosión

La erosión es un proceso que reduce el grosor de los objetos representados en blanco (se intenta representar siempre el objeto principal en este color) reduciendo los bordes de este.

Esto se realiza mediante un filtro kernel que se desplaza por toda la imagen realizando convoluciones en dos dimensiones.

1	1	1
1	1	1
1	1	1

Fig 4.25: Aspecto de un filtro kernel unitario de tamaño 3x3

Para cada píxel, este filtro determina su valor adjudicándole un “1”, o color blanco, en caso de que todos los píxeles de su alrededor tengan un valor de “1”, o le asigna un valor “0” en caso de que exista cualquier píxel a su alrededor que tenga un valor “0”.

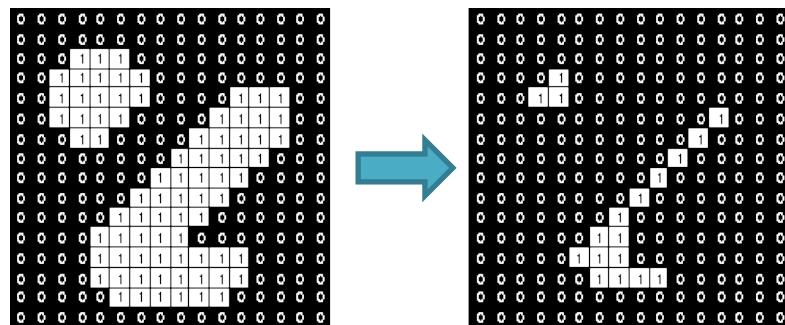


Fig 4.26: Visualización matemática de un proceso de erosión

El tamaño de la máscara de kernel es el parámetro que determina la cantidad de píxeles que se tendrán en cuenta a la hora de realizar la reducción. Así, cuanto mayor sea el tamaño de esta máscara mayor será la reducción que se producirá en los objetos mediante erosión.

Este tipo de transformación morfológica es muy efectiva para eliminar ruido impulsional.

4.1.4.2.2 Dilatación

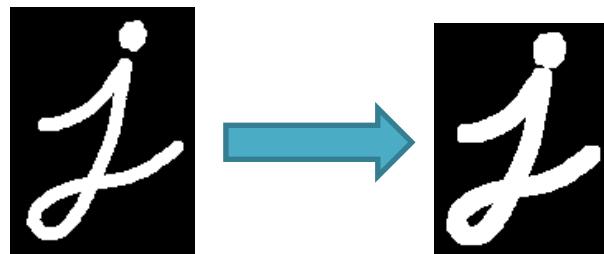


Fig 4.27: Ejemplo de un proceso de dilatación

Es el proceso opuesto a la erosión, aumenta el grosor del objeto mediante la extensión de sus bordes.

El filtro kernel en este caso adjudica un valor de “1” al pixel central sobre el que se encuentra en caso de que cualquiera de los píxeles de su alrededor sea 1.

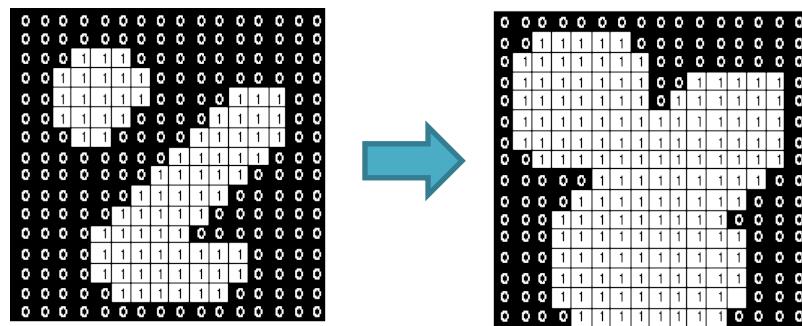


Fig 4.28: Visualización matemática de un proceso de dilatación

Es útil para recomponer partes separadas de un mismo objeto tras una erosión.

4.1.4.2.3 Opening

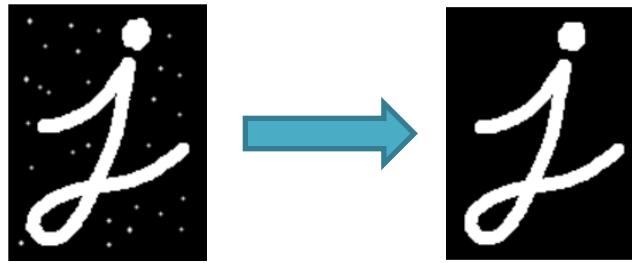


Fig 4.29: Ejemplo de un proceso de opening

“Opening” es el nombre que recibe un proceso en el que se realiza primero una erosión y luego una dilatación. Como hemos estudiado anteriormente, al realizar una erosión se eliminan ruidos impulsionales, pero a su vez se reduce el área del objeto principal. Para solucionar este problema y devolver el objeto principal a su tamaño original, se realiza una dilatación. El ruido impulsional, al haber sido eliminado, no reaparece con la dilatación.

4.1.5 Detector de bordes de Canny

Desarrollado por John F. Canny en 1986, este algoritmo es en la actualidad uno de los mejores métodos de detección de contornos, pero tiene un alto coste computacional [28] [29]. Se basa en el uso de máscaras de convolución, que representan aproximaciones en diferencias finitas; y busca satisfacer 3 criterios:

- Baja probabilidad de error, detecta todos y solo los bordes existentes.
- Buena localización, la distancia entre los píxeles detectados como bordes y los bordes reales ha de ser la mínima posible.
- Mínima respuesta, no debe identificar varios píxeles como bordes cuando sólo existe uno.

Es un operador óptimo, deriva de una gaussiana, y sus pasos son los siguientes:

1. Se tiene una imagen “I” y una gaussiana unidimensional “G”.
2. A partir de la gaussiana obtenemos, mediante derivadas unidimensionales, “Gx” y “Gy”:

$$Gx = \frac{\partial G(x)}{\partial x} \quad Gy = \frac{\partial G(y)}{\partial y}$$

3. Se convoluciona la imagen “I” con “Gx” y “Gy”, obteniendo “Ix” y “Iy”.

$$Ix = Gx * I(x, y) \quad Iy = Gy * I(x, y)$$

4. Obtenemos el módulo “M” de “Ix” e “Iy”.
5. Se eliminan los puntos que no sean máximos mediante umbrales T1 y T2
- 6.

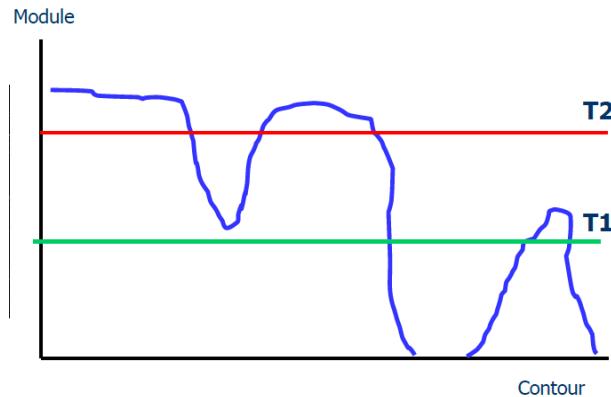


Fig 4.30: Representación gráfica de una selección de bordes mediante canny

Aquellos puntos que estén por encima de T2 son considerados bordes en su totalidad, los que están por debajo de T1 nunca son bordes, y los que se encuentran entre ambos umbrales son bordes solo si por lo menos uno de sus extremos alcanza el umbral T2.

4.2 Descripción del algoritmo

En el desarrollo de este trabajo se busca la máxima fiabilidad en la detección de las minas, optando por el uso de distintos métodos de detección de objetos para así, mediante su conjunción, eliminar los errores derivados por los puntos flojos de cada método.

Tras diversas pruebas, los métodos elegidos fueron los siguientes:

- Detección por SVM+HOG
- Detección por color

Una vez detectada la mina, se declarará que ese cuadrante está minado.

4.2.1 Detección mediante SVM+HOG

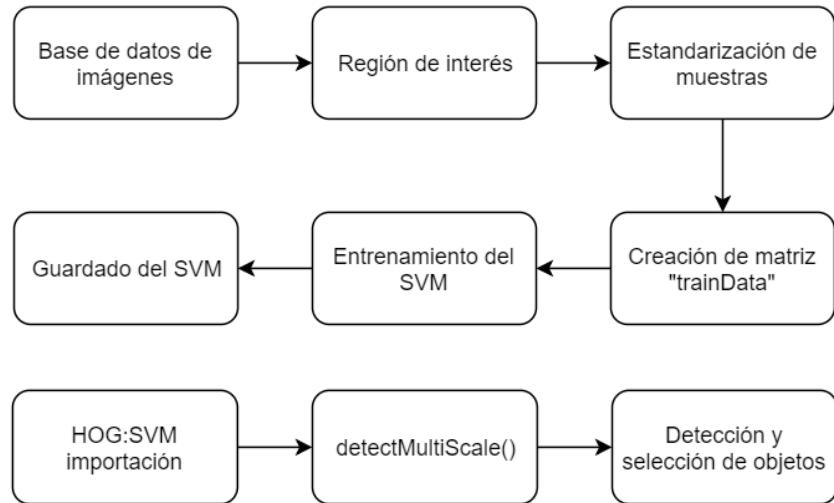


Fig 4.31: Procesos de la detección con SVM y HOG

La detección mediante SVM+HOG se basa en 2 conceptos:

- Uso del “Histogram of gradients” como descriptor para obtener la orientación del gradiente de cada píxel de una imagen representativa del objeto a detectar.
- Clasificación mediante “Support vector machine” de los objetos de interés basándose en los datos obtenidos con el HOG.

Estos métodos se explican con mayor detalle en el apartado de teoría del desarrollo del algoritmo.

4.2.1.1 Creación de la base de datos

Para realizar el entrenamiento de un SVM binario es necesario disponer de 2 bases de datos de imágenes:

- Positivas: las imágenes del objeto a detectar. Han de ser ROI's, es decir, regiones de interés en las que solo aparezca el objeto deseado con el mínimo ruido posible
- Negativas: cualquier imagen en la que no aparezca el patrón a detectar.

Todas estas imágenes, tanto las positivas como las negativas, han de ser de tamaño 32x32 o 64x64 y estar en escala de grises para poder ser computadas mediante HOG.

Dependiendo del tamaño elegido, podemos encontrar 2 posibilidades:

- 32x32: al ser un tamaño menor, el detector resultante requiere más gasto computacional, pero a su vez es más eficaz ya que es capaz de detectar los objetos buscados, aunque estos se encuentren en un tamaño reducido.
- 64x64: tanto el entrenamiento como la detección son más rápidos, pero se pierde efectividad ya que el tamaño de las máscaras de detección (64x64) será demasiado grande como para poder apreciar los objetos que ocupen menos espacio.

En este proyecto, se utiliza un tamaño de 32x32 para mejorar la detección de minas a distancias lejanas, tras haber comprobado que el equilibrio entre efectividad y gasto computacional es mejor manteniendo este tamaño y modificando otros valores como pueden ser el “winStride” y el “scale” de la función “detectMultiScale” del objeto HOG. Más adelante se explicarán con más detalles estas funciones.

Para simplificar la creación de esta base de datos, es implementado un código capaz de crear los ROI's (regiones de interés) de las minas a partir de fotos generales, y convertir cualquier imagen al formato deseado. Así mismo, se deja abierta la posibilidad de crear una base de datos de tamaño 64x64 si fuese necesario.

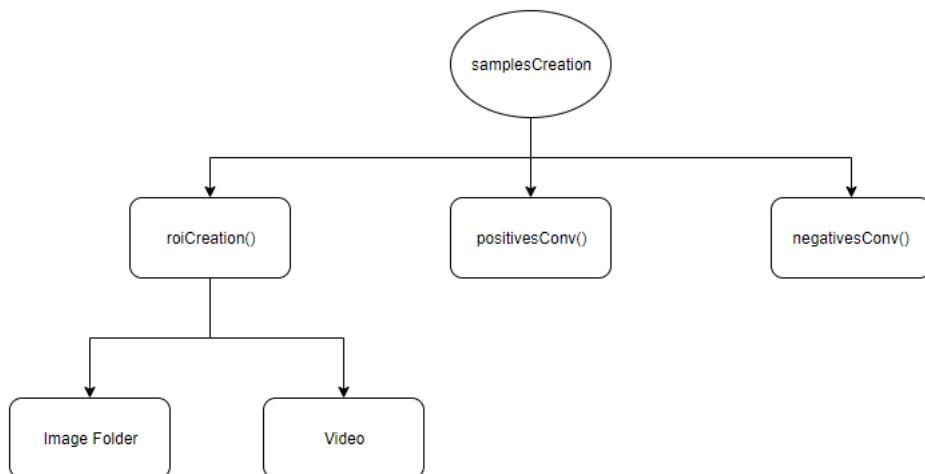


Fig 4.32: Estructura del archivo samplesCreation.cpp

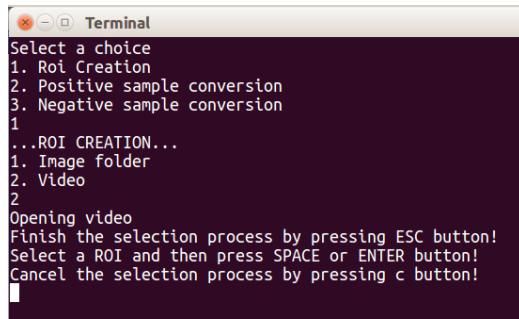


Fig 4.33: Ejecución del archivo *samplesCreation.cpp*

En la función *roiCreation()*, se ofrece la posibilidad de crear los ROI's deseados a partir de una carpeta de imágenes (Image Folder) o directamente de los frames de un video (Video). La carga de imágenes desde una carpeta se realiza mediante un objeto *directory_iterator* de la biblioteca adicional BOOST.

Para la selección de las regiones de interés, se utiliza la función *selectROIs* de la versión 3.3 de openCV, dentro del módulo de High-level GUI.

```
void cv::selectROIs (const String &windowName, InputArray img, std::vector< Rect > &boundingBoxes, bool showCrosshair=true, bool fromCenter=false)
Selects ROIs on the given image. Function creates a window and allows user to select a ROIs using mouse. Controls: use space or enter to finish current selection and start a new one, use esc to terminate multiple ROI selection process. More...
```

Fig 4.34: Desglose de la función "selectROIs" de la librería OpenCV

Esta función abre una imagen de entrada (“img”), y permite seleccionar mediante el ratón el recuadro que corresponde al objeto de interés.



Fig 4.35: Selección de una región de interés con la función "selectROIs"

Este recuadro se almacena en la recta “r”, y mediante esta recta es posible realizar un recorte en la imagen inicial para obtener la imagen ROI deseada.

```

case 2:
    cout << "Opening video" << endl;
    VideoCapture cap;
    Mat frame;
    //Video folder
    cap.open("/home/sikem/Vídeos/sampleVideo1.mp4");
    if (!cap.isOpened()) // if not success, exit program
    {
        cout << "Cannot open the video cam" << endl;
        break;
    }
    while (1)
    {
        bool bSuccess = cap.read(frame); // read a new frame from video
        if (!bSuccess) //if not success, break loop
        {
            cout << "Cannot read a frame from video stream" << endl;
            break;
        }
        //ROIs of the image
        vector<Rect> r;
        selectROIs("Selection of ROI's", frame, r);
        for (int i = 0; i < r.size(); i++)
        {
            Mat coneROI = frame(r[i]);
            vector<int> compression_params;
            compression_params.push_back(CV_IMWRITER_PNG_COMPRESSION);
            compression_params.push_back(9);
            String filenameROI = "/home/sikem/Imágenes/TFG/Test Dataset/positiveROI/" + to_string(imagesCounter) + ".png";
            cout << filenameROI << endl;
            imwrite(filenameROI, coneROI, compression_params);
            imagesCounter++;
        }
    }
    break;
}

```

Fig 4.36: Aspecto del código para la adquisición de ROIs a partir de un archivo de video

positivesConv() se encarga de cargar la carpeta en la que se han almacenado las imágenes obtenidas por *roiCreation()*, para posteriormente convertirlas a tamaño 32x32/64x64 y a escala de grises. En adición a esto, se modifica el tipo de imagen a .png, facilitando operaciones futuras que requieran la llamada a estas imágenes.



Fig 4.37: Recorte original



Fig 4.38: Recorte convertido a formato 32x32 en escala de grises

En *negativesConv()* se realiza un proceso parecido al de la anterior función, con la única diferencia de que estas imágenes se almacenan en una carpeta distinta.

Una vez ejecutadas estas 3 funciones, se obtienen 2 carpetas (positive-negative) compuestas por imágenes de tamaño 32x32 a escala de grises.

4.2.1.2 Matriz de entrenamiento

El siguiente paso tras crear la base de datos es realizar el proceso de entrenamiento mediante HOG de nuestro objeto SVM. OpenCV proporciona diversas herramientas que simplifican este proceso, pero es necesario convertir las imágenes al formato utilizado por la función *train()* de los objetos tipo *svm*.

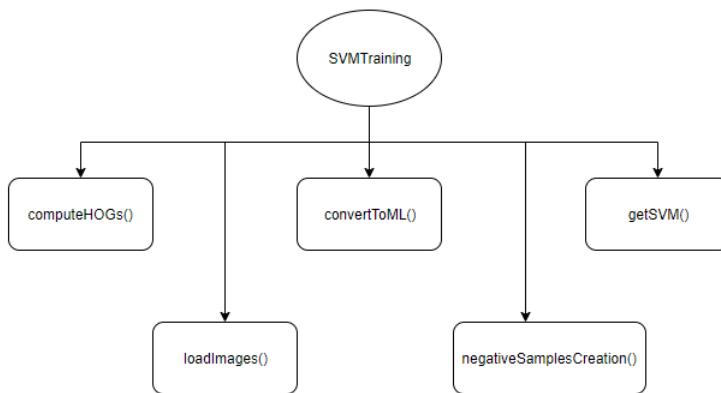


Fig 4.39: Estructura del archivo SVMTraining.cpp

Los pasos que se siguen desde la carga de las imágenes hasta el final del entrenamiento son los siguientes:

4.2.1.2.1 Inicialización

Mediante un “CommandLineParser” se cargan los directorios, valores,... introducidos en “const char* keys” . El formato es el siguiente:

“{*nombreVariable* |*directorio/valor/bool introducido*| *texto explicativo*} ”

Ejemplo:

```

"{help h}      | show help message"
"{pd  [/home/sikem/Imágenes/TFG/Test Dataset/samples32] path of directory contains possitive images}"
"{nd  [/home/sikem/Imágenes/TFG/Test Dataset/negative] path of directory contains negative images}"
"{td  [/home/sikem/Imágenes/TFG/Test Dataset/test] path of directory contains test images}"
"{tv  [/home/sikem/Vídeos/sampleVideo1.mp4] test video file name}"
"{dw  [32] width of the detector}"
"{dh  [32] height of the detector}"
"{f   [false] indicates if the program will generate and use mirrored samples or not}"
"{d   [true] train twice}"
"{g   [false] train again}"
"{t   [false] test a trained detector}"
"{v   [false] visualize training steps}"
"{fn  [mineDetector32_v1.yml] file name of trained SVM}"

```

Fig 4.40: Código en formato "CommandLineParser" para introducir los datos iniciales del entrenamiento

Valores:

- pd = positiveDir: directorio imágenes positivas
- nd = negativeDir: directorio imágenes negativas
- td = testDir: directorio imágenes de prueba
- fn = svmDetector: nombre con el que se guarda/carga el SVM entrenado
- tv = videoFilename: video de muestra
- dw = detectorWidth: anchura
- dh = detectorHeight: altura
- d = trainTwice: si es igual a TRUE, se realiza un doble entrenamiento incluyendo los falsos positivos del primer entrenamiento.
- f = flipSamples: se voltean las imágenes horizontalmente para obtener más imágenes positivas. Es eficaz en objetos simétricos.
- t = testDetector: si es positivo, se salta el entrenamiento y se prueba directamente el SVM

4.2.1.2.2 Entrenamiento

Se comprueban los directorios introducidos, y se crean los siguientes vectores:

- vector<Mat>: vectores donde se cargarán las imágenes. El número de columnas del vector será igual al número de imágenes por el tamaño horizontal de estas, y el número de filas será el tamaño vertical de las imágenes. Trabajando con imágenes de 32x32 el resultado sería:
 - N° Filas = 32
 - N° Columnas = 32 x N° Imágenes
- Vector<int>: vector con las etiquetas del tipo de imagen. En nuestro caso trabajaremos con imágenes positivas a las que se les asignará la etiqueta ‘1’, y con imágenes negativas con etiqueta ‘-1’. Su tamaño será el siguiente
 - N° Filas = 1
 - N° Columnas = N° Imágenes

Cargamos los vectores de tipo Mat con las imágenes positivas y negativas por separado con la función *loadImages()*. En el caso de las negativas, no es necesario dar las imágenes con tamaño 32x32 porque esto se realiza en la función *negativeSamplesCreation()*.

Mediante la función *computeHOGs()* se obtienen los descriptores de cada imagen.

Se comienza declarando un objeto HOG:

```
HOGDescriptor (Size _winSize, Size _blockSize, Size _blockStride, Size _cellSize, int _nbins, int _derivAperture=1, double _winSigma=-1, int
_histogramNormType=HOGDescriptor::L2Hys, double _L2HysThreshold=0.2, bool _gammaCorrection=false, int
_nlevels=HOGDescriptor::DEFAULT_NLEVELS, bool _signedGradient=false)
```

Fig 4.41: Desglose de la clase "HOGDescriptor" de la librería OpenCV

Es una de las partes más importantes del programa, ya que los valores elegidos en esta declaración tendrán mucha importancia en la efectividad, la velocidad,... de nuestro detector.

- winSize: tamaño de la imagen.
- blockSize: tamaño del bloque.
- blockStride: desplazamiento del bloque.
- cellSize: tamaño de las celdas.
- nbins: número de movimientos del bloque tanto vertical como horizontalmente.

Otros detalles de la declaración son explicados en la parte teórica del histograma de gradientes.

Una vez se ha creado el objeto, se llama a la función “compute” que se encuentra dentro de la librería de dicha clase [30].

```
compute (InputArray img, std::vector< float > &descriptors, Size winStride=Size(), Size padding=Size(), const std::vector< Point >
&locations=std::vector< Point >()) const
```

Fig 4.42: Desglose de la función "compute", dentro de la clase HOG de la librería OpenCV

- img: imagen (Mat) cargada del vector (vector<Mat>) convertida a gris mediante cvtColor.
- Descriptors: vector<float> con los valores obtenidos en hog.compute.
- winStride: tamaño de la celda. Si se ponen valores bajos es muy lento, pero puede que mejore la efectividad.

Se llama a esta función con cada imagen del vector Mat introducido. Estos descriptores tienen un formato vector <Float>, por lo que:

Una imagen de 64*64 (columnas*fila), se convierte en un vector float de tamaño 1764. Este valor es el resultado de 7x7x4x9: imagen dividida en bloques de 16x16 y en celdas de 8x8, es necesario mover 7 veces en horizontal y 7 en vertical los bloques, en cada bloque hay 4 celdas y cada celda tiene un histograma de 9 valores.

Los descriptores, tanto de las imágenes negativas como de las positivas, se almacenan en una matriz: `vector<Mat> trainDescriptors`. Esta matriz está compuesta por 1764 filas y X columnas, siendo X el número de muestras.

El hecho de guardar las imágenes positivas y negativas por separado permite al programa asignar los valores a las etiquetas:

- Si la imagen es positiva: `trainLabels.assign(positive_count, +1)`
- Si la imagen es negativa: `trainLabels.assign(positive_count, -1)`

Por último, se realiza la conversión de `trainDescriptors` a una matriz normal “`trainData`” compatible con el entrenamiento en Machine Learning – función “`convertToML`”.

La matriz `trainData` tiene que ser del tipo `CV_32FC1`, y tendrá el siguiente tamaño:

- Filas: N° total de samples – cada fila será el descriptor de una imagen.
- Columnas: 1764 – Tamaño del descriptor de cada imagen. En nuestro caso es 1764 porque tenemos imágenes de 64*64 divididas en bloques de 16*16 y celdas de 8*8, necesitando un total de 7x7 posiciones, verticalxHorizontal, para acaparar toda la imagen; 4 histogramas por bloque, y 9 valores por histograma: 7x7x4x9.

Una vez la matriz de entrenamiento está preparada, es necesario crear un objeto SVM.

- **svmType** –

Type of a SVM formulation. Possible values are:

- **SVM::C_SVC** C-Support Vector Classification. n-class classification ($n \geq 2$), allows imperfect separation of classes with penalty multiplier `C` for outliers.
- **SVM::NU_SVC** ν-Support Vector Classification. n-class classification with possible imperfect separation. Parameter `ν` (in the range 0..1, the larger the value, the smoother the decision boundary) is used instead of `C`.
- **SVM::ONE_CLASS** Distribution Estimation (One-class SVM). All the training data are from the same class, SVM builds a boundary that separates the class from the rest of the feature space.
- **SVM::EPS_SVR** ε-Support Vector Regression. The distance between feature vectors from the training set and the fitting hyper-plane must be less than `p`. For outliers the penalty multiplier `C` is used.
- **SVM::NU_SVR** ν-Support Vector Regression. `ν` is used instead of `p`.

Fig 4.43: Tipos de SVM (SVM - docs.opencv.org)

- **kernelType** –

Type of a SVM kernel. Possible values are:

- **SVM::LINEAR** Linear kernel. No mapping is done, linear discrimination (or regression) is done in the original feature space. It is the fastest option. $K(x_i, x_j) = x_i^T x_j$.
- **SVM::POLY** Polynomial kernel: $K(x_i, x_j) = (\gamma x_i^T x_j + \text{coef0})^{\text{degree}}, \gamma > 0$.
- **SVM::RBF** Radial basis function (RBF), a good choice in most cases. $K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0$.
- **SVM::SIGMOID** Sigmoid kernel: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + \text{coef0})$.
- **SVM::CHI2** Exponential Chi2 kernel, similar to the RBF kernel: $K(x_i, x_j) = e^{-\gamma x^2(x_i, x_j)}, \chi^2(x_i, x_j) = (x_i - x_j)^2 / (x_i + x_j), \gamma > 0$.
- **SVM::INTER** Histogram intersection kernel. A fast kernel. $K(x_i, x_j) = \min(x_i, x_j)$.

Fig 4.44: Tipos de kernel en un objeto SVM (SVM - docs.opencv.org)

- Type: Es el tipo de fórmula utilizada en el SVM. Se escoge *Support Vector Classification* (C_SVC), que permite separaciones imperfectas entre clases, pero con penalizaciones.
- Kernel: Indica el tipo de filtro kernel utilizado. En este caso, se usa un kernel lineal que no realiza ningún mapeado.
- C: Es un parámetro utilizado en clasificadores del tipo C_SVC para adjudicar valor a las penalizaciones.

Declarado el objeto SVM, se llama a su función *train*:

```
svm->train(trainingData, ROW_SAMPLE, trainLabels);
```

Fig 4.45: llamada a la función "train" en el código

- trainingData: la matriz de entrenamiento que se ha detallado con anterioridad.
- trainLabels: matriz que asigna las etiquetas (1 para positivas, -1 para negativas) a los datos de la matriz de entrenamiento.
- ROW_SAMPLE: indica que las matrices anteriormente nombradas se leerán fila a fila.

Esta función se encarga de entrenar el SVM para realizar la clasificación de los objetos a detectar [31] [32] [33] [34].

3º Entrenamiento

```
[ INFO:0] Initialize OpenCL runtime...
...[done]
Histogram of Gradients are being calculated for positive images.....[done] ( positive count : 708 )
Histogram of Gradients are being calculated for negative images.....[done] ( negative count : 3078 )
Training SVM again.....[done]
Testing trained detector on negative images. This may take a few minutes...[0.18020129, 0.09577433, 0.1]
```

Fig 4.49: Código de ejemplo del tercer entrenamiento

Por último, guardamos nuestro *support vector machine* en formato .txt o .yml.

```
svm->save("mineDetector_v1.yml");
```

4.2.1.3 Detección

La detección y selección se efectúan en un objeto HOGDescriptor (Histogram of oriented gradients descriptor). Para ello, se carga el svm entrenado en el objeto HOG creado, y posteriormente se inicia una detección múltiple utilizando las funciones proporcionadas por la librería openCV.

Así, este apartado se puede subdividir en 2 procesos:

4.2.1.3.1 Carga del SVM

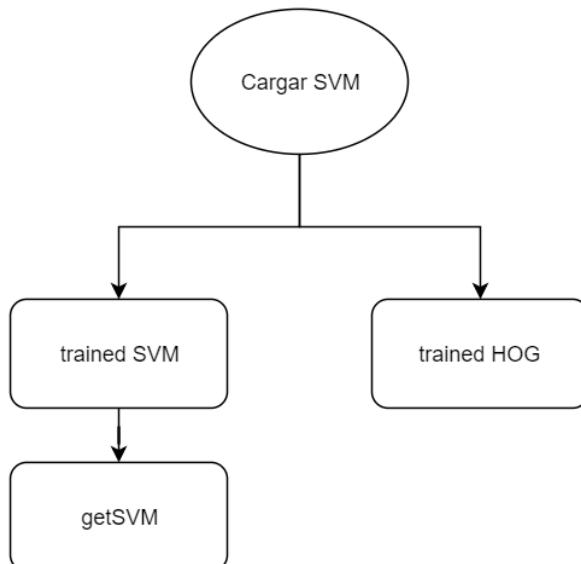


Fig 4.50: Procesos en la carga de un archivo SVM

El código en el que se realiza la detección es totalmente independiente del código de entrenamiento. Esto es así para evitar la necesidad de realizar nuevos entrenamientos cada vez que se quiera ejecutar un proceso de clasificación, lo que lleva a la necesidad de guardar el objeto SVM una vez se ha entrenado y a cargarlo posteriormente.

Existen 2 posibilidades a la hora de guardar y cargar un sistema entrenado:

- trained HOG: esta opción es la más simple. En el fichero de entrenamiento, se comprueba la efectividad del SVM entrenado mediante un HOG. Este detector HOG se puede almacenar mediante una función `.save` incluida en su misma clase, creando un archivo de formato `.txt` o `.yml`. A la hora de cargarlo, tan solo es necesario llamar a la función `.load` para asignarle todas sus características al nuevo objeto HOG, manteniendo el entrenamiento [35].
La principal desventaja de este método es que no es compatible con programación en CUDA y puede derivar en problemas de compatibilidad.
- Trained SVM: se trabaja con el objeto SVM entrenado. Es necesario realizar más operaciones que en la anterior opción, pero no da problemas de compatibilidad con programación en GPU's , etc. Por ello, este es el método utilizado para la carga del detector y será explicado con más detalles a continuación.

El primer paso es cargar el SVM entrenado en un nuevo objeto del mismo tipo.

Es obligatorio que el objeto creado y el cargado posean las mismas características, de lo contrario el programa terminará en error.

Posteriormente, se declara el objeto `HOGDescriptor`. Es necesario que su parámetro `winSize` sea del mismo tamaño que el `winSize` del SVM, en nuestro caso `32x32`.

La función de los objetos tipo `HOG` que permite seleccionar un SVM es `setSVMClassifier`, sin embargo es necesario convertir el objeto SVM a un tipo de dato compatible con esta función. Esta conversión se realiza en la función `getSVM`:

```

vector< float > getSVM(const Ptr< SVM >& svm)
{
    // Get the support vectors
    Mat sv = svm->getSupportVectors();
    const int sv_total = sv.rows;
    // Get the decision function
    Mat alpha, svidx;
    double rho = svm->getDecisionFunction(0, alpha, svidx);

    CV_Assert(alpha.total() == 1 && svidx.total() == 1 && sv_total == 1);
    CV_Assert((alpha.type() == CV_64F && alpha.at<double>(0) == 1.) ||
              (alpha.type() == CV_32F && alpha.at<float>(0) == 1.f));
    CV_Assert(sv.type() == CV_32F);

    vector< float > hog_detector(sv.cols + 1);
    memcpy(&hog_detector[0], sv.ptr(), sv.cols * sizeof(hog_detector[0]));
    hog_detector[sv.cols] = (float)-rho;
    return hog_detector;
}

```

Fig 4.51: Código de la función `getSVM()`

Tras este paso, solo es necesario probar el detector.

4.2.1.3.2 Algoritmo de detección

La función encargada de ejecutar la detección es la siguiente:

```

virtual void detectMultiScale (InputArray img, std::vector< Rect > &foundLocations, double hitThreshold=0, Size winStride=Size(), Size padding=Size(),
                             double scale=1.05, double finalThreshold=2.0, bool useMeanshiftGrouping=false) const
    Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles. More...

```

Fig 4.52: Desglose de la función "detectMultiScale" dentro de la clase `HOGDescriptor` de la librería `OpenCV`

- Img: Imagen sobre la que se realiza la detección.
- &foundLocations: vector en el que se almacenan los rectángulos que determinan la posición de las detecciones dentro de la imagen.
- hitThreshold: distancia máxima entre las características HOG entrantes y el plano de clasificación determinante del SVM. Si esta distancia es superior a la cifra dada, la detección es rechazada.
- winStride: determina el tamaño del cuadro de detección.
- Padding: indica la cantidad de píxeles en los que se rellena la ventana de detección antes de extraer las características HOG.
- Scale: determina el número de capas de transformación que sufrirá la imagen.
- useMeanshiftGrouping: permite elegir si se quiere agrupar detecciones muy cercanas suponiendo que pertenecen al mismo objeto.

Los parámetros más importantes son *winStride* y *Scale*. Una buena selección de ambos permite tanto una mejora en la fiabilidad de la detección como en la velocidad de esta [36] [37].

- *winStride*:

Este parámetro indica el tamaño de la máscara que se empleará para calcular los gradientes. Un tamaño pequeño de máscara, por ejemplo, 4x4, producirá una mayor sensibilidad y por lo tanto, un aumento de detecciones; pero a su vez limitará mucho la velocidad del detector por su coste computacional. Una máscara mayor (normalmente se suelen usar de 8x8), mejorará la velocidad pero perderá sensibilidad. Tras diversas pruebas, se utiliza una máscara de 4x4 por la redundancia de la velocidad de procesamiento del programa.

- *Scale*:

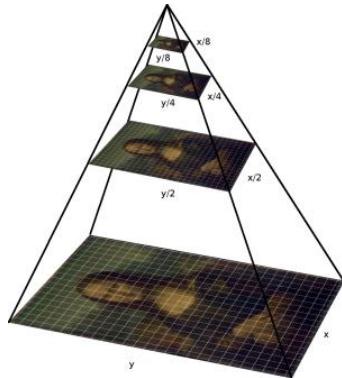


Fig 4.53: Representación de un "scale"

Este parámetro controla el factor en el que nuestra imagen será reducida en cada capa para su posterior procesado. Un factor de scale bajo será sinónimo de una baja reducción del tamaño de la imagen, lo que conlleva un aumento de la sensibilidad del detector y a su vez un aumento del coste computacional. En el programa se le asigna el valor mínimo 1.05 tras probar diversas combinaciones con el *winStride*, obteniendo una detección rápida y lo suficientemente sensible como para detectar correctamente las minas.

Por otro lado, *useMeanShiftGrouping* puede ser una buena opción en caso de que las detecciones realicen recuadros redundantes sobre los mismos objetos. Pero se ha demostrado mediante pruebas que es bastante ineficiente comparado a otros algoritmos utilizados con la misma finalidad.

Un ejemplo de estos algoritmos serían los conocidos como “Non-maximum Supression”, que basándose en los valores de confianza devueltos por la función *detectMultiScale* de la clase HOG agrupan las detecciones redundantes [38].



Fig 4.54: Aplicación de "Non-maximum Supresión" en una detección por HOG

Las detecciones son devueltas en forma de objetos tipo Rect, rectángulos recuadrando las áreas de la imagen en la que se han encontrado los patrones considerados como objetos por el SVM entrenado.



Fig 4.55: Resultado de una detección por SVM+HOG (1)

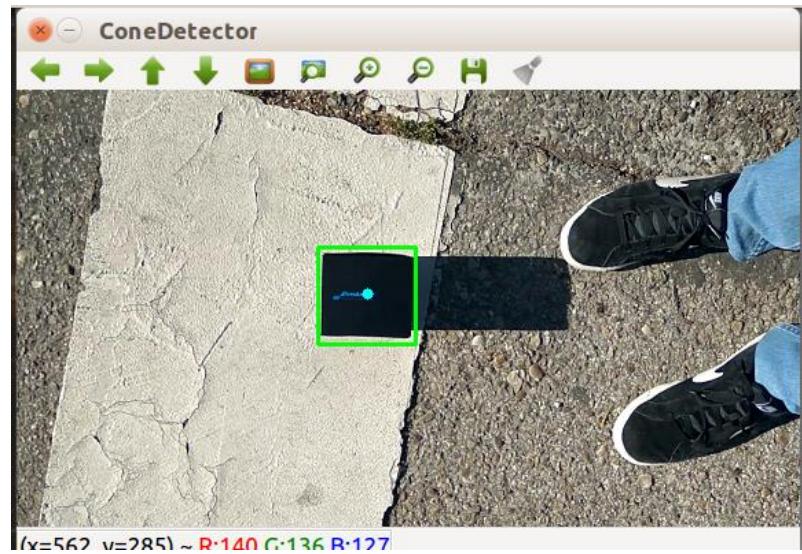


Fig 4.56: Resultado de una detección por SVM+HOG (2)

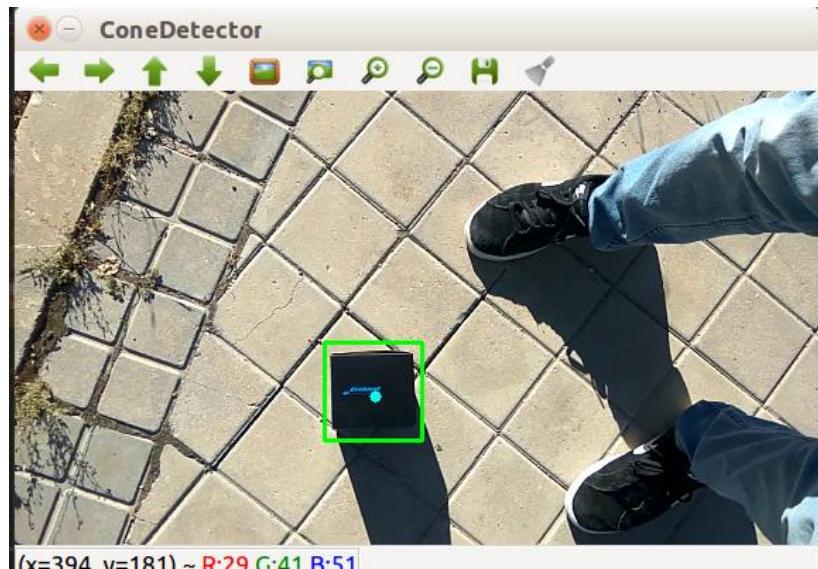


Fig 4.57: Resultado de una detección por SVM+HOG (3)

4.2.2 Detección mediante color

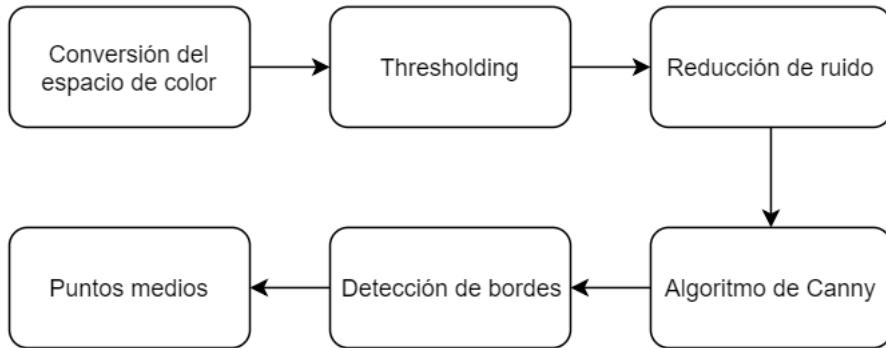


Fig 4.58: Procesos en una detección mediante color

En visión artificial, es posible separar mediante rangos los diferentes colores de una imagen. Al ser nuestra mina de un color específico, este método de detección es una opción válida, pero es necesario tener en cuenta posibles dificultades como reflejos, sombras,...

Por ello se realiza una detección en diversas escalas de color, aumentando la fiabilidad del detector y permitiendo su uso en distintas condiciones atmosféricas.

Para facilitar los posteriores algoritmos, también se realiza la detección de los centros de los contornos encontrados mediante color.

4.2.2.1 Conversión del espacio de color

El primer paso es obtener los valores del color a detectar, en nuestro caso el negro, en los distintos espacios de color que van a ser utilizados. Para ello se utiliza el archivo “getColour.cpp”, un programa capaz de proporcionar los valores correctos en los espacios “HSV”, “YCbCr” y “Lab” a partir de un valor de RGB.

El valor correcto en RGB se puede obtener mediante el programa Paint utilizando la herramienta “selector de color” sobre una imagen del objeto que queremos detectar. Una vez seleccionado este color, se abre la opción “Editar Colores” del programa y nos aparecerán los valores rojo (R), verde (G) y azul (B).

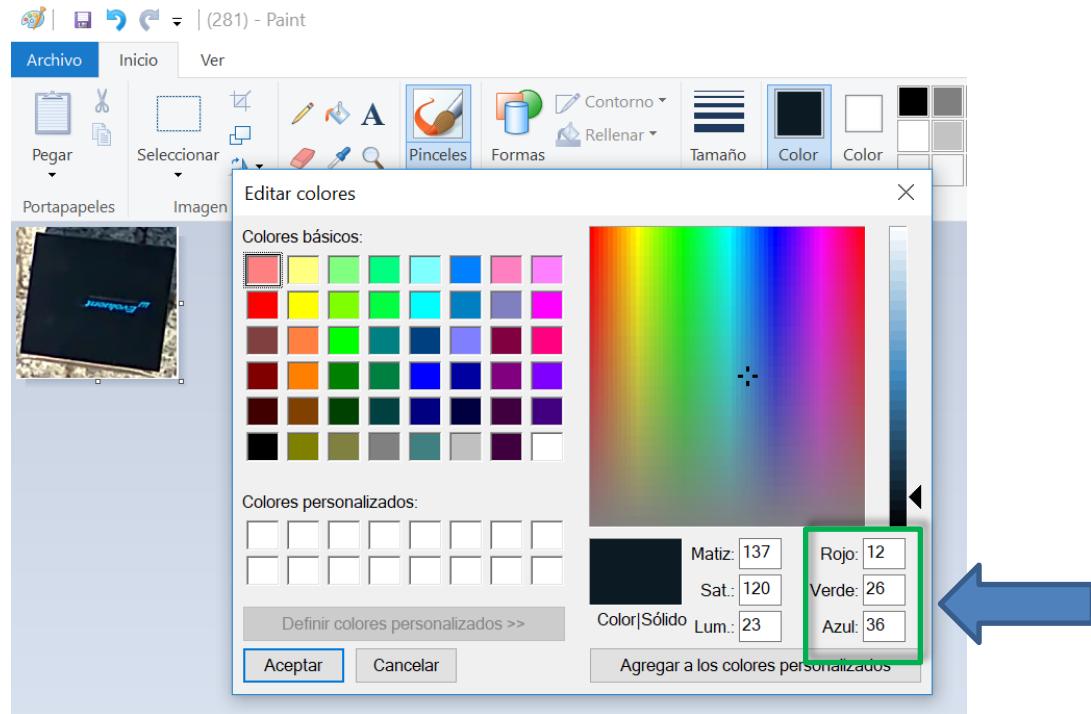


Fig 4.59: Obtención de los valores RGB de un color

Este proceso se repite sobre numerosas imágenes del objeto para poder focalizar el valor máximo y mínimo sobre los que se debería establecer el rango de detección.

En el caso del color negro, su valor RGB por lo general es (0,0,0), sin embargo en la práctica se le aplica un rango acorde a las variaciones que pueda sufrir por los reflejos.

Con el valor deseado ya hallado, se procede a crear un objeto tipo “Vec3b” con los valores de este, y posteriormente se convierte a uno tipo “Mat3b” para que resulte compatible con las funciones de conversión de color de OpenCV:

```
//C++ code
Vec3b bgrPixel(44, 30, 155);
// Create Mat object from vector since cvtColor accepts a Mat object
Mat3b bgr(bgrPixel);
```

Fig 4.60: Código de la conversión de Vec3b a Mat3b

Una vez obtenido el objeto con el color RGB deseado, se procede a su conversión a los tres espacios de color que serán utilizados:

```
//Convert pixel values to other color spaces.
Mat3b hsv, ycb, lab;
cvtColor(bgr, ycb, COLOR_BGR2YCrCb);
cvtColor(bgr, hsv, COLOR_BGR2HSV);
cvtColor(bgr, lab, COLOR_BGR2Lab);
```

Fig 4.61: Código de las conversiones de color

Estas conversiones siguen las siguientes fórmulas matemáticas [39]:

- RGB – HSV

$$\begin{aligned} V &\leftarrow \max(R, G, B) \\ S &\leftarrow \begin{cases} \frac{V - \min(R, G, B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases} \\ H &\leftarrow \begin{cases} 60(G - B)/(V - \min(R, G, B)) & \text{if } V = R \\ 120 + 60(B - R)/(V - \min(R, G, B)) & \text{if } V = G \\ 240 + 60(R - G)/(V - \min(R, G, B)) & \text{if } V = B \end{cases} \end{aligned}$$

Fig 4.62: Fórmula matemática de la conversión de RGB a HSV

Si el valor de H sale inferior a 0, se le suma 360.

Tras utilizar las anteriores formulas, se obtienen rangos de H entre 0 y 360, de V entre 0 y 1, y de S entre 0 y 1. Posteriormente es necesario convertir estos a los datos usados en los objetos Scalar de tipo SVM de OpenCV:

$$H = \frac{H}{2}$$

$$S = 255 \times S$$

$$V = 255 \times V$$

Siendo los valores máximos y mínimos de estos canales los siguientes:

H: 0 - 180	S: 0 - 255	V: 0 - 255
------------	------------	------------

- RGB – YCbCr

$$\begin{aligned}
 Y &\leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \\
 Cr &\leftarrow (R - Y) \cdot 0.713 + delta \\
 Cb &\leftarrow (B - Y) \cdot 0.564 + delta \\
 R &\leftarrow Y + 1.403 \cdot (Cr - delta) \\
 G &\leftarrow Y - 0.714 \cdot (Cr - delta) - 0.344 \cdot (Cb - delta) \\
 B &\leftarrow Y + 1.773 \cdot (Cb - delta)
 \end{aligned}$$

$$delta = \begin{cases} 128 & \text{for 8-bit images} \\ 32768 & \text{for 16-bit images} \\ 0.5 & \text{for floating-point images} \end{cases}$$

Fig 4.63: Fórmula matemática de la conversión de RGB a YCbCr

- RGB – LAB

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$X \leftarrow X/X_n, \text{ where } X_n = 0.950456$$

$$Z \leftarrow Z/Z_n, \text{ where } Z_n = 1.088754$$

$$L \leftarrow \begin{cases} 116 * Y^{1/3} - 16 & \text{for } Y > 0.008856 \\ 903.3 * Y & \text{for } Y \leq 0.008856 \end{cases}$$

$$a \leftarrow 500(f(X) - f(Y)) + delta$$

$$b \leftarrow 200(f(Y) - f(Z)) + delta$$

$$f(t) = \begin{cases} t^{1/3} & \text{for } t > 0.008856 \\ 7.787t + 16/116 & \text{for } t \leq 0.008856 \end{cases}$$

$$delta = \begin{cases} 128 & \text{for 8-bit images} \\ 0 & \text{for floating-point images} \end{cases}$$

Fig 4.64: Fórmula matemática de la conversión de RGB a LAB

Se obtienen valores entre: $0 < L < 100$, $-127 < a < 127$, $-127 < b < 127$. Es necesario convertir estos a los valores Lab de openCV, para ello se siguen las siguientes fórmulas:

$$L = \frac{L*255}{100}$$

$$a = a + 128$$

$$b = b + 128$$

Una vez realizadas las conversiones tendremos un objeto tipo “Mat3b” para cada espacio de color. Es necesario revertir la conversión hecha al inicio, para así poder usar el objeto Vec3b a la hora de crear los objetos “Scalar”.

```
//Get back the vector from Mat
Vec3b hsvPixel(hsv.at<Vec3b>(0, 0));
Vec3b ycbPixel(ycb.at<Vec3b>(0, 0));
Vec3b labPixel(lab.at<Vec3b>(0, 0));
```

Fig 4.65: Código de la conversión de Mat3b a Vec3b

Solo queda extraer los valores para usarlos como referencia a la hora de crear los rangos de detección. Estos rangos pueden establecerse con un threshold inicial, pero para mejorar la detección es necesario ir probando distintos valores hasta encontrar los que mejor se ajusten.

4.2.2.2 Thresholding

Para poder realizar la detección de contornos mediante el algoritmo de Canny es necesario trabajar con imágenes binarias, es decir, imágenes en las que el color a detectar aparecerá como blanco y el resto del espectro de colores aparecerá como negro (2 canales de color).

Se utiliza la función “inRange” proporcionada en la librería OpenCV:

```
void cv::inRange (InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst)
```

Fig 4.66: Desglose de la función "inRange" de la librería OpenCV

- src: objeto Mat sobre el que se realiza el threshold.
- lowerb: valor mínimo del rango, en formato Scalar.
- upperb: valor máximo del rango, en formato Scalar.
- dst: objeto Mat binario resultante.

Como se explicó anteriormente, para establecer los rangos de thresholding correctos se ha de trabajar sobre los valores obtenidos en el anterior paso. Pero este proceso depende del tipo de espacio de color, de la forma siguiente:

- HSV: Sobre los canales H y S no se aplica ningún filtro de rango, y para el canal V se establece un rango cercano al valor 0 (negro total).
- YCrCb/Lab: para estos espacios de color se utilizarán las conversiones explicadas anteriormente.

4.2.2.3 Reducción de ruido

Como resultado del proceso de thresholding obtenemos imágenes binarias como las siguientes:

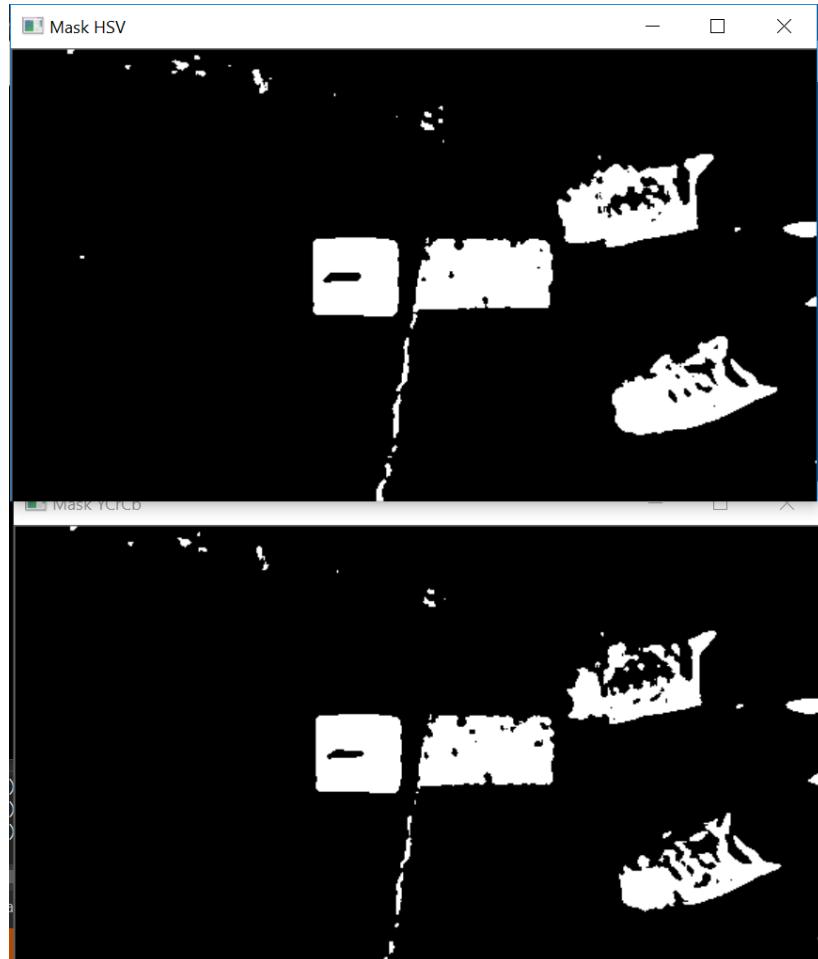


Fig 4.67: Imágenes binarias. La superior es la obtenida a partir de una imagen HSV, la inferior a partir de YCrCb

Se aprecia fácilmente que, aparte de los objetos negros con un volumen amplio que hay (en esta imagen se aprecian zapatillas negras, la caja, y la sombra de la caja), aparecen una serie de puntos tanto en la parte superior izquierda de la imagen como en la central inferior. Estos puntos se han de eliminar para evitar posibles falsos positivos en el detector.

Para ello, se utilizan métodos de reducción de ruido:

En primer lugar, se aplica un filtro de mediana

```
void cv::medianBlur (InputArray src, OutputArray dst, int ksize)
```

Fig 4.68: Desglose de la función "medianBlur" de la librería OpenCV

Con un tamaño de Kernel de 5 se obtiene una reducción considerable de ruido impulsional, pero no es eficaz a la hora de unificar detecciones por color para facilitar la posterior detección de contornos. Esto se soluciona con un proceso de “opening”, es decir, la unión de algoritmos de erosión y dilatación.

```
erode (colourMaskHSV, colourMaskHSV, element);
dilate(colourMaskHSV,colourMaskHSV, element);
```

Fig 4.69: Ejecución de un "opening" en código

El resultado es el siguiente:

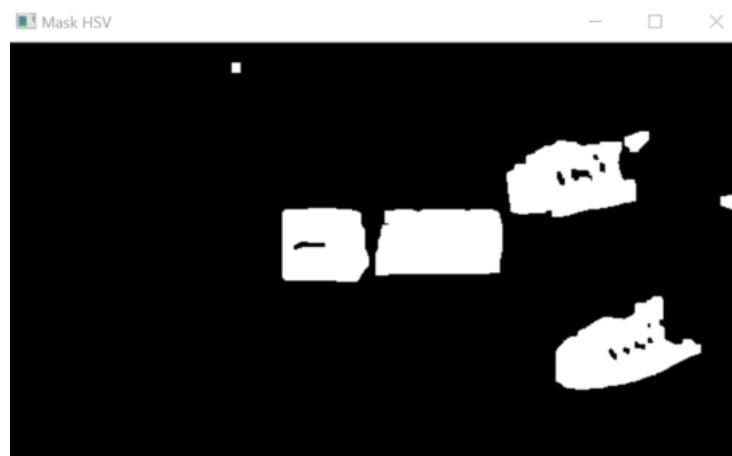


Fig 4.70: Resultado del opening

4.2.2.4 Canny

Una vez obtenida una imagen binaria y con poco ruido, llega el momento de realizar la detección de los bordes. El detector de canny es el método más fiable para realizar este proceso, aunque no el más rápido, y su función dentro de OpenCV es la siguiente:

```
void cv::Canny (InputArray image, OutputArray edges, double threshold1, double threshold2, int apertureSize=3, bool L2gradient=false)
```

Fig 4.71: Desglose de la función "Canny" de la librería OpenCV

Los threshold 1 y 2 marcan los umbrales del algoritmo de Canny, y por lo general se establecen en 10 y 40 respectivamente.

El resultado de la aplicación de canny sobre la imagen pre-procesada es el siguiente:

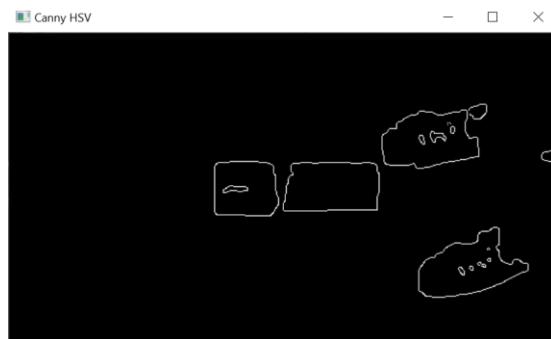


Fig 4.72: Resultado del Canny

4.2.2.5 Detección de contornos

Tras la aplicación de canny se pueden observar distintas figuras que forman contornos. Mediante la aplicación de un algoritmo creado por Satoshi Suzuki y otros, es posible enumerar los distintos contornos formados por estos bordes y almacenarlos. La función en OpenCV que realiza esta operación es la siguiente:

```
void cv::findContours (InputOutputArray image, OutputArrayOfArrays contours, int mode, int method, Point offset=Point())
```

Fig 4.73: Desglose de la función "findContours" de la librería OpenCV

Los contornos detectados se almacenan en un objeto de tipo “`<vector<vector<Point>> ...`”.

4.2.2.6 Puntos medios

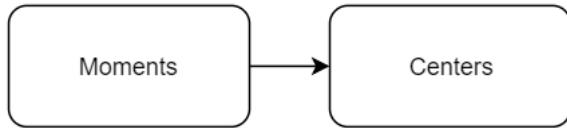


Fig 4.74: Procesos en la detección de centros

Para facilitar el trabajo con los contornos detectados, se halla su centro. Estos se obtienen siguiendo dos pasos:

1. Se crea un objeto de tipo “Moments” y se calcula el momento para cada contorno utilizando la función “moments” proporcionada por la librería OpenCV:

```
Moments cv::moments (InputArray array, bool binaryImage=false)
```

Fig 4.75: Desglose de la función "moments" de la librería OpenCV

En el array se introducen uno a uno los vectores tipo “Point” creados en el apartado 4.2.2.5.

2. Con estos momentos es posible obtener el centro del contorno. Para ello se sigue la siguiente fórmula:

$$Cx = \frac{M_{10}}{M_{00}} \quad Cy = \frac{M_{01}}{M_{00}}$$

De esta forma, tendríamos el siguiente código:

```
for (int i = 0; i < contoursHSV.size(); i++)
{
    hsvMoments = moments(contoursHSV[i], false);
    hsvCenters[i] = Point2f(hsvMoments.m10 / hsvMoments.m00, hsvMoments.m01 / hsvMoments.m00);
}
```

Fig 4.76: Código de la detección de centros

4.2.3 Detecciones finales

El objetivo final de este trabajo es obtener detecciones lo más precisas posibles. Para ello, se establece una lógica en la que un objeto será considerado como una mina solo si es detectado tanto por color como por el detector de HOG.

La detección mediante SVM+HOG nos dio como resultado un vector de rectángulos (objetos tipo “Rect” en OpenCV) que rodeaban las supuestas minas, mientras que la detección por color nos devolvió un vector de puntos (“Point2f”) que identificaban los centros de los objetos de color negro que podrían corresponderse a minas. Uniendo estos 2 vectores, es posible obtener una detección de minas fiable.

Así mismo, la detección por color se realiza en distintos espacios de color, por lo que las máscaras podrán ser intercambiadas y/o sobrepuertas para mejorar la fiabilidad del detector.

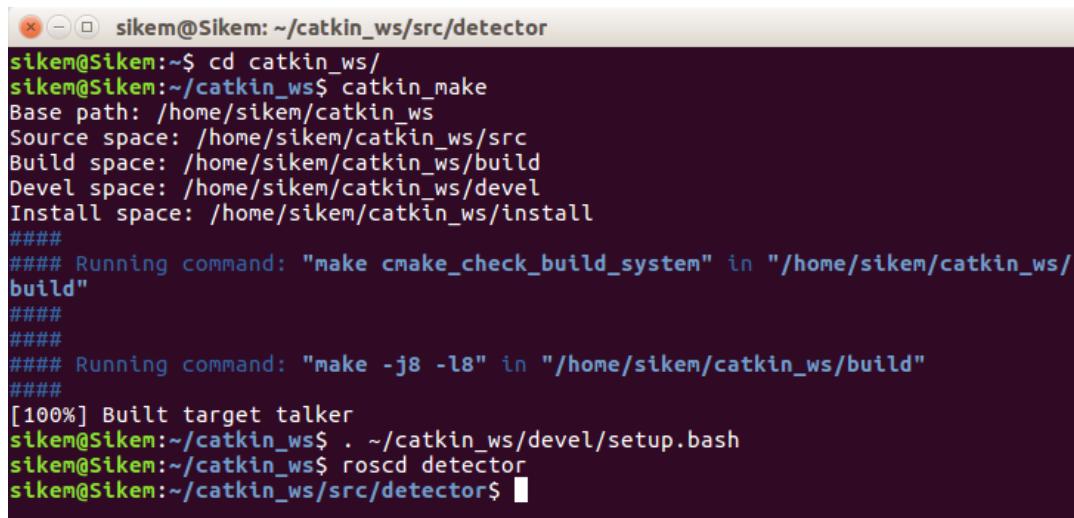
4.2.4 Implementación en ROS

4.2.4.1 Configuración inicial

El primer paso es instalar el sistema operativo robótico “ROS”, más concretamente la versión “Kinetic Kame” [40]. Al realizarse todo este proceso en un sistema operativo Linux, la instalación se lleva a cabo mediante comandos introducidos en la consola siguiendo los pasos en el tutorial citado [41].

Una vez se tienen instaladas todas las dependencias, configuraciones, y ROS en sí, se procede a crear un espacio de trabajo del tipo “catkin_ws”. Con el comando proporcionado por las librerías de ROS “catkin_make” se compila este espacio de trabajo, obteniendo un archivo “CMakeList.txt”.

Posteriormente, se utiliza la función “catkin_create_pkg” para crear un nuevo paquete al que llamaremos “detector”, asignándole dependencias de roscpp (programación en C++), std_msgs (librería básica de ROS), y rospy (programación en Python). Es necesario añadir este paquete al entorno de ROS utilizando el comando “`./catkin_ws/devel/setup.bash`”.



```
sikem@sikem:~/catkin_ws/src/detector
sikem@sikem:~$ cd catkin_ws/
sikem@sikem:~/catkin_ws$ catkin_make
Base path: /home/sikem/catkin_ws
Source space: /home/sikem/catkin_ws/src
Build space: /home/sikem/catkin_ws/build
Devel space: /home/sikem/catkin_ws/devel
Install space: /home/sikem/catkin_ws/install
#####
##### Running command: "make cmake_check_build_system" in "/home/sikem/catkin_ws/build"
#####
#####
##### Running command: "make -j8 -l8" in "/home/sikem/catkin_ws/build"
#####
[100%] Built target talker
sikem@sikem:~/catkin_ws$ . ~/catkin_ws/devel/setup.bash
sikem@sikem:~/catkin_ws$ roscl detector
sikem@sikem:~/catkin_ws/src/detector$ 
```

Fig 4.77: Compilación y acceso al pkg de ROS

Tras esto, ya dispondríamos de un paquete creado dentro de ROS, por lo que solo quedaría asignarle los códigos a compilar y configurar su CMakeList.txt .

El archivo .cpp se moverá a la siguiente carpeta:

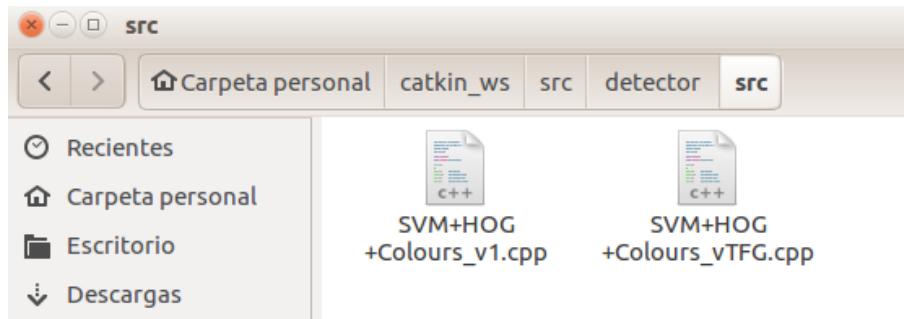


Fig 4.78: Directorio de los archivos .cpp

Por otro lado, el único cambio que hay que realizar dentro del código es incluir la siguiente librería:

```
#include "ros/ros.h"
```

Para finalizar la configuración del código, solo queda realizar modificaciones en el archivo CMakeList.txt de la siguiente ruta:

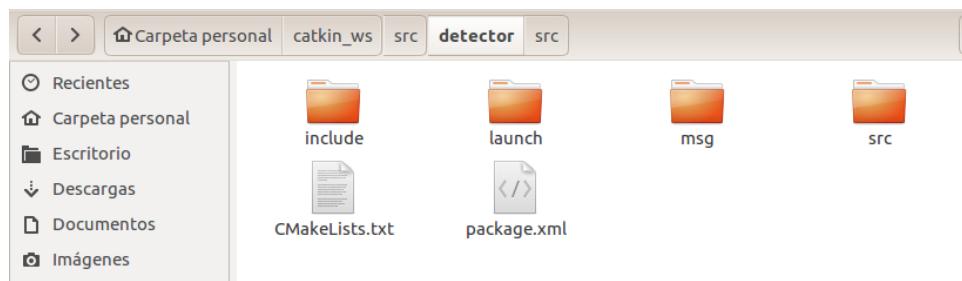


Fig 4.79: Directorio del archivo CMakeList.txt

Dentro de estas modificaciones se encuentra la inclusión de la librería OpenCV. Esta se encuentra ya implementada dentro de ROS, por lo que no es necesario realizar nada más que un “include_directories” para que sea funcional.

```
add_executable(talker src/SVM+HOG+Colours_v1.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
#target_link_libraries(talker ${OpenCV_LIBRARIES})
add_dependencies(talker detector_generate_messages_cpp)

include_directories(${OpenCV_INCLUDE_DIRS})
target_link_libraries(talker ${OpenCV_LIBRARIES})
```

Fig 4.80: Configuración del archivo CMakeList.txt

4.2.4.2 Compilación y ejecución

A la hora de compilar un código en ROS, el primer paso es inicializar “roscore”.

```
sikem@sikem:~$ roscore
... logging to /home/sikem/.ros/log/634522f4-a213-11e8-ac41-54271e1e5f81/roslaunch-Sikem-12706.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://Sikem:32927/
ros_comm version 1.12.13

SUMMARY
=====

PARAMETERS
* /rosdistro: kinetic
* /rosversion: 1.12.13

NODES

auto-starting new master
process[master]: started with pid [12722]
ROS_MASTER_URI=http://Sikem:11311/

setting /run_id to 634522f4-a213-11e8-ac41-54271e1e5f81
process[rosout-1]: started with pid [12735]
started core service [/rosout]
```

Fig 4.81: Inicialización de "roscore"

Esta ventana se deja en un segundo plano y se abre una nueva ventana de comandos. Es necesario tener “roscore” activo siempre que se quiera ejecutar un programa.

En la nueva ventana se realiza la llamada al paquete creado en el anterior apartado. Es posible que al reiniciar el ordenador se pierda la configuración, por la que se suele recomendar empezar repitiendo el comando “`./catkin_ws/devel/setup.bash`” .

Para acceder directamente al directorio de un paquete, solo es necesario saber su nombre. Así, con el comando “`roscd`” seguido de este nombre accedemos directamente a su directorio.

Una vez completado todo lo dicho anteriormente, solo queda ejecutar el programa. Se sigue el siguiente comando:

```
$ rosrun [package_name] [node_name]
```

Fig 4.82: Código para la ejecución del programa de ROS

Siendo:

- package_name: nombre del paquete, en nuestro caso “detector”
- node_name: nombre que se le asigna al archivo .cpp dentro del CMakeList.txt modificado, en nuestro caso “detect”.

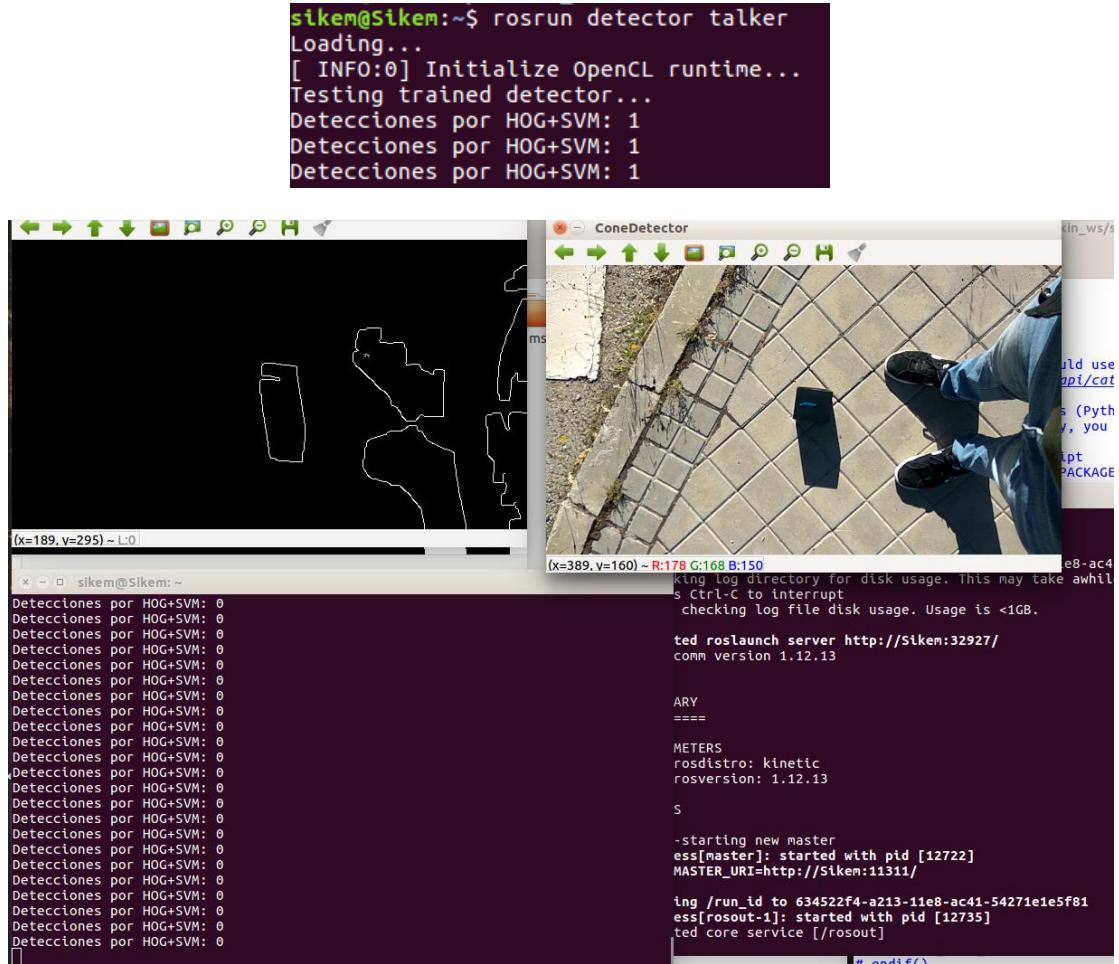


Fig 4.83: Resultado final de la detección utilizando ROS

5 RESULTADOS

A continuación, se analizarán los resultados obtenidos con los diferentes detectores entrenados.

La eficacia de estos detectores se determinará utilizando una “matriz de confusión”, una herramienta empleada en “machine learning” que permite obtener la precisión, el ratio de positivos, etc. [42].

5.1 Matriz de confusión

La matriz de confusión nos permite representar de forma clara los resultados de una detección por visión artificial frente a la realidad del objeto que se desea detectar. Sigue la siguiente estructura:

		PREDICIONES	
		NO MINA	MINA
REALIDAD	NO MINA	A	B
	MINA	C	D

Tabla 1: Estructura de una matriz de confusión

En predicciones se indica el resultado que proporciona el detector, y en realidad el resultado real. Los cuadrantes A, B, C y D indican lo siguiente:

- A: La predicción corresponde con la realidad, el objeto detectado no es una mina.
- B: La predicción es errónea, indica que existe una mina cuando en realidad no lo es. Es un falso positivo.
- C: La predicción es errónea, el detector no detecta correctamente una mina.
- D: Predicción acertada, una mina es seleccionada correctamente por el detector. Es un positivo.

A partir de estos datos se pueden hallar los siguientes resultados:

- Exactitud

$$AC = \frac{\text{Predicciones Correctas}}{\text{Número total de muestras}} = \frac{A + D}{A + B + C + D}$$

Fig 5.1: Fórmula para obtener la exactitud

Es la proporción del número total de predicciones correctas frente a todas las predicciones realizadas.

- Ratio de positivos

$$RP = \frac{\text{Minas detectadas correctamente}}{\text{Muestras de minas}} = \frac{D}{C + D}$$

Fig 5.2: Fórmula para obtener el ratio de positivos

Proporción de minas correctamente identificadas frente a todas las muestras de minas.

- Ratio de falsos positivos

$$RFP = \frac{\text{Minas detectadas incorrectamente}}{\text{Muestras sin minas}} = \frac{B}{A + B}$$

Fig 5.3: Fórmula para obtener el ratio de falsos positivos

Proporción de falsos positivos frente a todas las muestras sin minas.

- Ratio de negativos

$$RN = \frac{\text{Minas no detectadas correctamente}}{\text{Muestras sin minas}} = \frac{A}{A + B}$$

Fig 5.4: Fórmula para obtener el ratio de negativos

Proporción de predicciones negativas correctas frente a todas las muestras sin minas

- Ratio de falsos negativos

$$RFN = \frac{\text{Minas no detectadas erroneamente}}{\text{Muestras de minas}} = \frac{C}{C + D}$$

Fig 5.5: Fórmula para obtener el ratio de falsos negativos

Ratio de las imágenes que no fueron detectadas como minas aun siéndolo.

- Precisión

$$P = \frac{\text{Minas detectadas correctamente}}{\text{Total predicciones de minas}} = \frac{D}{B + D}$$

Fig 5.6: Fórmula para obtener la precisión

La precisión indica la calidad del detector mostrando el porcentaje de aciertos.

Con los valores de exactitud, precisión, RP y RN de cada detector, es posible analizar la calidad de cada detector.

5.2 Análisis de resultados

En el caso de este proyecto, al utilizarse vídeos para el testeo de los detectores resulta muy difícil hallar la cantidad de imágenes con minas y sin minas reales. Por ello, se realizarán dos tipos de análisis:

- Precisión del detector: siguiendo la fórmula de la matriz de confusión. Las minas detectadas correctamente serán aquellas que cumplen los 2 requisitos explicados anteriormente: detección por SVM+HOG y detección por color. Las minas no detectadas correctamente serán aquellas que se detecten mediante SVM+HOG pero no mediante color.
- Eficacia del detector: se realizará mediante la comprobación visual del video, de forma “humana”. Si un detector no funciona correctamente, no detectará ni recuadrará las minas.
- Resultado general: se analizará de forma objetiva el resultado teniendo en cuenta su tipo de entrenamiento y su resultado.

Se repetirá este proceso para cada detector entrenado, indicando el tipo de cada uno y el número de imágenes usadas en su entrenamiento. Para el testeo se utilizarán distintos videos, dependiendo del archivo.

Training ID	Number of samples	Mask Size	Flipped	Training file
1	35	32	N	mineDetector_32_N.yml
2	35	64	N	mineDetector_64_N.yml
3	35	32	Y	mineDetector_32_R.yml
4	35	64	Y	mineDetector_64_R.yml
5	354	32	Y	mineDetector_32_R_v2.yml
6	354	64	Y	mineDetector_64_R_v2.yml
7	463	32	Y	mineDetector_32_R_v3.yml
8	463	64	Y	mineDetector_64_R_v3.yml
9	629	32	Y	mineDetector_32_R_v4.yml
10	629	64	Y	mineDetector_64_R_v4.yml
11	673	32	Y	mineDetector_32_R_v5.yml
12	726	32	Y	mineDetector_32_R_v6.yml

Tabla 2: Detectores entrenados

5.2.1 Pruebas 1-6

Se comenzará las pruebas con un video sobre un terreno de fácil contraste con el negro, y se ejecutarán las pruebas iniciales con detectores creados a partir de una base de datos pequeña. Se observarán las diferencias entre los detectores de 32x32 y los de 64x64.

5.2.1.1 Prueba 1

El primer detector entrenado es del tipo 32x32 (tamaño de máscara), sin rotación de las imágenes de entrenamiento. Es el resultado de un entrenamiento con tan solo 35 imágenes. Su resultado en el video utilizado para el testeo es el siguiente:

```

Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamientos/mineDetector_32_N.yml
Número total de falsas detecciones: 0
Número total de detecciones positivas: 0

```

Fig 5.7: Detecciones para el detector 1

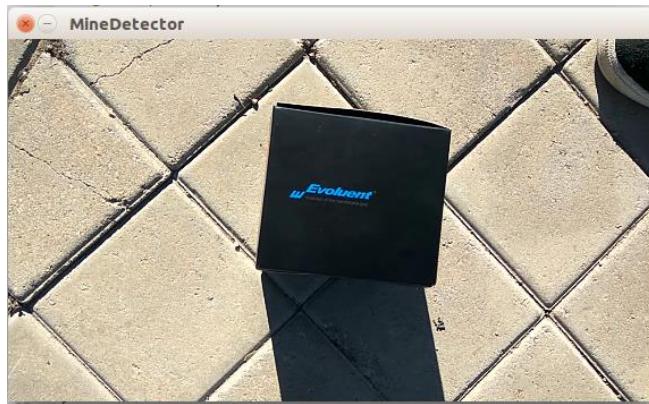


Fig 5.8: Imagen final detector 1

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
0	0	0

$$\text{Precisión} = \frac{\# \text{Detecciones Positivas}}{\# \text{Imágenes}}$$

Tabla 3: Precisión detector 1

Al no obtener ningún tipo de detección, no es posible calcular su precisión.

Eficacia: Su eficacia es nula, muy posiblemente por el uso de solo 35 imágenes.

Resultado general: Con tan solo 35 imágenes era de esperar que el resultado fuese bajo, pero en este caso el principal error es el tamaño de la máscara. En el video la caja aparece relativamente cerca, por lo que máscaras de 32x32 no consiguen abarcar todo el recuadro de la caja en ningún momento.

5.2.1.2 Prueba 2

Del tipo 64x64, sin rotación en las imágenes, y con una base de datos de tan solo 35 imágenes.

```
Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamiento/mineDetector_64_N.yml
Número total de falsas detecciones: 418
Número total de detecciones positivas: 832
```

Fig 5.9: Resultados para el detector 2



Fig 5.10: Imagen final detector 2

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
418	832	1250

$$\text{Precisión} = 0,6656$$

Tabla 4: Precisión detector 2

Tiene una precisión del 66,6%

Eficacia: no funciona eficazmente con las detecciones a amplias distancias.

Resultado general: teniendo en cuenta el bajo número de imágenes con el que ha sido entrenado, su resultado es superior al esperado. Sin embargo, la eficacia es baja a distancias amplias, lo cual es necesario mejorar.

5.2.1.3 Prueba 3

Máscara de 32x32 píxeles, con una base de datos de 35 imágenes duplicada gracias a la aplicación de una rotación de 90° a cada imagen. Total de 70 imágenes.

```

Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamiento/mineDetector_32_R.yml
Número total de falsas detecciones: 0
Número total de detecciones positivas: 1

```

Fig 5.11: Resultados para el detector 3



Fig 5.12: Imagen final detector 3

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
0	1	1

$$\text{Precisión} = \frac{1}{1} = 1$$

Tabla 5: Precisión detector 3

Al igual que en el caso del detector 1, la precisión es redundante debido a su baja eficacia.

Eficacia: nula, no detecta la caja.

Resultado general: al igual que en el detector 1, la máscara de 32x32 no sirve en este video porque la caja está demasiado cerca. Pese a ello, es necesario comprobar de nuevo este tipo de detectores en otros videos y con una base de datos mayor.

5.2.1.4 Prueba 4

Máscara de 64x64, entrenado con una base de datos de 35 imágenes que son volteadas 90°.

```
x - Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamiento/mineDetector_64_R.yml
Número total de falsas detecciones: 434
Número total de detecciones positivas: 842
```

Fig 5.13: Resultados para el detector 4



Fig 5.14: Imagen final detector 4

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
434	842	1276

$$\text{Precisión} = 0,659874608$$

Tabla 6: Precisión detector 4

La precisión empeora un poco respecto a su versión sin imágenes rotadas, pero es casi inapreciable.

Eficacia: muy parecida a la versión “2”, captura bien la caja a distancias cercanas-medias pero no a lo lejos.

Resultado general: no se nota una mejora importante frente a la versión con imágenes no rotadas, pero tampoco se observa lo contrario. Sigue fallando en distancias lejanas.

5.2.1.5 Prueba 5

Máscara de 32x32, con una base de datos de 354 imágenes que se duplicarán con un giro de 90°.

```
Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamientos/mineDetector_32_R_v2.yml
Número total de falsas detecciones: 0
Número total de detecciones positivas: 3
```

Fig 5.15: Resultados para el detector 5

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
0	3	3

$$\text{Precisión} = \frac{3}{3} = 1$$

Tabla 7: Precisión detector 5

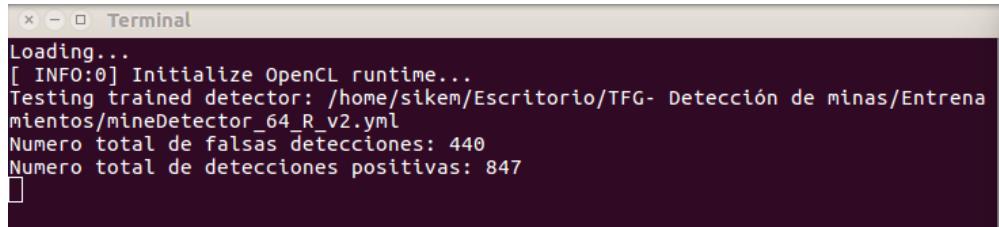
Los errores de las anteriores versiones persisten.

Eficacia: Nula

Resultado general: se llega a la conclusión de que es necesario utilizar un nuevo video en el que se pueda apreciar la efectividad de los detectores de 32x32.

5.2.1.6 Prueba 6

Tamaño 64x64, con una amplia diferencia de imágenes de entrenamiento respecto al anterior detector de tamaño 64x64.



A screenshot of a terminal window titled "Terminal". The window displays the following text output:
Loading...
[INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamientos/mineDetector_64_R_v2.yml
Número total de falsas detecciones: 440
Número total de detecciones positivas: 847
[]

Fig 5.16: Resultados para el detector 6

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
440	847	1287

$$\text{Precisión} = \frac{847}{1287} = 0,658119658$$

Tabla 8: Precisión detector 6

No se denota ninguna mejora respecto al anterior detector

Eficacia: Al igual que en la precisión, no se observa ninguna mejora importante.

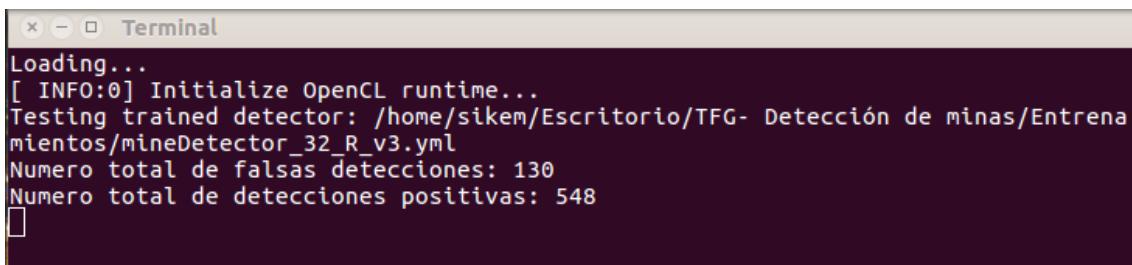
Resultado general: es necesario aumentar mucho mas la base de datos para obtener resultados mejores.

5.2.2 Pruebas 7 – 10

Para comprobar la eficacia y precisión de los detectores de 32x32, se cambia el video de prueba. Este video se realiza sobre césped, el entorno en el que se realizará la competición, y se alejará la cámara de la mina a una altura de más de 2 metros . Así mismo, se incrementará la base de datos de entrenamiento.

5.2.2.1 Prueba 7

Se añaden al entrenamiento alrededor de 100 imágenes realizadas sobre el nuevo entorno, y se utiliza una máscara de 32x32.



```
Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamientos/mineDetector_32_R_v3.yml
Número total de falsas detecciones: 130
Número total de detecciones positivas: 548
```

Fig 5.17: Resultados para el detector 7



Fig 5.18: Imagen final detector 7

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
130	548	678

$$\text{Precisión} = 0,808259587$$

Tabla 9: Precisión detector 7

Se observa a simple vista la gran diferencia respecto a los anteriores detectores de tamaño 32x32. Una eficacia de un 80%, teniendo en cuenta que se complementa con la detección por color, es un resultado bastante bueno.

Eficacia: mejora mucho respecto a sus predecesores, pero sigue siendo insuficiente.

Resultado general: como se comentó en experimentos anteriores, el detector de 32x32 adquiere fuerza a la hora de realizar detecciones lejanas debido a su tamaño.

5.2.2.2 Prueba 8

```
x - □ Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamiento/mineDetector_64_R_v3.yml
Número total de falsas detecciones: 75
Número total de detecciones positivas: 690
□
```

Fig 5.19: Resultados para el detector 8

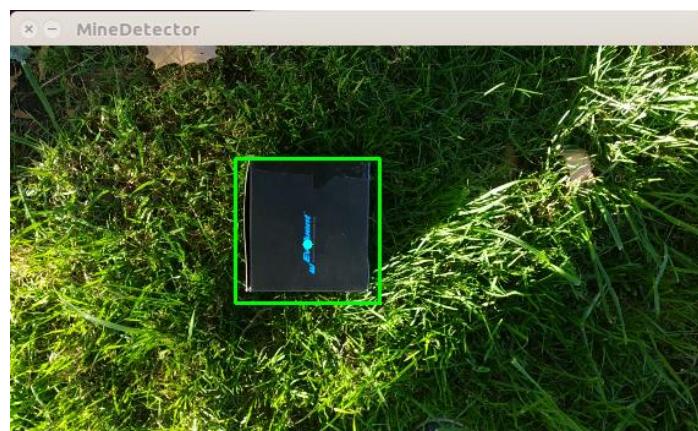


Fig 5.20: Imagen final detector 8

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
75	690	765

$$\text{Precisión} = 0,901960784$$

Tabla 10: Precisión detector 8

Con un 90% de precisión, este detector se pone en cabeza y aporta mucha seguridad en las detecciones.

Eficacia: en distancias medias-cercanas es casi perfecto, pero flojea cuando las distancias aumentan.

Resultado general: la precisión aumenta de un 65% hasta un 90%, pero no se puede comparar la eficacia porque se utilizan distintos videos en las detecciones.

5.2.2.3 Prueba 9

Detector de tamaño 32x32, con una base de datos de 629 imágenes, algunas de ellas con peor calidad al haberse tomado a distancias mas lejanas.

```
Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamientos/mineDetector_32_R_v4.yml
Número total de falsas detecciones: 1609
Número total de detecciones positivas: 1550
```

Fig 5.21: Resultados para el detector 9

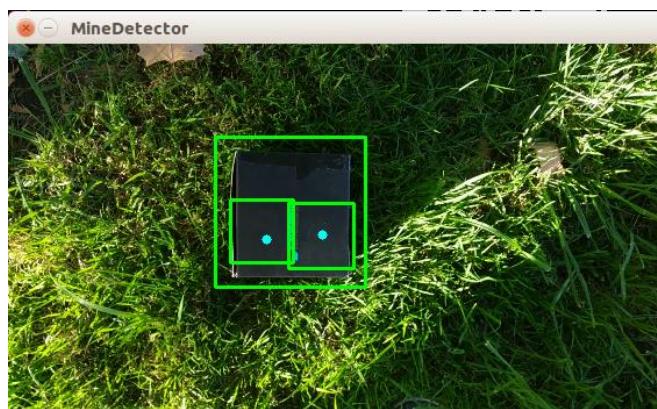


Fig 5.22: Imagen final detector 9

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
1609	1550	3159

$$\text{Precisión} = 0,490661602$$

Tabla 11: Precisión detector 9

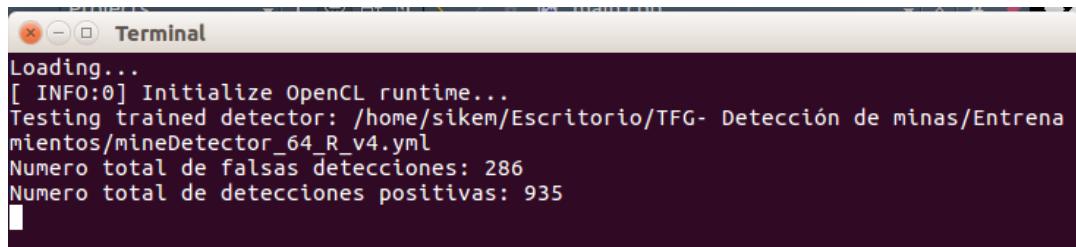
Con una precisión de 49%, este detector flojea respecto a su antecesor. En gran parte se debe a un aumento de su sensibilidad, debido a la inclusión de imágenes de peor calidad en la base de datos.

Eficacia: con un total de 1550 detecciones, es el detector que mejor detecta la mina. Se observan también múltiples detecciones sobre un mismo objeto, pero no predominan sobre los resultados generales.

Resultado general: pese a su baja precisión, este detector detecta la mina en casi toda la totalidad del video. Trabajando conjuntamente con un detector por color correcto, puede convertirse en una opción viable.

5.2.2.4 Prueba 10

Entrenado con 629 imágenes de un tamaño 64x64.



```
Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamientos/mineDetector_64_R_v4.yml
Número total de falsas detecciones: 286
Número total de detecciones positivas: 935
```

Fig 5.23: Resultados para el detector 10

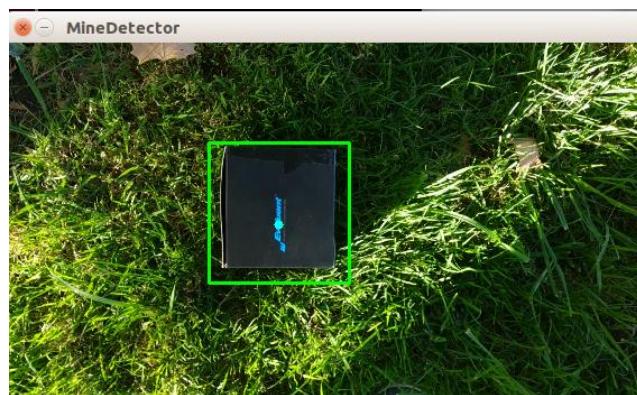


Fig 5.24: Imagen final detector 10

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
286	935	1221

$$\text{Precisión} = 0,765765766$$

Tabla 12: Precisión detector 10

Precisión de 76%, un 14% inferior a su anterior versión.

Eficacia: Con 935 detecciones, este detector aumenta considerablemente su eficacia.

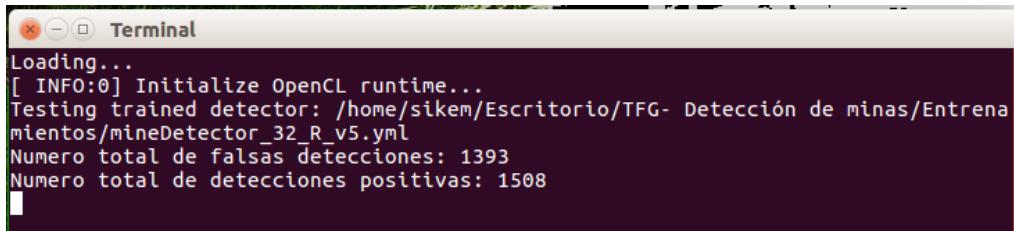
Resultado general: Pese a su empeoramiento en lo relativo a la precisión, el detector numero 10 mejora en eficacia. El motivo de esto puede ser la inclusión de imágenes menos claras en la base de datos.

5.2.3 Pruebas finales

Se ha comprobado que los detectores que mejor funcionan a distancias lejanas son los de tamaño 32x32, por ello, se crean dos detectores de tamaño 32x32 aumentando la base de datos.

5.2.3.1 Prueba 11

32x32, entrenado con una base de datos de 673 elementos de los cuales ninguno corresponde al video de prueba. Estas imágenes se voltean 90°.



```
Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamientos/mineDetector_32_R_v5.yml
Número total de falsas detecciones: 1393
Número total de detecciones positivas: 1508
```

Fig 5.25: Resultados para el detector 11

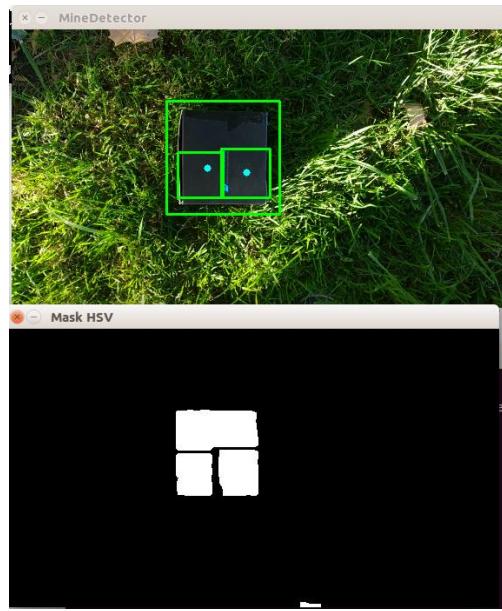


Fig 5.26: Imagen final, estándar y con HSV, para el detector 11

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
1393	1508	2901

$$\text{Precisión} = 0,519820751$$

Tabla 13: Precisión detector 11

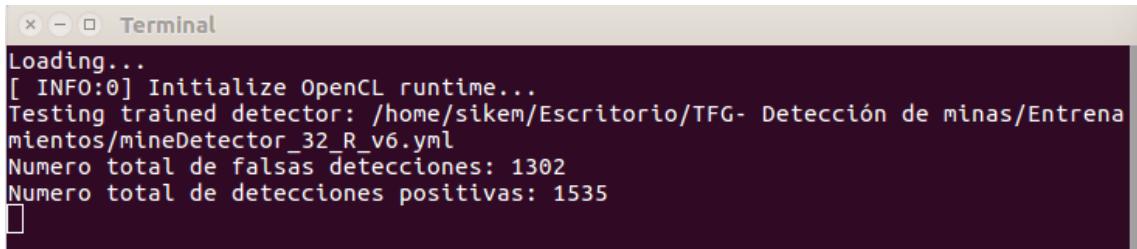
La precisión sigue siendo baja, pero se compensa con su eficacia y se soluciona con la fusión con la detección por color.

Eficacia: las detecciones perdidas respecto a su predecesor corresponden a dobles detecciones, el detector sigue siendo perfecto.

Resultado general: es muy efectivo, y mejora un poco respecto a la anterior versión de 32x32, pero sigue siendo mejorable.

5.2.3.2 Prueba 12

Como el anterior, 32x32, pero en este caso con una base de datos de 726 imágenes entre las que existen imágenes del video de prueba.



```

x - Terminal
Loading...
[ INFO:0] Initialize OpenCL runtime...
Testing trained detector: /home/sikem/Escritorio/TFG- Detección de minas/Entrenamientos/mineDetector_32_R_v6.yml
Número total de falsas detecciones: 1302
Número total de detecciones positivas: 1535

```

Fig 5.27: Resultados para el detector 12

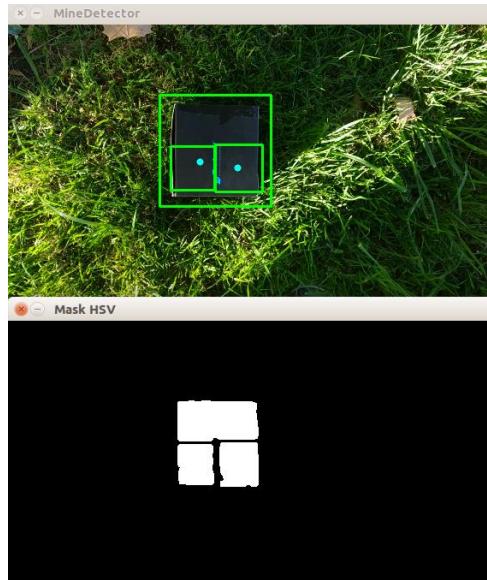


Fig 5.28: Imagen final, estándar y HSV, para el detector 12

Precisión:

Falsas detecciones	Minas Color+SVM	Detecciones Totales
1302	1535	2837

$$\text{Precisión} = 0,541064505$$

Tabla 14: Precisión detector 12

Gracias a la inclusión de imágenes del video, se mejora su precisión disminuyendo las falsas detecciones.

Eficacia: aparecen de nuevo dobles detecciones, pero esto no supone un gran problema para el detector.

Resultado general: se observa como con una pequeña inclusión de imágenes a la base de datos, el detector presenta mejores resultados. Se concluye que, pese a ser este detector totalmente válido para su propósito, si fuese necesario sería posible mejorarlo incluyendo nuevas imágenes, tanto positivas como negativas, a la base de datos.

6 CONCLUSIONES Y TRABAJOS FUTUROS

6.1 Conclusiones

Una vez concluido el proyecto, se pueden abordar distintos temas relativos a todos los estudios, procesos y resultados de este.

Para comenzar, la base de todo este proyecto es la librería OpenCV. Esta librería, en constante desarrollo, proporciona todas las funciones necesarias para la realización de proyectos de visión artificial. Su documentación es clara y completa, y dispone de compatibilidad con todos los sistemas operativos. Existen alternativas, pero en un ámbito no profesional no tiene rival.

Las detecciones basadas en el color de los objetos constan de muchos puntos negativos, siendo uno de ellos el efecto que puede tener la iluminación del entorno sobre la apariencia de estos colores. Este problema se resuelve mediante el uso de distintos espacios de color, principalmente el HSV, lo que aporta fiabilidad a este tipo de detecciones. Así, se deja abierta la opción de utilizar otros espacios de color como el Lab y el YCrCb, pero en este proyecto no se aplican del todo por resultar redundantes al lado del HSV. El resultado de la detección mediante color (espacio HSV) en el video de prueba es muy satisfactorio.

Por otro lado se encuentra la detección mediante el histograma de gradientes orientados y la máquina de soporte vectorial. Este tipo de detecciones basan su eficacia en el tipo de objeto a detectar y la base de datos utilizada. Al ser las minas cuadrados sin ningún tipo de detalle distintivo en su interior, su detección se complica por la existencia de muchos objetos con esta misma forma en cualquier lugar. Sin embargo, se cuenta que este prototipo será utilizado en una competición dentro de un entorno controlado, por lo que este tipo de detecciones es válido.

En el entrenamiento de SVMs para detección se concluye que, pese a los aparentes mejores resultados de las máscaras de 64x64, los mejores detectores son aquellos de tamaño 32x32. Mientras que los primeros son mucho más fiables en distancias cercanas, los segundos los superan ampliamente en distancias lejanas e igualan su eficacia en distancias cercanas.

El principal problema de los filtros de tamaño 32x32 es su precisión en distancias cercanas: detectan la mina sin problema, pero también tienen muchos falsos positivos. Sin embargo, este problema se resuelve mediante la fusión de las detecciones por color y las detecciones por SVM+HOG aplicada en el proyecto. Por ello, se le da más importancia a la eficacia de un detector (si detecta correctamente la mina), que a su precisión (la relación entre detecciones correctas y falsos positivos).

Para finalizar, es necesario hablar de los detectores creados en este proyecto. Las distintas versiones tienen diferentes características, pero el detector que cumple mejor todos los requisitos propuestos en la competición es el número 12. Durante los videos de prueba, este detector detecta constantemente la mina. Sus falsos positivos son solventados mediante la detección por color, y las dobles detecciones no suponen un problema en el cumplimiento de los objetivos propuestos al comienzo de este proyecto.

6.2 Trabajos futuros

Aunque el campo de la visión artificial se inició hace décadas, no ha sido hasta hace relativamente poco cuando este ha comenzado a adquirir fuerza gracias a las innovaciones en GPU's, y otros tipos de hardware; lo que lleva a que en la actualidad no esté muy desarrollado. Esto trae dificultades a la hora de pensar en trabajos futuros, ya que el sector evoluciona muy rápidamente y es casi imposible adivinar que mejoras llegarán en los próximos años.

Aun así, sí que existen una serie de propuestas que podrían aplicarse actualmente para conseguir mejoras en la efectividad, los requisitos mínimos para la misma, etc, del proyecto:

- Aumentar el número de imágenes utilizadas como base de datos a la hora de entrenar el detector SVM+HOG, tanto positivas como negativas.
- Optimizar la base de datos utilizada eliminando imágenes que puedan producir errores en el detector por falta de resolución, poca claridad, ...
- Realizar esta base de datos con imágenes tomadas del objeto concreto a detectar, es decir, la caja que se usará en la competición.
- Añadir más detecciones por color utilizando otros espacios de color.
- Optimizar las detecciones por color ya implementadas basándonos en las condiciones que existirán en el torno donde se realizará la competición para ver si es necesario ser más o menos restrictivo con los rangos. Esto puede variar dependiendo de la iluminación y de los posibles obstáculos que puedan confundir al detector por su color.
- Utilizar métodos de detección alternativos a SVM+HOG, como pueden ser los descriptores “SURF” o los “Cascade Classifiers”.

- Implementación de DEEP LEARNING. Este método de detección es el que más fuerza está adquiriendo en la actualidad, sin embargo trae consigo bastantes dificultades: necesidad de realizar una base de datos mucho mayor, complejidad a la hora de detectar una simple caja negra por lo usual que es en cualquier tipo de entorno/objeto, y su dificultad de implementación directamente en la librería OpenCV por no estar todavía optimizado. Así mismo, el Deep learning supone un coste computacional muy alto y podría producir errores en la detección del dron.

7 BIBLIOGRAFÍA

- [1] “Minesweepers, towards a landmine-free world”. [Acceso: agosto 2018]. Disponible en: <http://www.landminefree.org/wp-content/uploads/2018/05/Minesweepers-2018-CFP-ES.pdf>
- [2] “Mina antipersona”, *Wikipedia*. [Acceso: agosto 2018]. Disponible en: https://es.wikipedia.org/wiki/Mina_antipersona
- [3] “¿En qué países siguen matando las minas antipersona?”, *Público*, 04-04-2016. [Acceso: agosto 2018]. Disponible en: <https://www.publico.es/internacional/paises-siguen-matando-minas-antipersona.html>
- [4] “International Campaign to Ban Landmines”. [Acceso: agosto 2018]. Disponible en: <http://www.icbl.org/en-gb/about-us.aspx>
- [5] “International Campaign to Ban Landmines”, *Wikipedia*. [Acceso: agosto 2018]. Disponible en: https://en.wikipedia.org/wiki/International_Campaign_to_Ban_Landmines
- [6] “CAPECON Project”, *Wikipedia*. [Acceso: agosto 2018]. Disponible en: https://en.wikipedia.org/wiki/CAPECON_project
- [7] “Unnnamed Aerial Vehicle”, *Wikipedia*. [Acceso: agosto 2018]. Disponible en: https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle
- [8] “List of unmanned aerial vehicles”, *Wikipedia*. [Acceso: agosto 2018]. Disponible en: https://en.wikipedia.org/wiki/List_of_unmanned_aerial_vehicle_applications
- [9] “Demining”, *Wikipedia*. [Acceso: agosto 2018]. Disponible en: <https://en.wikipedia.org/wiki/Demining>
- [10] “Imaging drones to spot signs of explosive chemicals leaking from landmines”, *NewAtlas*, 11-04-2016. [Acceso: agosto 2018]. Disponible en: <https://newatlas.com/drones-landmines-bristol/42732/>
- [11] “Mine Kafon Drone”, *KickStarter*. [Acceso: agosto 2018]. Disponible en: <https://www.kickstarter.com/projects/massoudhassani/mine-kafon-drone>
- [12] “OpenCV”. [Acceso: febrero 2018]. Disponible en: <https://opencv.org/>
- [13] “OpenCV”, *Wikipedia*. [Acceso: febrero 2018]. Disponible en: <https://en.wikipedia.org/wiki/OpenCV>
- [14] “About ROS”, *ROS*. [Acceso: junio 2018]. Disponible en: <http://www.ros.org/about-ros/>

- [15] “History of ROS”, *ROS*. [Acceso: junio 2018]. Disponible en: <http://www.ros.org/history/>
- [16] “Histogram of oriented gradients”, *Wikipedia*. [Acceso: enero 2018]. Disponible en: https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
- [17] Satya Mallick, “Histogram of Oriented Gradients”, *Learn OpenCV*, 06-12-2016. [Acceso: febrero 2018]. Disponible en: <https://www.learnopencv.com/histogram-of-oriented-gradients/>
- [18] Chris McCormick, “HOG Person Detector Tutorial”, *mccormickml*, 09-05-2013. [Acceso: febrero 2018]. Disponible en: <http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/>
- [19] “Histogram of Oriented Gradients”, *Scikit-Image*. [Acceso: febrero 2018]. Disponible en: http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html
- [20] Dalal, N. and Triggs, B., “Histograms of Oriented Gradients for Human Detection”, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.
- [21] David G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol 60, pp. 91-110, feb.. 2004.
- [22] Gustavo A.Betancourt, “Las máquinas de soporte vectorial (SVMs)”, *Scientia et Technica*, vol. 1, n.º 27, Abr. 2005. [Acceso: febrero 2018]. Disponible en: <http://revistas.utp.edu.co/index.php/revistaciencia/article/view/6895> . Acceso: 16-09-2018
- [23] Enrique J. Carmona Suárez, “Tutorial sobre Máquinas de Vectores Soporte (SVM)”, Tutorial, Dpto. de Inteligencia Artificial, Universidad Nacional de Educación a Distancia (UNED), Madrid, España, 2014. [Acceso: febrero 2018]. Disponible en: [http://www.ia.uned.es/~ejcarmona/publicaciones/\[2013-Carmona\]%20SVM.pdfc](http://www.ia.uned.es/~ejcarmona/publicaciones/[2013-Carmona]%20SVM.pdfc)
- [24] Vikas Gupta, “Color spaces in OpenCV (C++/ Python)”, *Learn OpenCV*, 07-05-2017. [Acceso: abril 2018]. Disponible en: <https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/>
- [25] “YCbCr”, *Wikipedia*. [Acceso: abril 2018]. Disponible en: <https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/>
- [26] “Espacio de color Lab”, *Wikipedia*. [Acceso: abril 2018]. Disponible en: https://es.wikipedia.org/wiki/Espacio_de_color_Lab
- [27] “Morphological Transformations”, *opencv.org*. [Acceso: abril 2018]. Disponible en: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

- [28] Jorge Valverde Rebaza, “Detección de bordes mediante el algoritmo de Canny”, Artículo, Escuela Académico Profesional de Informática, Universidad Nacional de Trujillo, 2007. [Acceso: abril 2018]. Disponible en: [https://www.researchgate.net/publication/267240432 Deteccion de bordes mediante el algoritmo de Canny](https://www.researchgate.net/publication/267240432_Deteccion_de_bordes_mediante_el_algoritmo_de_Canny)
- [29] “Canny Edge Detector”, *opencv.org*. [Acceso: abril 2018]. Disponible en: https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html
- [30] “HOG parameters in OpenCV Java Version”, *StackOverflow*, 03-07-2014. [Acceso: febrero 2018]. Disponible en: <https://stackoverflow.com/questions/24560626/hog-parameters-in-opencv-java-version>
- [31] “Introduction to Support Vector Machines”, *opencv.org*. [Acceso: febrero 2018]. Disponible en: https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
- [32] “using OpenCV and SVM with images”, *StackOverflow*, 04-02-2013. [Acceso: marzo 2018]. Disponible en: <https://stackoverflow.com/questions/14694810/using-opencv-and-svm-with-images>
- [33] “How to create new TrainData object”, *StackOverflow*, 30-01-2018. [Acceso: marzo 2018]. Disponible en: <https://stackoverflow.com/questions/48520856/how-to-create-new-traindata-object>
- [34] “SVM Class Reference”, *opencv.org*. [Acceso: marzo 2018]. Disponible en: https://docs.opencv.org/3.3.1/d1/d2d/classcv_1_1ml_1_1SVM.html
- [35] “How to import a trained HOG detector”, *nvidia.com*, 30-05-2017. [Acceso: marzo 2018]. Disponible en: <https://devtalk.nvidia.com/default/topic/1010988/how-to-import-a-trained-hog-detector/>
- [36] Adrian Rosebrock, “HOG detectMultiScale parameters explained”, *pyimagesearch*, 16-11-2015. [Acceso: febrero 2018]. Disponible en: <https://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained>
- [37] “Improving accuracy OpenCV HOG people detector”, *StackOverflow*, 28-10-2014. [Acceso: marzo 2018]. Disponible en: <https://stackoverflow.com/questions/26607418/improving-accuracy-opencv-hog-people-detector>
- [38] “Deep Neural Network Module”, *opencv.org*. [Acceso: marzo 2018]. Disponible en: https://docs.opencv.org/master/d6/d0f/group_dnn.html#ga9d118d70a1659af729d01b10233213ee

- [39] “Color conversions”, *opencv.org*. [Acceso: abril 2018]. Disponible en: https://docs.opencv.org/3.3.0/de/d25/imgproc_color_conversions.html
- [40] “Installing and Configuring Your ROS Environment”, *wiki.ros.org*. [Acceso: julio 2018]. Disponible en: <http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment>
- [41] “Ubuntu install of ROS Kinetic”, *wiki.ros.org*. [Acceso: julio 2018]. Disponible en: <http://wiki.ros.org/kinetic/Installation/Ubuntu>
- [42] “Confusion Matrix”, *uregina*. [Acceso: agosto 2018]. Disponible en: http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html