Instituto Tecnológico de Buenos Aires - ITBA
Escuela de Ingeniería y Gestión

Caracterización de un sistema de robots autopropulsados

Autores

Arco, Martina. 57091
Lavarello, Agustín. 57046
Palacci, Julián Franco. 57157

Tutor

Patterson, Germán Agustín.

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN INFORMÁTICA

Buenos Aires
30 de Julio de 2020

Resumen

Los kilobots son robots diseñados para la simulación de partículas autopropulsadas en un entorno determinado. Utilizan motores de vibración para su movimiento y componentes LED para comunicar comportamiento.
En este trabajo se analizaron videos de un grupo de kilobots interactuando dentro de un cuadrado de madera blanco. Para esto, se desarrolló una herramienta en Python que utiliza distintos algoritmos para poder determinar la posición y el color de la luz LED de los kilobots. En el caso de la detección del color de la luz LED, se utilizó un algoritmo genético. Este evoluciona para hallar la máscara que determine el color de forma más exacta.
Además, se realizaron diferentes pruebas para evaluar la eficacia de los algoritmos utilizados.

Índice

1. Justificación del trabajo **3**

2. Estado del Arte **4**

3. Objetivos **4**

4. Planificación del proyecto **4**

5. Requerimientos **5**
5.1. Requerimientos funcionales 5
5.2. Requerimientos no funcionales 5

6. Diseño **6**
6.1. Arquitectura 8
6.1.1. Servicios 8
6.1.2. Modelos 8

7. Datos de implementación **8**
7.1. Input 8
7.2. Output 11
7.3. Servicio de calibración de cámara 11
7.4. Algoritmos utilizados 11
7.4.1. Detección de los robots y de los tornillos 11
7.4.2. Detección de cuadrado 13
7.5. Seguimiento de robots 14
7.6. Detección de colores de luz LED 15
7.7. Algoritmo genético 17
7.7.1. Representación de los individuos 18
7.7.2. Función de fitness 18
7.7.3. Método de selección 19
7.7.4. Cruza 19
7.7.5. Mutación 20
7.7.6. Método de reemplazo de la población 20

8. Evaluación **20**
8.1. Especificación de casos de prueba 20
8.2. Resultados experimentales 21
8.2.1. Esquinas del cuadrado 21
8.2.2. Exactitud de máscaras 22

9. Conclusiones **27**

10. Trabajo Futuro **27**

1. Justificación del trabajo

Las partículas autopropulsadas son agentes autónomos que convierten cierta cantidad de energía interna en movimiento. Es de interés estudiar los tipos de interacción que presentan estos tipos de agentes ya que, a partir de estos, pueden producir comportamientos emergentes que modifican la dinámica del sistema [1].
La propuesta es la de caracterizar un sistema de robots autopropulsados conocidos comercialmente como kilobots. El kilobot es un robot que puede ser programado, permitiendo experimentar con comportamientos colectivos y enjambres a gran escala [2].
Cada robot tiene capacidades básicas para ser autónomo, como un control programable, locomoción y comunicación local. Están ensamblados mediante procesos automáticos y hechos para que un usuario pueda operar un conjunto colectivo de kilobots. Son utilizados para investigar inteligencia artificial colectiva así como también para explorar nuevas teorías que relacionan capacidades individuales mínimas con comportamientos conjuntos complejos. Consecuentemente, el objetivo es desarrollar nuevos algoritmos robustos y escalables para el análisis de la organización de conjuntos colectivos con individuos limitados [3].

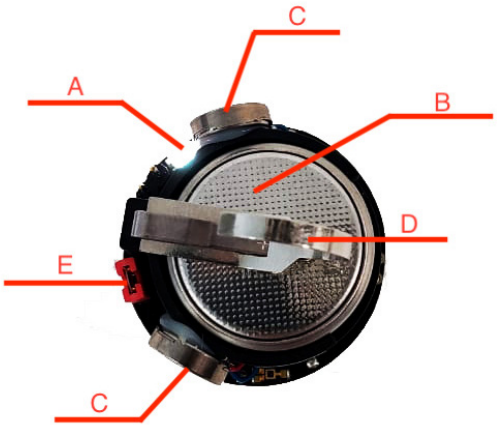


Figura 1: Componentes de un kilobot

En la figura 1 pueden observarse marcados los distintos componentes de un kilobot, los cuales corresponden a:

- A: Luz LED RGB. El color de la luz puede programarse y modificarse según el comportamiento.
- B: Batería recargable de 3.7V, la cual permite una autonomía de entre 3 y 10 horas de funcionamiento.
- C: Motores de vibración, los cuales permiten el movimiento del robot.
- D: Pestaña de carga, utilizada para realizar la carga de la batería.

- E: Jumper de encendido, utilizado para enceder el robot.

El comportamiento de los kilobots será analizado a partir de videos que se harán desde una posición cenital. Utilizando las trayectorias obtenidas y eventualmente algún otro observable, se podría caracterizar la evolución del sistema y definir métricas que cuantifiquen la respuesta del mismo. Para poder analizar el comportamiento de los robots a partir del video, es necesario desarrollar una aplicación que pueda detectar la posición de cada robot y su estado, el cual se identifica mediante el color de la luz LED que dispone cada robot.
De esta forma, los datos generados a partir del análisis del video pueden ser utilizados para caracterizar el comportamiento del sistema variando distintos parámetros de los robots, siendo uno de ellos su comportamiento. El comportamiento de cada robot puede ser programado con una API hecha en lenguaje C, mediante la cual puede definirse el comportamiento individual de cada robot y también el comportamiento que el robot adopta frente a otros agentes cercanos. La detección de los agentes cercanos se realiza por medio de un sistema de comunicación infrarroja.

2. Estado del Arte

Una investigación similar de seguimiento de agentes autopropulsados fue realizada en un proyecto final de alumnos del ITBA [4]. Este consiste de un sistema de procesamiento de imágenes que determina las posiciones y orientaciones espaciales de robots llamados HEXBUG nano, también conocidos como VDVs.
Para poder realizar el análisis, los robots fueron etiquetados con rectángulos de colores únicos para cada agente y luego se realizaron operaciones para suavizar las imágenes, erosionarlas y dilatarlas. Esto permite la localización de estos colores para luego hacer una detección de componentes y poder establecer la posición y orientación de cada robot.

3. Objetivos

El objetivo de este proyecto es poder determinar la posición y color de la luz LED de cada kilobot. Estos se mueven dentro de una tabla cuadrada de madera.
Para lograr este objetivo se tienen *timelapses*, que consisten de fotos tomadas cada una determinada cantidad de milisegundos. Este tiempo es mayor a la de un video, que son 60 *frames* por segundo. Esto se decidió teniendo en cuenta la velocidad a la que se mueven los robots que, al ser de aproximadamente 1 cm/s, se podían sacar las fotos de manera más espaciada y así lograr menos tiempo de procesamiento sin perder precisión. Además, las fotos tienen una mejor definición en comparación con los videos convencionales.

4. Planificación del proyecto

El proyecto fue dividido en diferentes tareas y, a su vez, cada tarea fue divida en subtareas. Las tareas principales consistieron en: análisis del problema, estudio de diferentes algoritmos de seguimiento y de tratamiento de imágenes, desarrollo, pruebas y escritura de informe. Para seguir el estado de las mismas se utilizó Trello, una plataforma de administración de proyectos. Allí las diferentes tareas fueron

asignadas a diferentes miembros del equipo, así como también se subieron archivos y documentación utilizada para cada una.

Tareas	**Duración (horas/hombre)**
Análisis del problema	**10**
Entendimiento de kilobots y su funcionamiento	3
Medición de tabla y kilobots	1
Definición de configuración de cámara	1
Elección de tecnología a utilizar	5
Estudio de algoritmos	**20**
Desarrollo	**110**
Configuración de entorno	4
Definición de parámetros de input	4
Definición de output	2
Desarrollo de código	100
Pruebas	**34**
Tomar fotos y videos	6
Desarrollo de código de testeo	20
Tomar métricas	8
Escritura de informe	**30**
Total	**204**

5. Requerimientos

5.1. Requerimientos funcionales

Los requerimientos funcionales determinados para el presente problema fueron:

- Identificar el cuadrado donde se mueven los robots
- Identificar la posición de los robots en un video
- Identificar el color del LED de los robots en un video

5.2. Requerimientos no funcionales

Los requerimientos no funcionales determinados para el presente problema fueron:

1. Precisión
2. Mantenibilidad
3. Facilidad de pruebas

Como principal atributo se eligió **Precisión** ya que es de vital importancia poder establecer en cada frame del video la posición correcta y precisa de cada robot. Si la posición que el programa informa contiene errores, entonces los resultados de los análisis posteriores serán también erróneos.
Como segundo atributo de calidad se eligió **Mantenibilidad** porque es importante que sea fácil poder agregarle funcionalidad al programa ya que pueden aparecer nuevos requisitos en un futuro.
Como tercer atributo de calidad se eligió **Facilidad de pruebas**, ya que debe ser posible realizar pruebas sobre el sistema para verificar que el mismo esté funcionando de forma correcta. Esto es importante pues los resultados que se obtengan por parte del uso de la aplicación serán utilizados para posteriores análisis.

6. Diseño

El programa fue realizado en Python ya que se consideró que es el mejor lenguaje para esta problemática gracias a su facilidad de uso. Además, cuenta con una variedad de librerías para el uso de algoritmos de detección de patrones en imágenes, como OpenCV, y es útil para el paradigma de programación que se utilizó, programación orientada a objetos.
En el diagrama de la figura 2 pueden observarse los pasos que se llevan a cabo a lo largo del proceso de seguimiento de kilobots en un video.

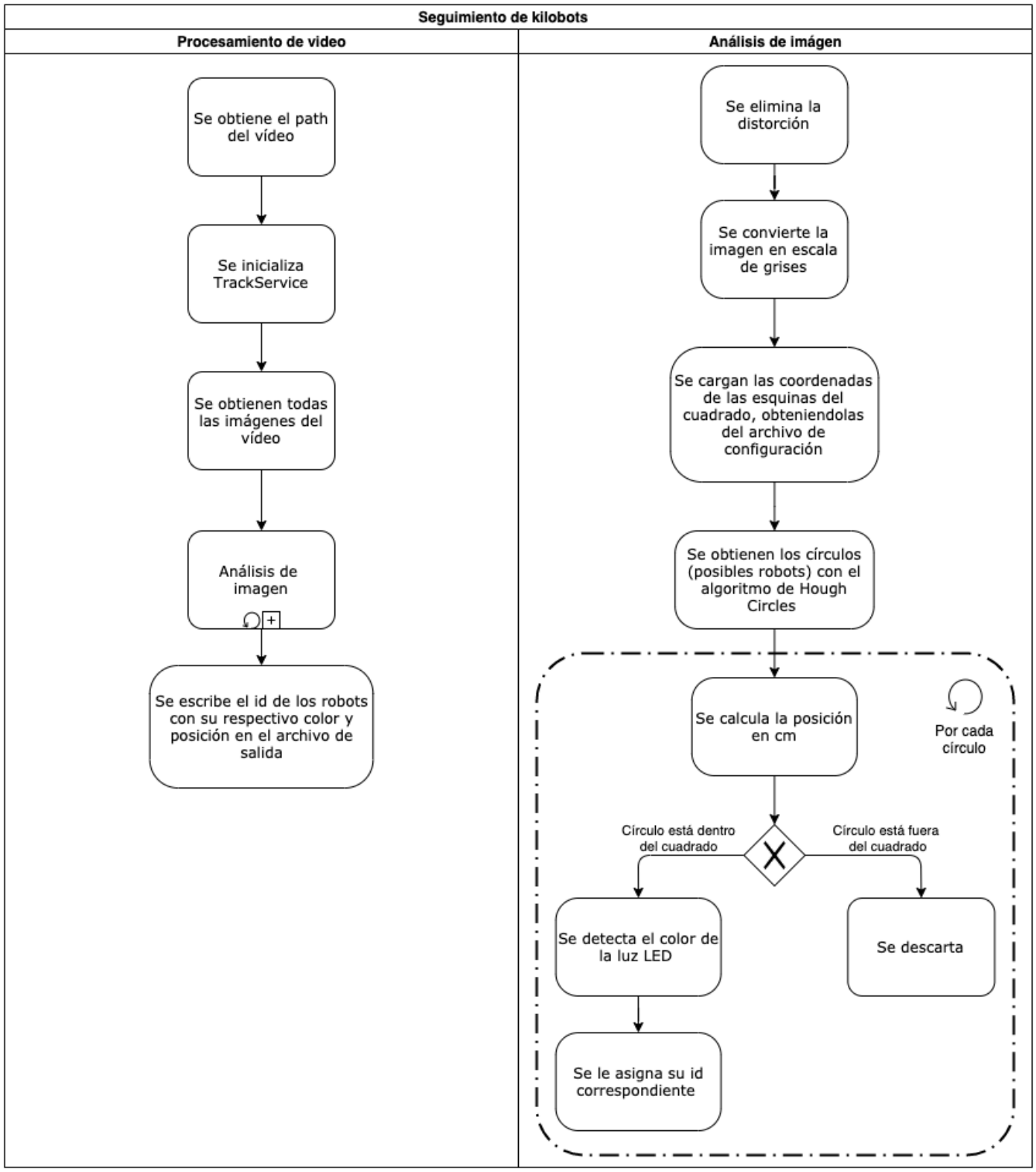


Figura 2: Diagrama de flujo del proceso de seguimiento de robots

6.1. Arquitectura

Como este es un programa que se correrá ocasionalmente, no fue diseñado para soportar muchas consultas simultáneamente, sino que analiza un video o imagen por vez. El resultado es escrito en un archivo por lo que no es necesario contar con una base de datos.
Se tienen diferentes módulos para poder organizar el código de manera eficiente y que sea mantenible.

6.1.1. Servicios

- ClientService: Este servicio se encarga de correr las distintas funcionalidades del programa. Está hecho para que se pueda correr tanto de forma programática como interactiva en consola.
- TrackService: Este servicio se encarga de analizar las fotos y videos. Además, es el que escribe los resultados de los análisis en los distintos archivos.
- BotService: Este servicio se encarga de encontrar a los kilobots en una imagen usando el algoritmo de Hough Circles.
- GridService: Este servicio se encarga de definir la grilla mediante un archivo de configuración y transformar los píxeles a centímetros.
- ColorService: Este servicio se encarga de detectar los colores para cada kilobot usando las máscaras que se encuentran especificadas en el archivo de *settings*.
- DirectoryService: Este servicio se encarga de la lectura y escritura de los archivos.
- CalibrationService: Este servicio se encarga de calibrar la imagen y eliminar la distorsión generada por el ojo de pez en el lente de la cámara.

6.1.2. Modelos

- Bot: Representa un kilobot en una foto. Tiene la posición del kilobot en centímetros y en píxeles dentro de la foto y el color de la luz LED.
- Image: Este objeto representa una imagen. Contiene métodos que permiten aplicar distintas operaciones sobre la imagen, como dibujar sobre ella o contar la cantidad de píxeles.
- AnalyzedImage: Este objeto hereda de Image y es el que usa los servicios para poder analizar la imagen y obtener los kilobots, su posición y el color de la luz.
- Settings: Esta clase se usa para guardar las configuraciones del programa que está corriendo.

7. Datos de implementación

7.1. Input

El programa recibe siempre un parámetro obligatorio. Este parámetro se especifica con -s o --settings seguido del nombre y directorio del archivo de configuraciones del programa. Además mediante el

parámetro -i o --interactive es posible correr el programa de manera interactiva. Para poder seleccionar los servicios que se quieren correr en el modo interactivo, se utilizó una librería llamada PyInquirer. Los servicios que se brindan a través del menú interactivo son:

- Calibrate: permite realizar la calibración de la cámara a utilizar.
- Define square: permite definir la posición del cuadrado a utilizar en el análisis de la imagen. Este servicio permite elegir la foto sobre la cual se desea detectar el cuadrado. En el caso de que el usuario no esté conforme con la detección que realizó el programa, es posible seleccionar otra foto para volver a realizar la detección.
- Track video: permite realizar un seguimiento de los kilobots en el video.
- Track photo: permite realizar un seguimiento de los kilobots en la imagen.
- Exit: permite salir del programa.

Además se cuenta con un archivo de configuración en formato YAML, en donde se pueden especificar los parámetros a utilizar. Estos son:

- Ubicación de archivos
  - photos_directory: especifica la carpeta que contiene las fotos a analizar.
  - video_directory :especifica la carpeta que contiene los videos a analizar.
  - results_directory: especifica la carpeta en donde se escribirán los resultados.
  - calibration_directory: especifica la carpeta en donde se encuentran las carpetas con las imágenes a calibrar.
  - config_directory: especifica el directorio en donde el programa va a guardar todos los archivos de configuración.
  - calibration_files_directory: especifica el nombre de la carpeta adentro de config_directory en donde están guardados los archivos de configuración de la calibración.
  - square_conf_name: especifica el archivo donde se escribe la definición del cuadrado y es el utilizado en los otros procesos.
- Dimensión de grilla
  - grid_width: es el ancho del cuadrado en cm. Se mide entre los tornillos.
  - grid_height: es la altura del cuadrado en cm. Se mide entre los tornillos.
- Máscaras
  - blue
    - blue_inf: es el límite inferior del color azul utilizado en la máscara, en formato HSV.
    - blue_sup: es el límite superior del color azul utilizado en la máscara, en formato HSV.

- green
  - green_inf: es el límite inferior del color verde utilizado en la máscara, en formato HSV.
  - green_sup: es el límite superior del color verde utilizado en la máscara, en formato HSV.
- red
  - red_inf: es el límite inferior del color rojo utilizado en la máscara, en formato HSV.
  - red_sup: es el límite superior del color rojo utilizado en la máscara, en formato HSV.
- white
  - white_inf: es el límite inferior del color blanco utilizado en la máscara, en formato HSV.
  - white_sup: es el límite superior del color blanco utilizado en la máscara, en formato HSV.

Parámetros a utilizar en Hough Circles. Hay un set de estos para la detección de tornillos y otro para la detección de bots

- bots_params: parametros que se utilizan para detectar a los robots.
  - param1: es el primer parámetro utilizado en el algoritmo de Hough Circles.
  - param2: es el segundo parámetro utilizado en el algoritmo de Hough Circles.
  - min_radius: es el radio mínimo de los círculos a detectar en el algoritmo de Hough Circles expresado en centímetros.
  - max_radius: es el radio máximo de los círculos a detectar en el algoritmo de Hough Circles expresado en centímetros.
  - dp: especifica la relación inversa de la resolución del acumulador utilizado en el algoritmo de Hough Circles.
- screws_params: parametros que se utilizan para detectar a los tornillos.
  - param1: es el primer parámetro utilizado en el algoritmo de Hough Circles.
  - param2: es el segundo parámetro utilizado en el algoritmo de Hough Circles.
  - min_radius: es el radio mínimo de los círculos a detectar en el algoritmo de Hough Circles expresado en pixeles.
  - max_radius: es el radio máximo de los círculos a detectar en el algoritmo de Hough Circles expresado en pixeles.
  - dp: especifica la relación inversa de la resolución del acumulador utilizado en el algoritmo de Hough Circles.

Parámetros para identificar qué se muestra en el video de salida. Sus valores pueden ser *True* o *False*

- show_square: Muestra el cuadrado que detecto para poder transformar los píxeles a centímetros.

- show_screws: Muestra en todos los *frames* los tornillos que se detectaron
- create_video: Si se quiere crear el video que contiene todas las fotos analizadas.
- square_detection_method: Qué algoritmo se quiere usar para la detección de cuadrados. Este puede ser 'linear-regresion' o 'border_screws'.
- lr_delta: La cantidad de píxeles que el algoritmo de regresión lineal va a considerar para construir una recta.
- commands: Lista de comandos.

7.2. Output

En el caso del seguimiento de kilobots, se genera un archivo de formato *xyz*. Por cada *frame*, se tiene un renglón con el número de bots encontrados y luego se escribe una línea por cada robot encontrado. Estas líneas consisten de:

ID X Y R G B

En donde ID es el identificador de cada robot y X e Y corresponden a la posición en centímetros, tomando como origen de coordenadas la esquina inferior izquierda del cuadrado detectado. Finalmente R, G, B hacen referencia al color de la luz LED que tiene el kilobot en ese momento expresado en el espacio de color RGB. Además de esto, opcionalmente, se puede generar un video en donde se muestran los elementos analizados de cada *frame*.

7.3. Servicio de calibración de cámara

Al tener un lente ojo de pez, se debe llevar a cabo el proceso de calibración para que la imagen no esté distorsionada y pueda ser analizada. Para hacerlo se deben guardar las fotos de calibración descriptas en GoPro calibration dentro de una carpeta que se encuentra dentro de *calibration_directory*.
Como resultado se generan dos archivos, *dist.conf* y *matrix.conf* que son utilizados por los demás servicios. En caso de tener la cámara ya calibrada, sólo con especificar el *path* a la carpeta donde se encuentran estos archivos es suficiente.

7.4. Algoritmos utilizados

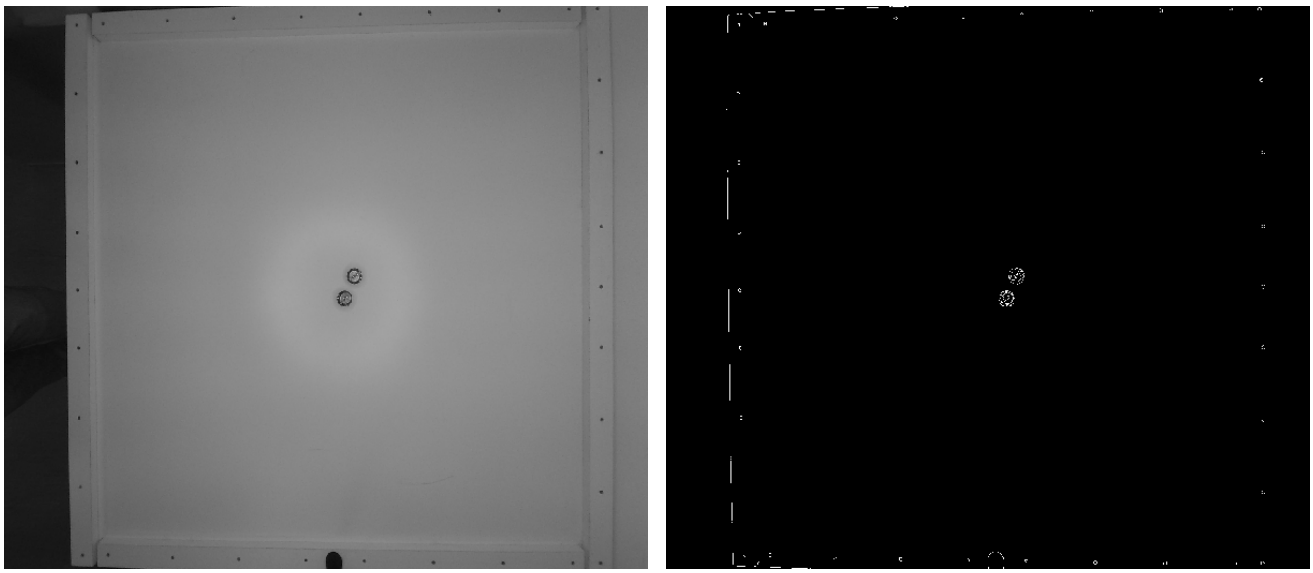
7.4.1. Detección de los robots y de los tornillos

Se utilizó la transformada de Hough para la detección de los robots y de los tornillos ya que estos se pueden ver como círculos en cada una de las imágenes. Cada círculo puede ser representado por

$$(x-a)^2 + (y-b)^2 = r^2. \tag{1}$$

Por lo tanto, cada círculo es definido por los parámetros (a, b, r), siendo (a, b) el centro de la circunferencia y r el radio.
Se construye una matriz M de tres dimensiones denominada acumulador. El algoritmo recibe como entrada una imagen en escala de grises, por lo que cada píxel se puede representar como un valor entre

0 y 255 ubicado en la posición (x_i, y_i) de la imagen.
Para poder aplicar el algoritmo es necesario, en primer lugar, detectar los bordes en la imagen. Para ello, es posible utilizar el método denominado Canny [5]. Este algoritmo realiza un proceso denominado Umbralización con Histéresis para lo cual recibe dos parámetros de umbral u_1 y u_2, siendo $u_1 < u_2$, y de esta forma detecta los bordes en la imagen.



(a) Foto antes de aplicar el filtro de Canny (b) Foto resultado de aplicar el filtro de Canny

Figura 3: Fotos ilustrando proceso de detección de robots y tornillos

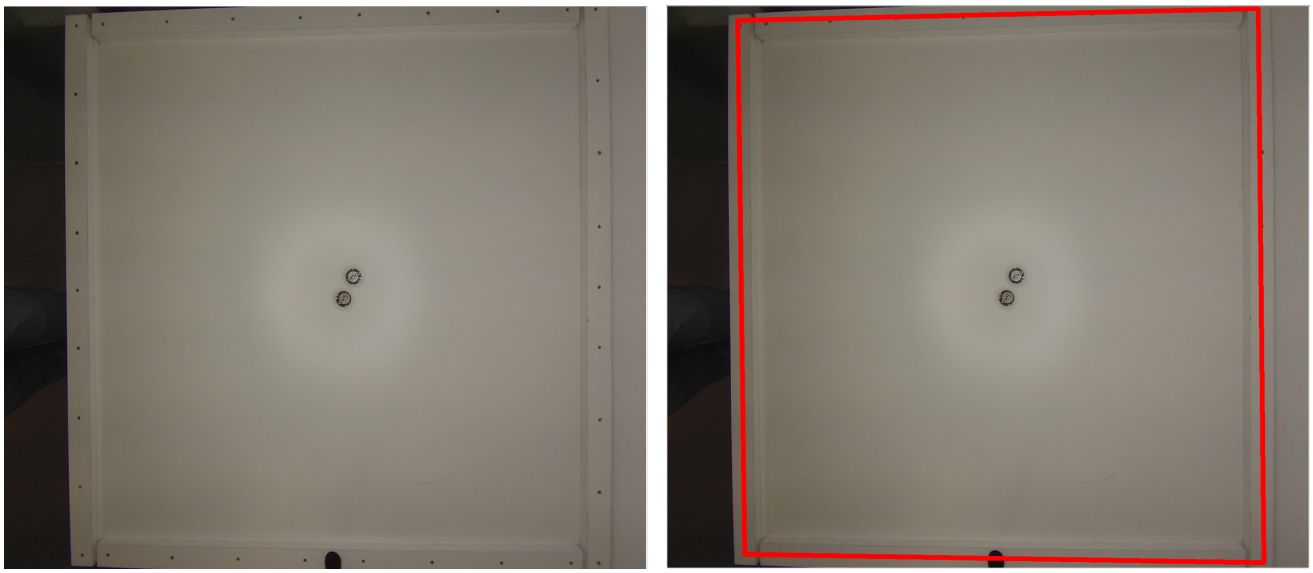
Por cada punto (x_i, y_i) correspondiente a un borde, se varían los parámetros a y b dentro de ciertos rangos. Se obtiene el valor de r utilizando 1. Se actualiza el valor de M según

$$M[a, b, r] = M[a, b, r] + 1. \tag{2}$$

Una vez realizado el procedimiento anterior, para todos los puntos se eligen los índices $[a, b, r]$ para los cuales $M[a, b, r]$ es máximo. Dichos índices corresponderán a los círculos hallados por el algoritmo de centro (a, b) y radio r.
El algoritmo descripto anteriormente supone un costo computacional elevado al requerir construir un acumulador de 3 dimensiones. Es por esto que OpenCV utiliza una optimización denominada 21HT descripta en [6]. Dicha optimización permite utilizar un acumulador de dos dimensiones y reducir el costo computacional.
La implementación de Hough Circles de OpenCV requiere especificar dos parámetros: param1 y param2. El primer parámetro, param1, corresponde al valor más grande del umbral (u_2) a aplicar en el algoritmo de Canny. El otro valor del umbral a aplicar será dos veces menor a este valor especificado. El segundo parámetro, param2, corresponde al umbral del acumulador para los centros de los círculos en la fase de detección del algoritmo. Cuanto más pequeño sea este valor, mayor cantidad de círculos falsos serán detectados.

7.4.2. Detección de cuadrado

Este servicio consiste en la detección de los bordes de la tabla de madera en donde se mueven los robots, a partir de los tornillos en la misma. La figura 4a ilustra cómo se ve la imagen antes del proceso y la figura 4b luego de este.



(a) Imagen antes de la detección del cuadrado (b) Imagen después de la detección del cuadrado

Figura 4: Fotos ilustrando proceso de detección de cuadrado

Para detectar el cuadrado se puede elegir entre dos algoritmos diferentes: el primero consiste en detectar los tornillos de las esquinas, llamados *tornillos de borde*, y el segundo algoritmo utiliza regresión lineal. En el primer algoritmo, se buscan todos los posibles tornillos y se eligen aquellos que están en los extremos. Es decir, se busca el tornillo más a la izquierda y arriba, el más a la izquierda y abajo, el más a la derecha y arriba y el más a la derecha y abajo. Luego se trazan las líneas correspondientes uniendo los puntos para formar el cuadrado.
Para el algoritmo de regresión lineal, los pasos son los siguientes:

1. Se agrupan los tornillos detectados en grupos de acuerdo a su proximidad. Dichos grupos pueden ser de dos tipos; G_x en donde la distancia se mide en la coordenada x o G_y en donde la distancia se mide en la coordenada y. Así la pertenencia a G_x se define como

$$p \in G_x \Leftrightarrow \exists\ p'_x \in G_x / |p_x - p'_x| \leq lr_delta, \tag{3}$$

y a G_y como

$$p \in G_y \Leftrightarrow \exists\ p'_y \in G_y / |p_y - p'_y| \leq lr_delta, \tag{4}$$

donde *lr_delta* es un parámetro en el archivo de configuración que se utiliza para determinar cuántos píxeles máximos de diferencia puede haber entre cada punto para que pertenezcan al mismo grupo.

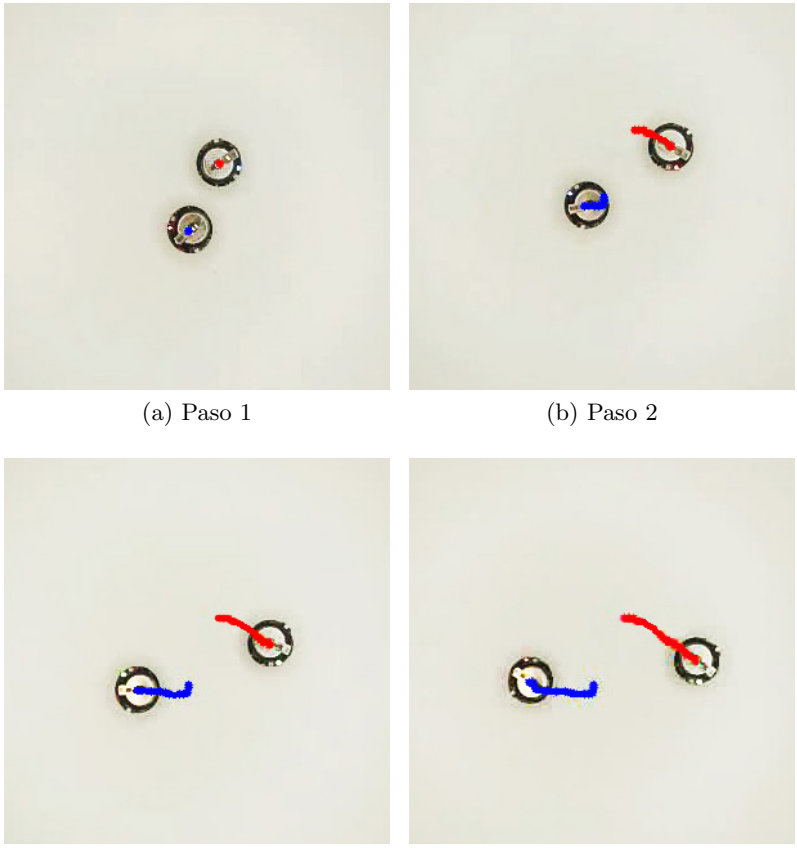
2. Se eligen los dos grupos G_x y G_y con mayor cantidad de puntos.

3. Con cada uno de los grupos seleccionados se realiza una regresión lineal por cuadrados mínimos [8] para determinar la recta que mejor aproxima a cada conjunto. Las rectas horizontales son las calculadas con los puntos en cada G_y y las verticales, en G_x.

4. Se obtiene la intersección entre las rectas para determinar las esquinas del cuadrado.

7.5. Seguimiento de robots

Para poder detectar el movimiento de cada robot se utilizó un algoritmo denominado *centroid tracker*. En el primer *frame*, para cada robot, el algoritmo le asigna un id único. Para asignar los ids en los siguientes *frames*, el algoritmo calcula para cada robot del *frame* anterior cuál es el robot más cercano en el siguiente *frame*. Para esto se computa la distancia euclídea entre los centros de los robots, calculados mediante la transformada de Hough de OpenCV. Cuando se remueven robots durante el video el algoritmo seguirá analizando los restantes pero, en el caso de que se agreguen, el algoritmo seguirá trabajando con los iniciales sin incluir a los nuevos.
Se puede observar en la figura 5 la trayectoria de los robots, donde cada punto representa la posición detectada por el algoritmo de seguimiento de robots. Los colores de las trayectorias dibujadas corresponden a un id determinado. De esta forma, se logra mostrar cómo se distinguen los diferentes robots.



(a) Paso 1 (b) Paso 2

(c) Paso 3 (d) Paso 4

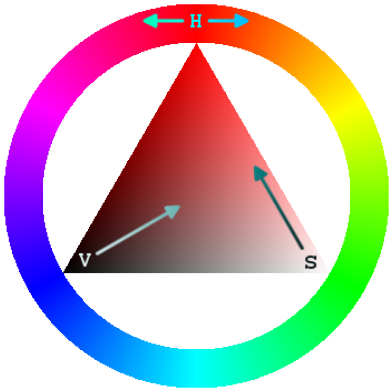
Figura 5: Seguimiento de trayectoria de robots

7.6. Detección de colores de luz LED

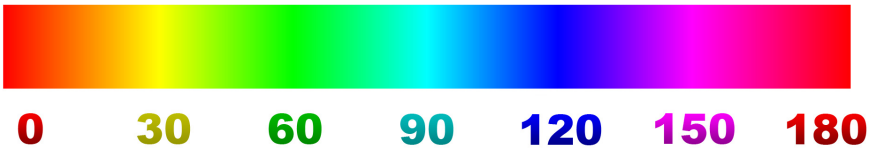
Para detectar los colores de las luces LED, se convirtió la imagen del espacio RGB al de HSV [9], donde H representa la matiz del color, S la saturación y V la altura en el eje blanco-negro (ver figura 6a). Como se utiliza el espacio HSV de OpenCV, la H puede tomar valores entre 0 y 180, la S puede tomar valores entre 0 y 255 y la V puede tomar valores entre 0 y 255 [10].
Para cada color se utilizó una máscara que define el rango del color dentro de HSV. Estas se encuentran en el archivo de configuración.
Como el valor H del color rojo se encuentra centrado en el 0, las máscaras de este color pueden tomar valores entre [160-180] y [0-20], como se observa en la figura 6b. Para simplificar la selección de la máscara, se decidió que los valores entre 160 y 180 se convirtieran en números negativos. A cada H se le aplica la función dada por

$$F(H) = \begin{cases} H + 180 & \text{si } H < 0 \\ H & \text{si } H \geq 0 \end{cases}, \tag{5}$$

la cual define de qué manera las máscaras negativas se convierten al rango de H para el rojo correspondiente.



(a) Cono HSV

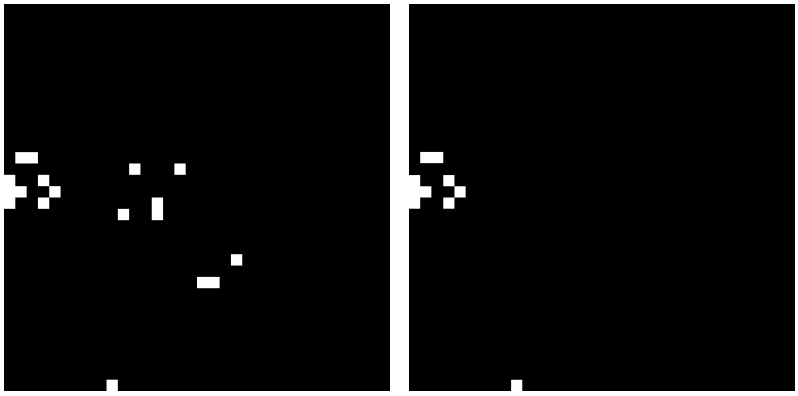


(b) Rango de valores de H en espectro HSV

Figura 6: Diagramas representando espectro HSV

Las máscaras son aplicadas a una foto que sólo contiene el robot detectado mediante el algoritmo de Hough Circles. La imagen que se observa en la figura 7 es un ejemplo del tipo de fotos que se utilizan para detectar los colores. Para poder contar los píxeles que están dentro del umbral de la máscara se aplicó la función de OpenCV *bitwise_or*, la cual transforma todos los píxeles que están dentro de los límites de la máscara en blancos y los demás en negros. A continuación se cuentan los píxeles blancos utilizando la función *countNonZero*.
Como puede observarse en 8a el robot presenta tonalidades de rojos que son detectados por la máscara pero que no son parte de la luz LED. Como solución a este problema, se tapa el centro de la imagen con un círculo negro, cuyo diámetro se encontró mediante prueba y error, ya que la luz LED siempre va a estar cerca del borde. De esta manera se puede evitar tomar en cuenta colores parecidos a la LED por equivocación, como puede apreciarse en la figura 8b. Esta técnica también se usa cuando se detecta si el color del LED es blanco. Esto se debe a que el reflejo de la luz hace que los píxeles en el robot se vean blancos.

Figura 7: Imagen que se usa para aplicar la máscara



(a) Sin el círculo (b) Con el círculo

Figura 8: Resultado de aplicarle la máscara a la foto de la figura 7

7.7. Algoritmo genético

Para la elección de las máscaras a utilizar en la detección de los colores de los LEDs de los kilobots, se utilizó un algoritmo genético. La decisión de utilizar este tipo de algoritmos tuvo que ver con el espacio de búsqueda de la solución.
Los algoritmos genéticos son algoritmos de búsqueda u optimización basados en la mecánica de la selección natural (supervivencia del más apto), creados por John Holland en 1975. Están basados en la Hipótesis de los Bloques Constructivos, la cual consiste en que en lugar de probar todas las combinaciones posibles de cromosomas, se van construyendo cada vez mejores cromosomas usando las mejores soluciones parciales obtenidas [11].
Los pasos del algoritmo son los siguientes:

- Se genera una población de individuos (también llamados cromosomas) que son las posibles

soluciones.

- Se seleccionan individuos para reproducción según su aptitud o *fitness*.
- Se genera la descendencia (Cruce o *Crossover*).
- Se mutan al azar los nuevos individuos.
- Se reemplazan los individuos de la generación anterior para obtener una nueva generación.

En el algoritmo 1 se puede observar una descripción en pseudocódigo del algoritmo implementado.

Algoritmo 1 Algoritmo genético

```
población ⇐ generar_población_aleatoria()
generación_actual ⇐ 0
max_fitness ⇐ 0
while generación_actual ≠ max_generaciones or max_fitness ≠ fitness_deseado do
  nueva_población ⇐ ∅
  for i = 0 to len(población) do
    progenitor_uno ⇐ roulette(población)
    progenitor_dos ⇐ roulette(población)
    nuevo_individuo ⇐ cruza(progenitor_uno, progenitor_dos)
    nuevo_individuo ⇐ mutación(nuevo_individuo)
    nueva_población ∪ nuevo_individuo
  end for
  generación_actual ⇐ generación_actual + 1
  max_fitness ⇐ max(población)
end while
```

7.7.1. Representación de los individuos

Para la aplicación del algoritmo al problema descrito anteriormente, se decidió representar a cada individuo como una máscara superior y una inferior.

7.7.2. Función de fitness

En el caso de la función de aptitud o *fitness*, la misma corresponde a la exactitud o *accuracy* del color que se está evaluando. La *accuracy* se calcula como

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{6}$$

en donde:

- TP = corresponde a los verdaderos positivos
- TN = corresponde a los verdaderos negativos

- FP = corresponde a los falsos positivos
- FN = corresponde a los falsos negativos

Para obtener la métrica anterior y poder evaluar la función de fitness, se corre el algoritmo de detección de colores en los videos de prueba utilizando la máscara del color que se quiere evaluar. Previamente fue necesario observar los videos y escribir para cada *frame* el color del LED que se observaba para cada robot. Esto permite posteriormente poder comparar el color detectado por el algoritmo contra el color verdadero observado previamente y poder determinar si el caso detectado corresponde a un TP, TN, FP o FN.
De lo anterior se desprende que los individuos más aptos serán aquellos compuestos por las máscaras que, al correr el algoritmo de detección de colores, ocasionen que la métrica de *accuracy* para ese color sea la más alta.

7.7.3. Método de selección

Para la selección de los individuos a reproducir, se eligió utilizar *Roulette*, un método de selección estocástico. Dicho método consiste en los siguientes pasos:

1. Se calculan las aptitudes relativas para cada individuo como

$$p_i = \frac{f_i}{\sum_{j=1}^{N} p_j},\ 1 \leq i \leq N \tag{7}$$

   siendo f_i el resultado de evaluar la función de fitness para el i-ésimo individuo de la población y N el numero de individuos en la población.

2. Se calculan las aptitudes dado

$$q_i = \sum_{j=1}^{i} p_j.\ 1 \leq i \leq N \tag{8}$$

3. Se generan dos números aleatorios y se seleccionan los dos individuos x_i que cumplen

$$q_{i-1} < r_j < q_i, \tag{9}$$

   siendo r_j el j-ésimo número aleatorio generado.
   Estos dos individuos seleccionados serán los que se utilizarán para realizar la cruza y la generación del individuo hijo.

7.7.4. Cruza

Una vez que se seleccionaron los dos progenitores, se realiza la cruza. Se decidió que exista una probabilidad de 0.5 tanto como para que se intercambien las máscaras superiores como de que se intercambien las máscaras inferiores de los progenitores. Es decir, de existir cruza, la máscara superior del primer progenitor se intercambia con la máscara superior del segundo progenitor. Lo mismo ocurre en el caso de las máscaras inferiores.

7.7.5. Mutación

Una vez generada la descendencia, se procede a mutar a los hijos que se generaron. El objetivo es evitar una convergencia prematura la cual impediría la exploración de un espacio lo suficientemente grande, produciendo así el estancamiento en un máximo local que no es óptimo.
Se decidió que exista un probabilidad de 0.6 de que no haya mutación. A su vez, existe una probabilidad de 0.2 de que solamente mute la máscara superior y una probabilidad de 0.2 de que solamente mute la máscara inferior. Estos valores fueron elegidos mediante un método de prueba y error, donde se buscó obtener una exactitud mayor al 97 %.

7.7.6. Método de reemplazo de la población

Se decidió generar N hijos a partir de la población anterior y que la población siguiente esté compuesta únicamente por esta descendencia generada.

8. Evaluación

8.1. Especificación de casos de prueba

Se realizaron pruebas para evaluar la efectividad de los algoritmos utilizados. Se analizó la exactitud al localizar las esquinas del cuadrado y para detectar los colores de los LEDs. Una aclaración a hacer es que todos los videos analizados cuentan con dos robots ya que no fue posible grabar más debido a la situación sanitaria actual. La efectividad para calcular la cantidad de los robots en todos los videos fue de 100 %, por lo que esta variable no fue necesario evaluarla.
Las pruebas llevadas a cabo se hicieron sobre dos grupos de videos. El primer grupo fueron videos que se grabaron utilizando la función de *timelapse*, obteniendo una resolución de las imágenes de 2592x1944 píxeles. Además, se utilizó un anillo de luz LED sobre la mesa para mejorar la iluminación. El segundo grupo fueron *frames* sacados de videos grabados sin el anillo de luz LED, utilizando la herramienta FFmpeg. Debido a que se usó la función de video de la cámara estas imágenes tienen una resolución de 1280x960.
En el caso de la evaluación del algoritmo de detección de cuadrado, se variaron los parámetros param1, entre 3 y 12, y param2, entre 40 y 100, utilizados en Hough Circles. Se calculó la distancia euclídea en píxeles entre el tornillo que se encuentra en cada esquina y la esquina correspondiente calculada por el algoritmo. Luego, se sumaron las distancias de cada una de las esquinas y se computó el promedio para cinco imágenes de videos diferentes. A este promedio se lo llamará Δd.
Para poder evaluar la precisión de las máscaras, se hizo una lista por cada video en la que se anotaron todos los colores de las luces LED agrupados por *frame*. Después, se comparó el resultado de correr el programa con las máscaras por cada uno de los videos. De esta manera se pudo sacar la *accuracy* y el porcentaje de robots acertados, definido por

$$porcentaje = \frac{kilobots_{acertados}}{kilobots_{totales}}, \tag{10}$$

en donde $kilobots_{acertados}$ corresponde a la cantidad de aciertos de color en todos los *frames* y $kilobots_{totales}$ corresponde a la cantidad de *frames* multiplicado por la cantidad de robots.

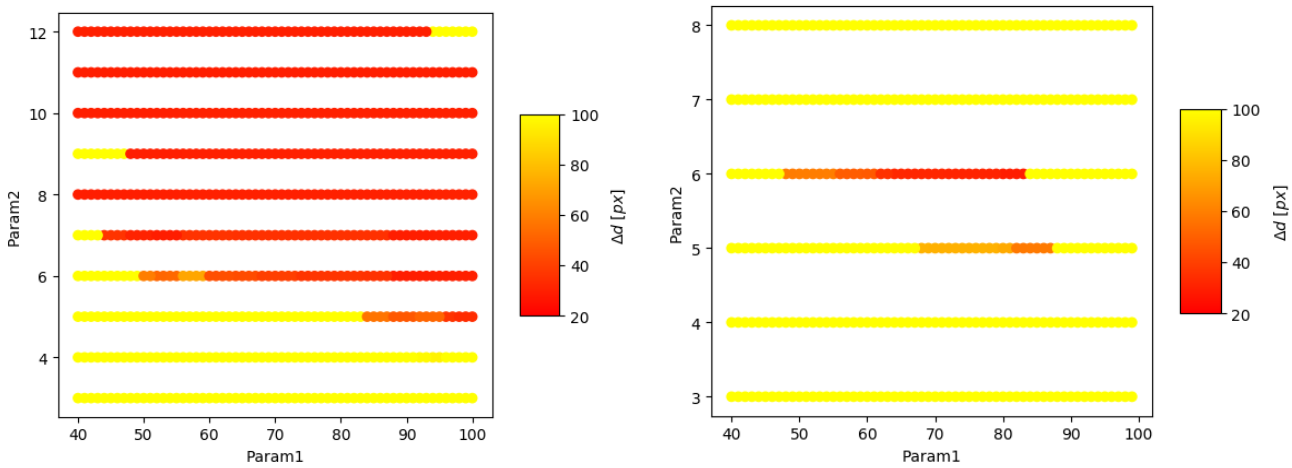
8.2. Resultados experimentales

8.2.1. Esquinas del cuadrado

Para cada prueba realizada, se graficaron mapas de calor en donde el color representa el valor de Δd. Cuanto más rojo, el valor de Δd es menor y al acercarse al amarillo, va incrementando. El rango de Δd analizado fue entre 20 y 100 píxeles ya que no hubo ninguna prueba con un Δd menor a la cota inferior y no fue de interés diferenciar valores superiores a la cota superior porque se considera que el error es demasiado grande para que el resultado sea viable. Se variaron los parámetros param1 en el eje x y param2 en el eje y.
Hubo pruebas en las que, si el param2 era muy elevado, el algoritmo no lograba determinar el cuadrado y se cortaba la ejecución. En ese caso, el rango de valores de param2 graficado fue modificado de manera acorde.

Algoritmo de regresión lineal

Al utilizar imágenes de resolución 2592x1944, puede observarse en 9a que, para valores de param2 menores a 4, independientemente del valor de param1, el valor de Δd siempre supera el umbral de 100 píxeles. Para valores de param2 entre 5 y 7, cuanto mayor es el valor de param1, se obtienen mejores resultados. Además, para valores de param2 entre 8 y 11, sin importar el valor de param1, los valores de Δd son similares y cercanos a 20 píxeles.
En el caso de imágenes de menor resolución, se puede observar en la figura 9b que se obtuvieron peores resultados, donde para param2 diferente a 5 o 6, el Δd es mayor a 100 píxeles. En el caso de param2 igual a 5 y param1 entre 70 y 80 el Δd es de alrededor de 70 píxeles, mientras que para un param2 de 6 y param1 entre 60 y 80 se obtuvieron resultados más cercanos a 20.



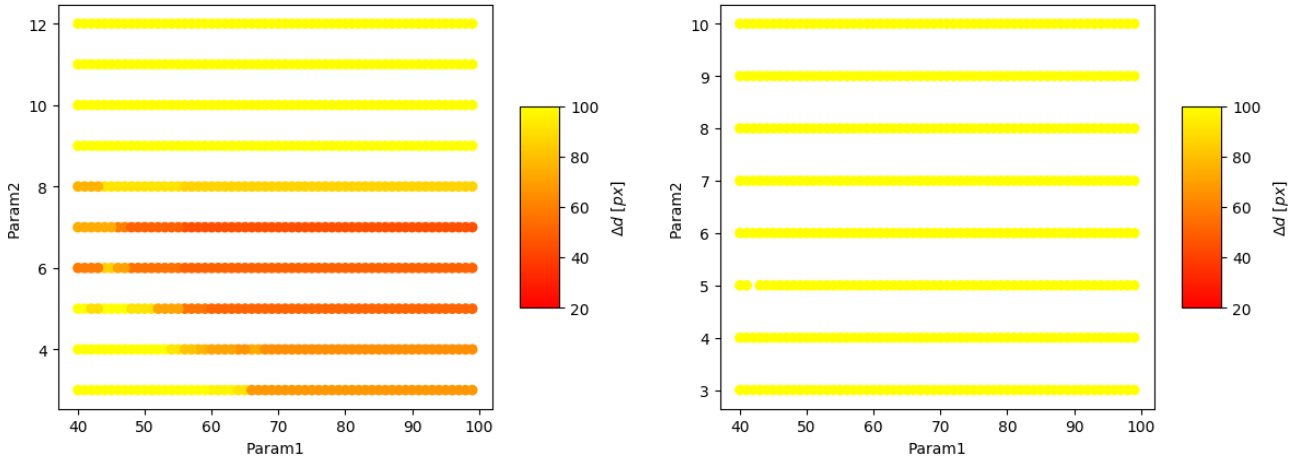
(a) Imágenes de resolución 2592x1944 con anillo de luz

(b) Imágenes de resolución 1280x960 sin anillo de luz

Figura 9: Mapas de calor representando Δd en función de param1 y param2. Se aplicó el algoritmo de regresión lineal

Algoritmo de tornillos de borde

Para el algoritmo de tornillos de borde se obtuvieron peores resultados comparados con el algoritmo de regresión lineal. Independientemente de la resolución de la imagen, no se observa un rojo tan intenso como en el algoritmo anterior, indicando que el valor de Δd es superior.
En el caso de imágenes de resolución 2592x1944, se aprecia en la figura 10a que, para valores de param2 mayores a 8, Δd siempre supera los 100 píxeles y para valores menores a 6, el mínimo valor al que llega es de alrededor de 60 píxeles. Algo a considerar es que para el algoritmo de regresión lineal, con valores de param2 menores o iguales a 4, Δd superaba los 100 píxeles, mientras que en este caso, para valores de param1 mayores a 70 se evidencia un Δd de alrededor de 70 píxeles.
Para imágenes de resolución 1280x960, se puede observar en la figura 10b que, indistintamente del valor de param1 y param2, Δd es mayor a 100 píxeles.



(a) Imágenes de resolución 2592x1944 con anillo de luz

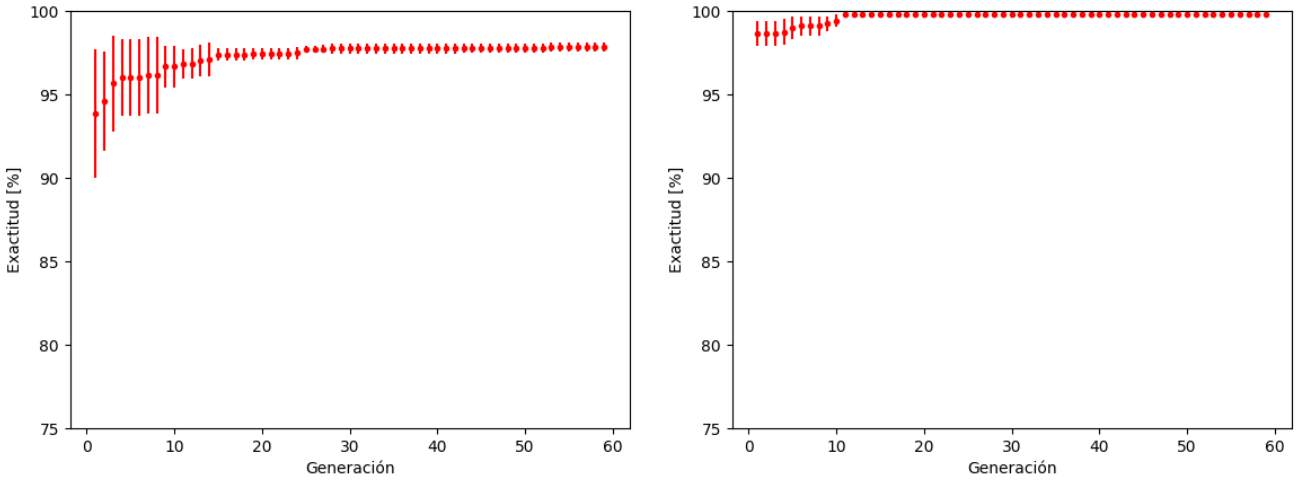
(b) Imágenes de resolución 1280x960 sin anillo de luz

Figura 10: Mapas de calor representando Δd en función de param1 y param2. Se aplicó el algoritmo de tornillos de borde

8.2.2. Exactitud de máscaras

Para determinar la máscara a utilizar para la detección de cada color del LED de los kilobots, se corrió un algoritmo genético para cada color. En las figuras 11, 12, 13, 14 se observa para cada generación del algoritmo genético, el porcentaje de exactitud (o *accuracy*) obtenido utilizando la máscara que corresponde al mejor individuo de la generación. Se realizaron 5 corridas del algoritmo genético por 60 generaciones y se promediaron los valores de *accuracy* del mejor individuo encontrado en esa generación.
Puede observarse en la figura 11a que las poblaciones del color rojo no superan el valor de 98 % de *accuracy* en las 60 generaciones, presentando un menor valor de este comparado con el resto de los colores. A su vez, los colores rojos presentan una mayor desviación estándar del *accuracy* de los mejores

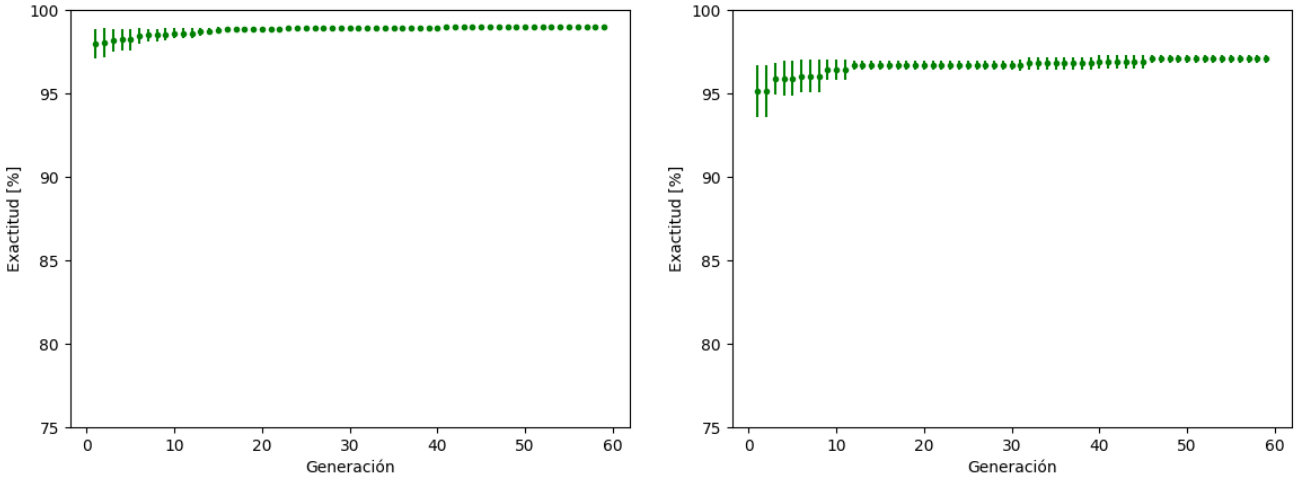
individuos.
Por otra parte, como se puede observar en la figura 13a, el *accuracy* de los colores azules tiende a estabilizarse en un período corto de generaciones comparado con el resto de los colores en donde la convergencia de la población lleva un mayor número de generaciones.
Para los videos con menor resolución se puede observar en la figura 14b que no llega a valores altos de exactitud estabilizándose cerca de las 35 generaciones alrededor de 81 %.
En las figuras 11 y 13 se puede observar que, para las imágenes de menor resolución, se lograron obtener valores más altos de exactitud, mientras que en las figuras 12 y 14 se obtienen valores menores.



(a) Imágenes de resolución 2592x1944 con anillo de luz

(b) Imágenes de resolución 1280x960 sin anillo de luz

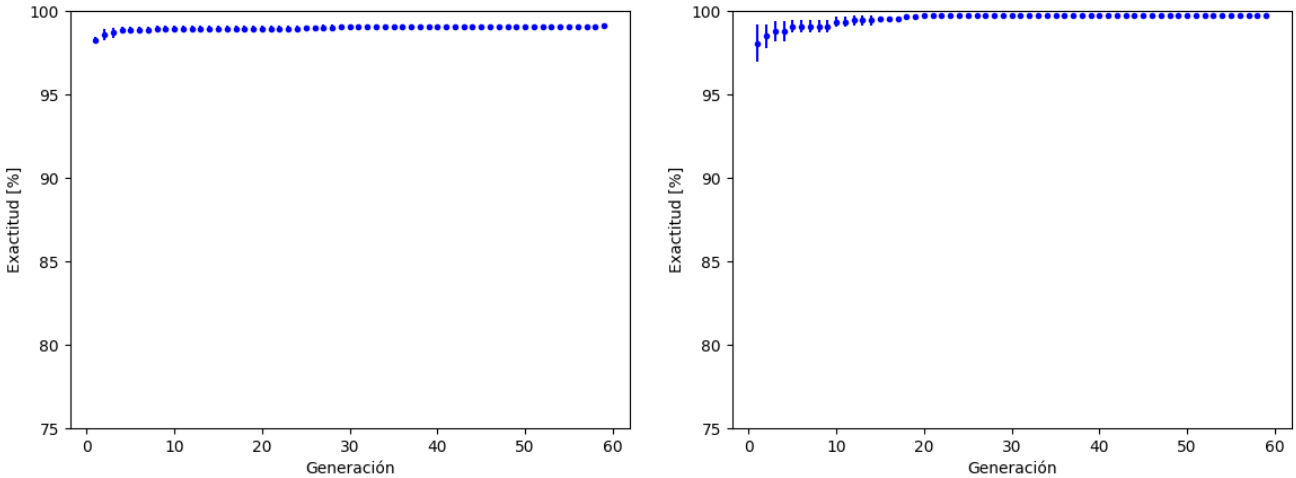
Figura 11: Porcentaje de exactitud (*accuracy*) utilizando la máscara de color rojo del mejor individuo encontrado en la generación



(a) Imágenes de resolución 2592x1944 con anillo de luz

(b) Imágenes de resolución 1280x960 sin anillo de luz

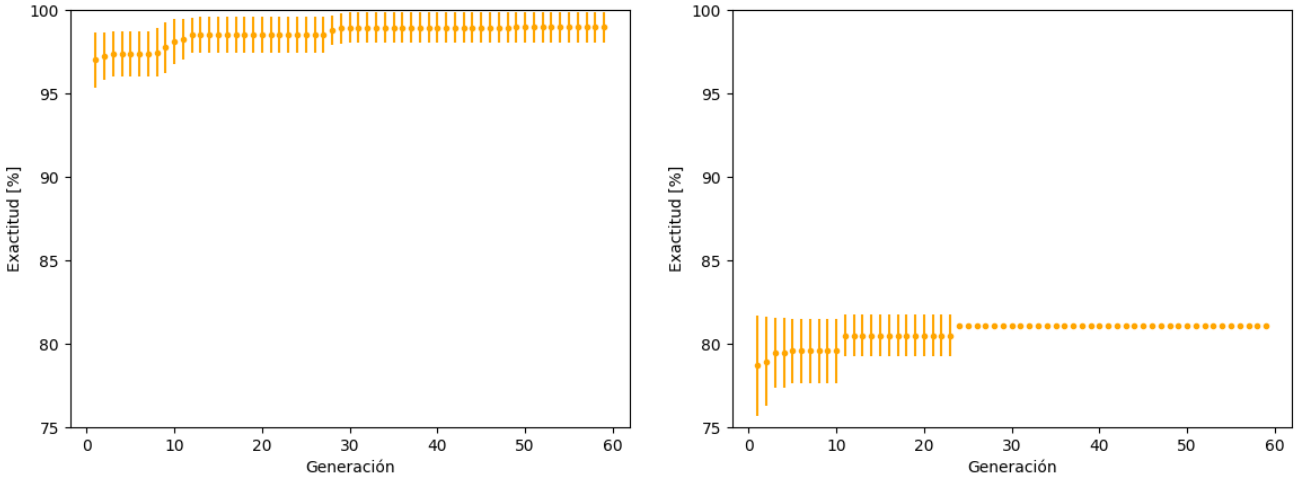
Figura 12: Porcentaje de exactitud (*accuracy*) utilizando la máscara de color verde del mejor individuo encontrado en la generación



(a) Imágenes de resolución 2592x1944 con anillo de luz

(b) Imágenes de resolución 1280x960 sin anillo de luz

Figura 13: Porcentaje de exactitud (*accuracy*) utilizando la máscara de color azul del mejor individuo encontrado en la generación

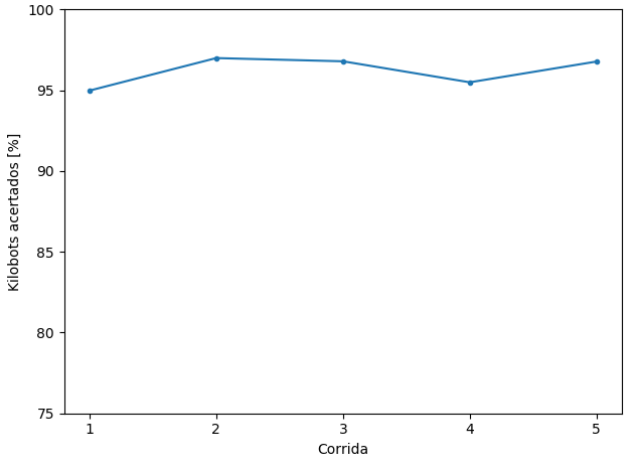


(a) Imágenes de resolución 2592x1944 con anillo de luz

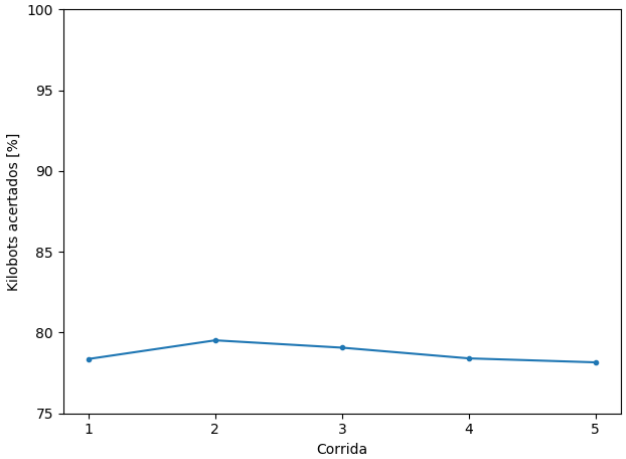
(b) Imágenes de resolución 1280x960 sin anillo de luz

Figura 14: Porcentaje de exactitud (*accuracy*) utilizando la máscara de color blanco del mejor individuo encontrado en la generación

Puede observarse en la figura 15 que las máscaras que dieron la mayor cantidad de aciertos, para las imágenes de mejor resolución y con anillo de luz, fueron las de las corrida 2. Como se observa en la tabla 1, estas corresponden a [-18,65,98]-[8,210,242] para el rojo, [43,63,58]-[71,208,234] para el verde, [108,31,47]-[129,204,216] para el azul y [0,1,240]-[177,9,254] para el blanco. Por el otro lado, para las imágenes de menor resolución, las máscaras correspondientes a la corrida con mayor exactitud son [-1,67,44]-[13,216,250] para el rojo, [37,49,14]-[62,220,218] para el verde, [101,6,36]-[132,216,229] para el azul y [1,1,235]-[172,9,253] para el blanco. Las máscaras fueron obtenidas de la corrida 2 en la tabla 2.



(a) Imágenes de resolución 2592x1944 con anillo de luz



(b) Imágenes de resolución 1280x960 sin anillo de luz

Figura 15: Porcentaje de aciertos obtenidos con las máscaras generadas por las distintas corridas

Corrida	Máscaras			
	Rojo	Verde	Azul	Blanco
1	[-2,88,91]-[11,240,203]	[46,39,54]-[57,243,254]	[110,17,44]-[127,239,214]	[0,2,244]-[179,9,254]
2	[-18,65,98]-[8,210,242]	[43,63,58]-[71,208,234]	[108,31,47]-[129,204,216]	[0,1,240]-[177,9,254]
3	[-19,60,94]-[8,219,253]	[46,21,60]-[71,243,216]	[100,40,47]-[130,203,233]	[5,2,239]-[179,9,254]
4	[-8,53,98]-[7,248,246]	[45,17,74]-[65,245,242]	[108,27,52]-[130,237,235]	[2,1,240]-[171,9,254]
5	[-1,76,96]-[14,245,232]	[45,27,64]-[67,235,251]	[108,28,47]-[128,250,251]	[0,0,239]-[177,9,254]

Tabla 1: Máscaras correspondientes a cada corrida con los videos de mejor resolución con anillo de luz

Corrida	Máscaras			
	Rojo	Verde	Azul	Blanco
1	[-7,45,58]-[12,206,201]	[36,55,44]-[56,233,235]	[102,6,6]-[136,240,206]	[0,1,238]-[171,9,252]
2	[-1,67,44]-[13,216,250]	[37,49,14]-[62,220,218]	[101,6,36]-[132,216,229]	[1,1,235]-[172,9,253]
3	[-14,84,27]-[12,221,232]	[37,51,0]-[55,250,204]	[101,6,38]-[131,253,223]	[0,0,237]-[172,9,252]
4	[-10,82,43]-[10,246,219]	[38,48,29]-[71,221,253]	[106,3,30]-[135,249,225]	[9,0,236]-[172,9,254]
5	[-8,83,23]-[12,251,208]	[38,57,12]-[59,239,216]	[100,1,33]-[131,247,202]	[1,0,240]-[171,9,253]

Tabla 2: Máscaras correspondientes a cada corrida con los videos de menor resolución sin anillo de luz

9. Conclusiones

En cuanto a los algoritmos de detección de cuadrado, se recomienda utilizar el algoritmo de regresión lineal en lugar del algoritmo de tornillos de borde ya que, para la mayoría de los casos, se obtuvo un menor Δd. Además, utilizar imágenes de mayor resolución también mostró mejores resultados.
Comparando ambos algoritmos se observa que, para el caso de detección de tornillos de borde, el mínimo Δd que se obtuvo fue superior al obtenido con regresión lineal. Sin embargo, para imágenes de resolución 2592x1944, utilizando un param2 entre 3 y 4 y param1 mayor a 70, se consiguió un Δd de alrededor de 60 píxeles, cuando para el otro algoritmo fue de más de 100 píxeles. Esto se debe a que, cuanto menor sea el param2, se encuentran más círculos que no son tornillos y, al buscar los círculos de los extremos, el máximo error que puede tener este algoritmo es la diferencia entre el contorno de la imagen y el tornillo de la esquina correspondiente. Por otro lado, en el algoritmo de regresión lineal, los falsos positivos desvían la recta calculada de tal forma que el error es mayor.
En el caso de utilizar imágenes de mayor resolución, se aconseja utilizar un param1 de 70 y un param2 de 8, mientras que para el caso de imágenes de menor resolución un param1 de 70 y un param2 de 6. Las diferencias de eficiencia de parámetros encontradas entre estas dos resoluciones se debe a que, al tener una mayor resolución, se tienen más píxeles en un área determinada. Consecuentemente se puede exigir una mayor cantidad de píxeles para determinar un círculo y así evitar los falsos positivos. Esta exigencia se expresa en un mayor param2.
Teniendo en cuenta los algoritmos de detección de colores, al subir la resolución de las imágenes y usar el anillo de luz, en general, se obtuvieron mejores resultados. Las únicas excepciones son para el color azul y el rojo. Sin embargo, la variable más importante a analizar, que tiene en cuenta todos los colores, es el porcentaje de aciertos obtenidos. En este caso, las imágenes de mayor resolución con el anillo de luz superaron ampliamente su valor, siendo alrededor de 96 %, en comparación con las otras imágenes analizadas, que obtuvieron un porcentaje de alrededor del 76 %. Esto es debido a que la mayor resolución permite tener una mayor cantidad de píxeles para detectar el color. Además, el anillo de luz LED realza los colores y ayuda a que la distinción de los mismos sea más notable.
En cuanto al caso del color rojo, para las imágenes de mejor resolución, la menor *accuracy* de las poblaciones iniciales y el mayor tiempo de convergencia se debe a que este color está presente en distintas partes de los robots que hacen que el algoritmo se los confunda con la luz LED. De este modo, resulta más dificultoso encontrar mejores máscaras al incrementarse el espacio de búsqueda.
Por otra parte, se observa que el color blanco es capaz de llegar a valores más altos de *accuracy* y esto puede deberse a que la luz blanca que alumbra desde la posición cenital a los robots hace que se realcen los colores blancos de las LEDs. Esto también explica por qué la exactitud del color blanco para las imágenes sin el anillo de luz fue considerablemente inferior a los demás, con un máximo de 81 %.

10. Trabajo Futuro

Debido a la pandemia global no se pudo recolectar material adicional para hacer distintas pruebas. Es por esto que todos los videos analizados cuentan con dos robots. Se deberían hacer más pruebas con diferentes cantidades de robots para determinar si la detección de robots funciona adecuadamente con grandes cantidades de ellos. Se tuvo un 100 % de efectividad para encontrarlos en todos los videos que se pudieron obtener.

No se pudo determinar la orientación de los kilobots con los videos obtenidos. Una de las soluciones que se planteó es poner etiquetas de colores en la parte delantera de estos para poder determinar cuál es el frente de los mismos y así poder calcular la orientación. Debido a la pandemia no se pudieron realizar estas pruebas y por ende no se pudo detectar la orientación de los mismos.

Referencias

[1] Ramaswamy, S. (2017). *Active matter*. Journal of Statistical Mechanics: Theory and Experiment, 2017(5), 54002.

[2] Kilobot – K-Team Corporation. K-team.com. (2020). Recuperado el 5 de julio de 2020, de https://www.k-team.com/mobile-robotics-products/kilobot.

[3] Rubenstein, M., Ahler, C., Hoff, N., Cabrera, A., & Nagpal, R. (2014). *Kilobot: A low cost robot with scalable operations designed for collective behaviors*. Robotics and Autonomous Systems, 62(7), 966-975.

[4] Nardini G., Sakuda M., Vázquez D. *Detección y seguimiento de partículas autopropulsadas*. ITBA

[5] Canny, J. (1986). A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(6), 679–698

[6] Yuen, H., Princen, J., Illingworth, J., & Kittler, J. (1990). *Comparative study of Hough Transform methods for circle finding*. Image and Vision Computing, 8(1), 71-77

[7] Kilobots. Self-Organizing Systems Research Group. (2020). Recuperado el 5 de julio de 2020 de https://ssr.seas.harvard.edu/kilobots.

[8] sklearn.linear_model.LinearRegression scikit-learn 0.23.1 documentation. Scikit-learn.org. (2020). Recuperado el 5 de julio de 2020, de https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html.

[9] Modelo HSV - EcuRed. Ecured.cu. (2020). Recuperado el 5 de julio de 2020, de https://www.ecured.cu/Modelo_HSV.

[10] OpenCV: Changing Colorspaces. Docs.opencv.org. (2020). Recuperado el 5 de julio de 2020, de https://docs.opencv.org/trunk/df/d9d/tutorial_py_colorspaces.html.

[11] Cátedra de Sistemas de Inteligencia Artificial. (2019). *Algoritmos Genéticos*. ITBA