# ALGORITMO DE NAVEGACIÓN PARA UN ROBOT MÓVIL BASADO EN EL RECONOCIMIENTO VISUAL DE PATRONES EN TIEMPO REAL Y PLANIFICACIÓN DE TRAYECTORIAS

SERGIO RODRÍGUEZ MÉNDEZ
PABLO CÉSAR RUBIANO ACOSTA

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍAS
INGENIERÍA ELECTRÓNICA
BOGOTÁ D.C
2015

# ALGORITMO DE NAVEGACIÓN PARA UN ROBOT MÓVIL BASADO EN EL RECONOCIMIENTO VISUAL DE PATRONES EN TIEMPO REAL Y PLANIFICACIÓN DE TRAYECTORIAS

SERGIO RODRÍGUEZ MÉNDEZ
PABLO CÉSAR RUBIANO ACOSTA

**Trabajo de grado para optar al título de:**
**Ingeniero Electrónico**

**DIRECTOR**
**Ivan Dario Ladino Vega**
**Ingeniero Electrónico**

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍAS
INGENIERÍA ELECTRÓNICA
BOGOTÁ D.C
2015

**Nota de aceptación**

_____

_____

_____

_____

_____

Firma presidente del jurado
Nombre:

_____

Firma jurado
Nombre:

_____

Firma jurado
Nombre:

Bogotá D.C, 18 de marzo de 2015

*Dedicado a nuestras familias,*
*amigos y a todos quienes*
*siempre estuvieron a lo largo*
*de este proceso de formación,*
*para brindarnos todo su apoyo,*
*ayuda y comprensión.*

# AGRADECIMENTOS

Este trabajo no habría sido posible sin la influencia directa o indirecta de muchas personas a las que agradecemos profundamente por estar presentes en las distintas etapas de su elaboración, así como en el resto de nuestras vidas.

Agradecemos al ingeniero Ivan Dario Ladino Vega por manifestar su interés en dirigir este trabajo de grado, por su confianza, colaboración y apoyo a lo largo de este trabajo y a nuestra formación como profesionales.

# CONTENIDO

# LISTA DE FIGURAS

pág.

# INTRODUCCIÓN

La navegación de robots móviles capaces de desplazarse de forma autónoma en un entorno desconocido o conocido parcialmente, es un área de amplio interés en robótica por el gran número de aplicaciones en donde resulta necesario determinar la posición del robot (entornos industriales, militares, domésticos y de seguridad). La navegación hace posible que un robot móvil pueda realizar un mapa de su entorno, a partir del cual pueda determinar su posición y planificar trayectorias para lograr desplazarse de un punto a un destino concreto mientras se evitan obstáculos.

Uno de los más importantes desafíos dentro de la navegación robótica es conseguir que un robot conozca su posición mientras se desplaza a través de su entorno. Afortunadamente, durante las dos últimas décadas se han realizado importantes avances en el desarrollo de estrategias dentro del campo de la navegación robótica, para determinar la localización, construir de mapas, e incluso una combinación de ambas. Como resultado se han desarrollado diferentes métodos probabilísticos muy robustos, por ejemplo técnicas basadas en el filtro de Kalman, filtro de partículas, redes bayesianas, entre otros. A pesar de esta amplia evolución de métodos, los robots son todavía incapaces de extraer información del entorno con la misma facilidad y precisión que lo hacemos los seres humanos.

La capacidad de navegar en ambientes desconocidos forma una parte importante de lo que significa un robot autónomo. En este sentido, uno de los principales problemas con los que se tiene que enfrentar un investigador o desarrollador de robots móviles es el problema de la localización y confección de mapas del entorno. El robot autónomo Curiosity creado por la NASA (del inglés National Aeronautics and Space Administration, o en español *Administración Nacional de la Aeronáutica y del Espacio*), se encuentra en un ambiente desconocido y sin tener información precisa sobre su ubicación. Sin embargo, el principal objetivo con el que fue creado este robot es la navegación.

Una de las áreas de aplicación en donde se utiliza la navegación robótica móvil en entornos controlados, es en el sector del entretenimiento. En donde se utilizan robots móviles como guías de museo o para realizar algún deporte como jugar en el campeonato de fútbol de la RoboCup, este proyecto internacional fue creado para promover, a través de competiciones integradas por robots autónomos, la investigación y educación sobre inteligencia artificial.

En esta competición se utilizan robots humanoides como el robot NAO de Aldebarán que cuenta con una cámara como sensor principal para determinar su posición en el campo, percibir la pelota, las porterías. Las técnicas habituales para la construcción de mapas del campo de fútbol emplean directamente los datos capturados de las cámaras sin ningún tratamiento, es decir, sin extraer ningún tipo de característica. Las técnicas empleadas y que funcionan de este modo son: Localización Markov, localización basada en filtro de partículas.

El objetivo de la RoboCup es que año tras año se mejoren los algoritmos y técnicas con las cuales se pueda conseguir desarrollar un equipo autónomo de robots de aspecto humanoide que sea capaz de superar a la selección campeona del mundo en 2050.

## RESUMEN

Este trabajo de grado describe el diseño, implementación y análisis de un algoritmo para el reconocimiento de patrones y la planificación de trayectorias que permitan a un robot móvil desplazarse desde un punto inicial a un punto final en un entorno. Para el reconocimiento de patrones, se hace uso del procesamiento digital de imágenes, a partir del que se encuentra la posición y orientación del robot en el entorno.

La planificación de trayectorias se realiza utilizando un método geométrico (método de tangentes) con el que se consigue una trayectoria que permita al robot llegar a un punto destino de forma óptima.

# OBJETIVOS

## OBJETIVO GENERAL

Desarrollar un algoritmo que permita a un robot móvil dentro de un entorno, obtener un mapa y luego utilizarlo para navegar a través de el, desde una posición inicial a una final.

## OBJETIVOS ESPECÍFICOS

- Análisis del estado del arte de la navegación, localización, y mapeo en robots móviles basados en métodos del entorno.
- Desarrollar, implementar y evaluar un algoritmo para la obtención de las coordenadas espaciales del reconocimiento de un patrón de punto dentro de un entorno.
- Desarrollar, implementar y evaluar un algoritmo para la planificación de trayectorias que permitan a un robot móvil desplazarse de un punto inicial a un punto final.

# ANTECEDENTES

El principal problema de los robots móviles consiste en contestar a la pregunta ¿Dónde estoy? desde el punto de vista del robot. Esto significa que el robot tiene que ser capaz de encontrar su posición relativa al entorno en el que está. Cuando se habla de posición se refiere a las coordenadas $(x, y)$ y una orientación del robot en un sistema de coordenadas global.

Existen una gran cantidad de maneras de realizar la tarea de localización para robots y cada una de estas conlleva otra buena cantidad de variantes. Generalmente se usan métodos probabilísticos como localización Monte Carlo o filtros de Kalman con todas sus variantes. Lo importante de estos métodos es que integran la información de los sensores internos del robot (por ejemplo edometría, giroscopios, acelerómetros etc.) con información de sensores exteroceptivos (cámaras, láser, sonares etc.).

Se han utilizado diferentes aproximaciones para resolver este problema en la liga de la RoboCup, siendo el filtro de partículas localización de Monte Carlo el más popular y exitoso. También hay equipos que usan el filtro extendido de Kalman, una combinación de localización de Monte Carlo y filtro extendido de Kalman, un método difuso o técnicas de visión (triangulación) para resolver el problema.

El filtro de Kalman es una técnica bien conocida para estimación de parámetros, siendo su principal inconveniente la necesidad de que el sistema se represente como una función lineal, además no asume que los sensores estén sujetos a ruido gaussiano, y que la creencia se puede representar como una función gaussiana. El filtro extendido de Kalman amplía la idea del filtro de Kalman intentando abordar problemas que no pueden ser descritos mediante ecuaciones lineales, aunque todavía no tiene en consideración el ruido y creencia gaussiano.

El filtro de Partículas basado en los procesos parcialmente observables de Markov supera el inconveniente de la linealidad y las asunciones gaussianas implementando un factor de muestreo. Este muestreo se realiza en el espacio de estados del robot, y mantiene una población de partículas que van evolucionando y agrupándose en las posiciones donde cree que el robot se encuentra.

Los resultados obtenidos en los diferentes proyectos realizados para la competición, muestran que los métodos propuestos tienen un buen desempeño. En cada competición se aprecia el avance realizado por cada equipo, por lo que se considera el trabajo realizado como una mejora integral al sistema de visión del equipo de fútbol robótico. No obstante, no deja de ser interesante buscar mejores aproximaciones y mejoras para lograr un mejor desempeño de los robots y aportar nuevos modelos en el campo de la navegación robótica móvil.

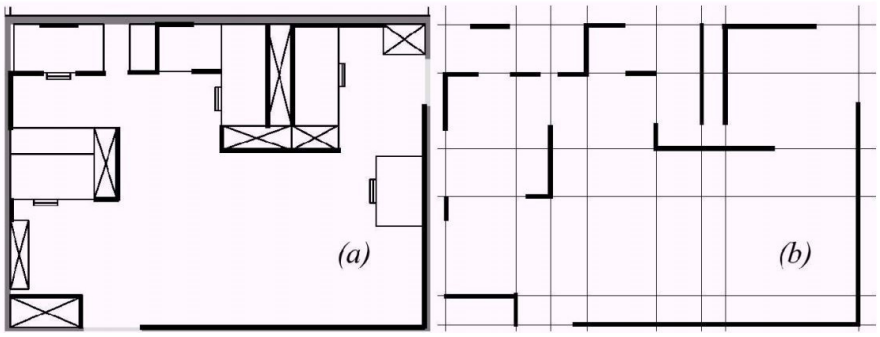
# Capítulo 1

# Marco Teórico

Para que un robot móvil pueda realizar satisfactoriamente tareas como generar trayectorias, evitar obstáculos o moverse en un entorno, se requiere que sea capaz de determinar su ubicación con respecto a un sistema de referencia absoluto. El determinar la ubicación de un robot móvil en un entorno equivale a encontrar las coordenadas de posición $(x, y, z)$ y de rotación $(\theta_x, \theta_y, \theta_z)$ del sistema de coordenadas del robot con respecto a un sistema absoluto. Sin embargo, la localización por sí sola no es suficiente, también se necesita contar con una representación del entorno sobre la cual ubicar el sistema. [1].

Con esta información un robot puede incluso trasladarse desde un punto inicial a un punto deseado. De acuerdo al tipo de entorno en el que se desenvuelven los robots, se puede ver al tema de localización desde dos perspectivas: la primera de ellas se refiere a la situación en que el robot se encuentra en una posición desconocida en un entorno desconocido, en este caso se suelen usar algoritmos que construyen un mapa del lugar y estiman la localización del robot conforme éste recorre el entorno, a estos algoritmos se les conoce como SLAM (del inglés Simultaneous Localization And Mapping, o en español *Localización Y Mapeado Simultáneos*). El segundo tipo de entorno corresponde a un robot que debe encontrar su posición en un entorno estructurado previamente conocido, en el que la posición de los puntos de referencia es fijo y se conocen previamente.

Figura 1.0.1: Representación de un entorno modelado y construido por un robot móvil *(b)* a partir de un entorno arquitectónico *(a)*.




Para el caso de localización en entornos previamente definidos, es posible diferenciar en la literatura dos tendencias principales de análisis, los métodos probabilísticos y no probabilísticos. Los métodos probabilísticos suelen tener menor error en la estimación de localización aunque requieren de un mayor uso de procesador ya que se debe tener un constante registro de la actividad del robot. Los métodos no probabilísticos aunque por lo general están más sujetos a errores, no necesitan de una constante actualización de datos y la respuesta de localización es inmediata ante cambios bruscos en la posición del robot.

Entre los métodos probabilísticos más usados se encuentran: redes bayesianas, adecuaciones del filtro de Kalman y cadenas de Markov. Mientras que en los no probabilísticos se encuentran métodos geométricos como: triangulación, trilateración y métodos específicos que hacen uso de equipo como radares, sonares o sensores de orientación. Ambos métodos poseen ventajas y desventajas dependiendo de la aplicación en que se utilicen. [2].

## 1.1. Localización y Mapeado

La localización de un robot móvil en un entorno conocido es uno de los problemas fundamentales de la robótica móvil, junto con la construcción automática de mapas del entorno. Se han propuesto un gran número de modelos, enfoques y técnicas para

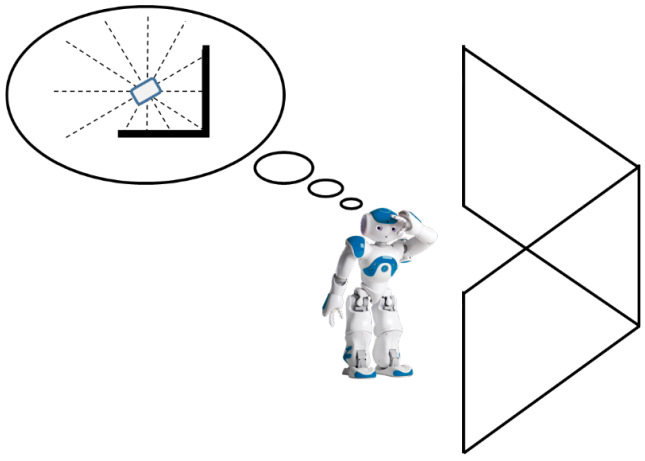
resolver ambos problemas. Muchas de estas técnicas presentan soluciones que sólo son útiles en situaciones muy específicas. Frente a este tipo de técnicas, han aparecido recientemente algunas propuestas generales que utilizan enfoques bayesianos, como por ejemplo los modelos de Markov o las redes bayesianas.

Los elementos fundamentales de la formulación de ambos problemas son la definición de un modelo de observación que proporciona la probabilidad de las lecturas de los sensores dada una posición en el entorno definido y la definición de un modelo de movimiento que proporciona la probabilidad de la siguiente posición del robot, dada la posición actual y la acción ejecutada. [3].

### 1.1.1. Modelos del Entorno

Un elemento fundamental de la localización y el mapeado es el modelo de representación del entorno. Un modelo o mapa del entorno es una abstracción y modelado de la información sensorial recibida por un robot móvil (ver Fig. 1.1.1) con la que se representan únicamente aquellas características del entorno que se consideran útiles para la navegación o la localización del robot. Al realizar esta abstracción se desechan características de grano fino que no se consideran útiles, debido a que pueden ser demasiado variables o no pueden ser detectadas con fiabilidad por los sensores.

Figura 1.1.1: Modelado de la información sensorial recibida por un robot móvil.



La utilidad principal de un modelo del entorno es proporcionar el elemento fundamental para la localización del robot. En general, los algoritmos de localización suelen comparar las lecturas obtenidas por los sensores del robot con el modelo del entorno, actualizando la posición del robot de forma acorde con el resultado de esta comparación. La forma de realizar la comparación depende totalmente del tipo de modelo de entorno y de la propuesta realizada.

Se han desarrollado dos paradigmas fundamentales de modelado de entornos: modelos topológicos y modelos métricos. A su vez, los modelos métricos pueden dividirse en: modelos basados en rejillas de ocupación y en modelos geométricos. En los siguientes apartados se revisarán los modelos de entorno más utilizados en la literatura, analizando las características y suposiciones de cada enfoque. [3].

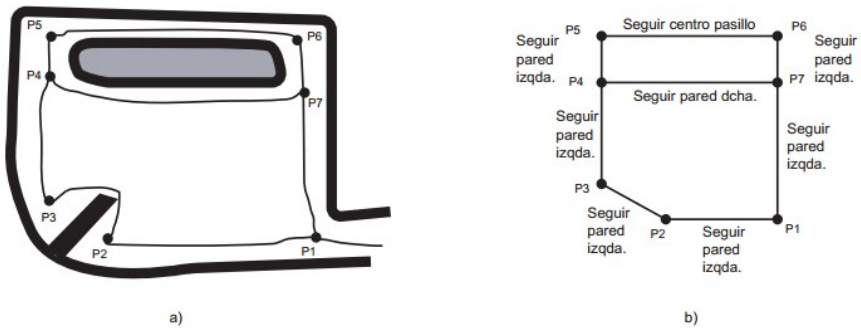
#### 1.1.1.1. Modelos Topológicos

La idea central de los modelos topológicos es representar las características esenciales del entorno percibidas por el robot móvil utilizando un grafo como un modelo de alto nivel. Los nodos se utilizan para representar lugares del entorno y los arcos caminos entre lugares.

Los lugares constituyen zonas del entorno *landmarks* (marcas u objetos pasivos inmersos en el entorno cuya posición relativa o absoluta es conocida) con características sensoriales distinguibles de forma absoluta, o respecto a sus nodos vecinos. Los nodos corresponden a la unidad elemental de localización, de manera que toda una zona geométrica del mapa real se representa por un único lugar.

A partir del mapa topológico no es posible distinguir localizaciones más finas que las representadas por los lugares. Veamos, como ejemplo, la propuesta de mapas topológicos de Kuipers [4], (ver Fig. 1.1.2).

Figura 1.1.2: Ejemplo de modelo topológico de Kuipers.




Los nodos corresponden a puntos distintivos del entorno y los arcos a caminos recorridos por el robot. Una posición del entorno correspondiente a un nodo debe distinguirse localmente de su vecindad mediante algún criterio definible en términos de los datos sensoriales. En el caso de los experimentos realizados por Kuipers, la función de distinción calcula el número de objetos cercanos que se encuentran a igual distancia del robot. Los arcos entre los nodos representan caminos que se han seguido para llegar de un nodo a otro utilizando una determinada estrategia de control local (seguir centro de pasillo, seguir pared a la derecha o seguir pared a la izquierda).

Un inconveniente de los mapas topológicos es que la necesidad de distinción sensorial entre lugares hace imposible la representación de zonas abiertas (habitaciones grandes, halls) en las que el alcance limitado de los sensores no obtiene información. Otro punto débil es que la definición de lugares y la conexión entre los mismos son muy dependientes de la aplicación, no utilizándose normalmente ningún criterio formal para su construcción.

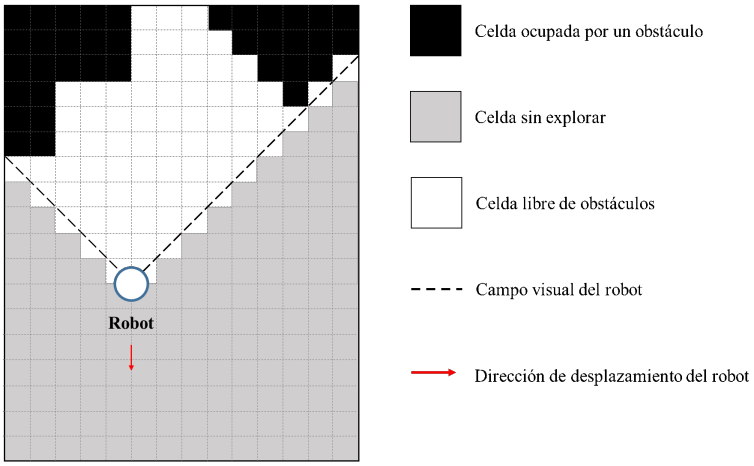
Por último, como se puede observar en el ejemplo de la Figura 1.1.2, los mapas construidos dependen excesivamente de la historia de las percepciones del robot al construirlos. Esto los hace muy sensibles a la aparición de elementos no modelados (personas, obstáculos imprevistos) que proporcionan información sensorial muy distinta de la modelada, haciendo que el robot pierda su localización. [3].

#### 1.1.1.2. Modelos Métricos

Representan las propiedades métricas o las coordenadas de los objetos del entorno (áreas, distancias, tamaño, localización, orientación, etc.). Este tipo de representación suele realizarse en el mismo sistema de coordenadas en dos dimensiones en el cual se representa al robot, lo que facilita el procesamiento de los datos que se obtienen mediante los sensores del robot. Existen dos tipos de representación en este sentido, las rejillas de ocupación y los modelos geométricos de entorno. [5].

##### 1.1.1.2.1. Rejillas de Ocupación

Figura 1.1.3: Ejemplo de rejilla de ocupación.




Los mapas basados en rejillas de ocupación dividen el entorno en una serie de celdas uniformes, pudiendo ser representado en dos dimensiones o tres dimensiones. Cada celda contiene un valor de probabilidad, que nos indica si está ocupada por un obstáculo, está libre o no ha sido explorada por el momento. Es posible además, utilizar rejillas de ocupación definidas por el usuario, pero lo usual es que sea el propio robot móvil el que realice la construcción de la rejilla de forma autónoma, mediante algún algoritmo de exploración.

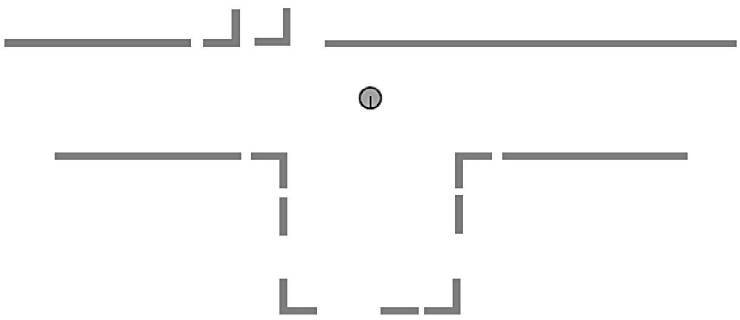
La mayor ventaja de este tipo de mapas es que son fáciles de construir y de mantener, incluso en entornos de gran espacio. Por otro lado, puesto que la geometría de las celdas

se corresponde con la geometría del terreno, es fácil para un robot determinar su posición dentro del mapa tan sólo conociendo su posición y orientación en el mundo real. Este tipo de representación facilita el cómputo de la trayectoria más corta. [6].

#### 1.1.1.2.2. Modelos Geométricos

Los modelos geométricos definen el entorno mediante sus características geométricas (distancias, dimensiones de los elementos que lo componen, posiciones absolutas). La ventaja principal de estos modelos es que, si se utilizan junto con un buen modelo de sensor, es posible simular los datos que los sensores del robot obtendrían en cualquier posición del entorno. Esto hace posible comparar los datos percibidos por el robot con los datos que se obtendrían en posiciones candidatas, calculándose una actualización de la probabilidad asociada a cada posición.

Figura 1.1.4: Ejemplo de modelo geométrico.



Existen distintos tipos de modelos geométricos. Un primer enfoque define el entorno mediante un conjunto de características geométricas (segmentos de rectas, esquinas) y mediante las relaciones geométricas entre ellas (distancia, posición, etc.). Otro enfoque, los modelos geométricos basados en características, se relacionan directamente con implementaciones de modelos de sensor en las que se utilizan estas características geométricas como elementos base del modelado. En la Figura 1.1.4 se muestra un ejemplo de mapa del entorno construido a base de las características geométricas definidas: esquinas, aristas y segmentos de rectas. [3].

### 1.1.2. Métodos Probabilísticos

Para solucionar el problema de estimar la posición de un robot móvil, se estima que los métodos probabilísticos tienen mucho que ofrecer en cuanto a robustez en la navegación, ya que permiten una representación constante y en tiempo de real de la incertidumbre de la posición global. Esto último posibilita también la definición de estrategias inteligentes que permitan reducir esta incertidumbre una vez se hayan evitado problemas (como el encuentro inesperado con un obstáculo móvil dentro del mapa).

Otros aspectos clave de los enfoques probabilísticos es que proporcionan una localización global más precisa que con modelos de localización basados en la subdivisión del mapa en celdas fijas, permitiendo además una fácil implementación. [7].

#### 1.1.2.1. Método Bayesiano

La estimación bayesiana para los robots móviles se basa en la definición probabilística de un modelo del entorno, un modelo de observación y un modelo de movimiento. El tipo de modelo del entorno (topológico o métrico) influye directamente en la formulación del modelo de observación, así como en el tratamiento del problema del mapeado. El uso de modelos métricos parametrizables, como los que definimos en este capítulo, permite estimar posiciones absolutas del robot, así como buscar, mediante algoritmos de mapeado, los parámetros del mapa que mejor explican una secuencia de movimientos y observaciones del robot.

El modelo de observación de las lecturas de sonares de un robot permite evaluar la probabilidad (verosimilitud) de que unas lecturas se hayan realizado en una determinada posición del entorno. Para que un modelo de observación se ajuste a la realidad hay que considerar una cierta probabilidad de que las lecturas hayan sido producidas por obstáculos o características no modeladas del entorno.

Por último, el modelo de movimiento evalúa la probabilidad de que el robot se encuentre en una posición nueva, dada la posición anterior del mismo y la acción realizada.

La calidad de los modelos de observación y de movimiento es la clave de una estimación bayesiana robusta y fiable. [3].

El teorema de Bayes determina la probabilidad de una causa sabiendo el efecto que ha producido. Sea un espacio muestral $E$, compuesto de un determinado número de sucesos disjuntos $A_i$, de tal manera que:

$$E = \bigcup_{i=1}^{n} A_i \tag{1.1.1}$$

La probabilidad de que ocurra un suceso $B|B \subseteq E$, se puede escribir como:

$$P(B) = P(\bigcup_{i=1}^{n} B \cap A_i) = \sum_{i=1}^{n} P(B \cap A) \tag{1.1.2}$$

Aplicando la ecuación de reducción del espacio muestral se obtiene la siguiente ecuación, que es el enunciado del teorema de la partición.

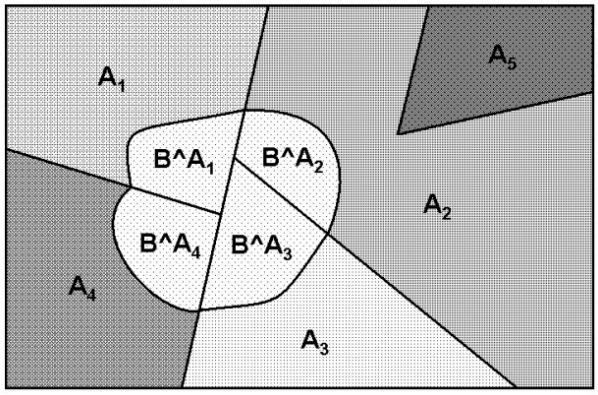
$$P(B) = \sum P(A_i)P(B|A_i) \tag{1.1.3}$$

$P(B)$ Probabilidad *a posteriori* de que ocurra el suceso $B$

$P(A_i)$ Probabilidad *a priori* de que ocurra la causa $A_i$

$P(B|A_i)$ Probabilidad condicional de que haya ocurrido el suceso $B$, dado que existe la causa $A_i$

Figura 1.1.5: Representación de un espacio muestral $E$.



La fusión de datos consiste en combinar varias piezas de información de un mismo fenómeno pero emanadas de diferentes fuentes, con la finalidad de tomar la mejor decisión respecto a este mismo fenómeno. La información extraída de cada sensor es representada mediante números reales como el grado de confianza en cierto evento, tomando así en cuenta la imprecisión, la incertidumbre y la naturaleza incompleta de la información. En el caso de la fusión Bayesiana, el grado de confianza está representado por probabilidades (*a priori*, condicional y *a posteriori*). Las decisiones son usualmente tomadas a partir de una probabilidad *a posteriori*.

Sea $A$ el evento a ser evaluado, y $x_1, x_2$ la información proveniente de dos sensores distintos. En este caso cada sensor asigna una probabilidad al evento $A$ dada cierta medida $x$. Así, dada una cierta medida $x_1$, la misma es convertida a un valor de probabilidad mediante el modelo inverso del sensor $P(x_1|A)$. Este modelo se relaciona con las propiedades del sensor mediante la regla de Bayes según,

$$P(A|x_1) = \frac{(P(x_1|A)}{P(x_1)}P(A) \tag{1.1.4}$$

De la misma forma, el modelo inverso del sensor *2* estaría relacionada con el evento $A$ a través de la ecuación,

$$P(A|x_2) = \frac{(P(x_2|A)}{P(x_2)}P(A) \tag{1.1.5}$$

La probabilidad de observación del evento $A$ dado las mediciones $x_1$ y $x_2$ se puede derivar de la regla del producto,

$$P(A|x_1, x_2) = \frac{(A|x_1, x_2)}{P(x_1|x_2)}P(A|x_2) \tag{1.1.6}$$

Asumiendo la independencia de las mediciones $x_1$ y $x_2$, entonces $P(A|x_1, x_2)$ y $P(x_1|x_2)$ corresponden a $P(x_1|A)$ y $P(x_1)$ respectivamente, transformando la probabilidad conjunta anterior en:

$$P(A|x_1, x_2) = \frac{(x_1|A)}{P(x_1)}P(A|x_2) \tag{1.1.7}$$

Aplicando nuevamente la regla de Bayes a la probabilidad condicional $P(x_1|A)$, se obtiene la densidad de probabilidad conjunta,

$$P(A|x_1, x_2) = \frac{P(A|x_1)P(A|x_2)}{P(A)} \tag{1.1.8}$$

La función anterior representa los estimados y los valores de certidumbre del evento $A$ a partir de la fusión de las medidas $x_1$ y $x_2$ de dos sensores diferentes. [8].

#### 1.1.2.1.1. Inferencia bayesiana

La incertidumbre es natural en el proceso de razonamiento donde se pueden establecer reglas para inferir de manera deductiva una proposición determinada que puede ser verdadera o falsa, según sea el límite de esta estimación. Dentro de los métodos de razonamiento se encuentran los modelos bayesianos, que simulan diferentes condiciones de incertidumbre cuando no se conoce si es verdadera o falsa la hipótesis enunciada en un rango de variación.

Todos los modelos bayesianos tienen en común la asignación de la probabilidad como medida de creencia de una hipótesis, así es que, la inferencia es un proceso de reajuste de medidas de creencia al conocerse nuevos axiomas.

Cuando se utilizan evidencias y observaciones para establecer que una suposición sea cierta, es lo que se denomina como Inferencia Bayesiana. La inferencia bayesiana observa la evidencia y calcula un valor estimado según el grado de creencia planteado en la hipótesis. Esto implica que al tener mayor cantidad de datos disponibles se podrá obtener resultados más satisfactorios. El uso de la inferencia bayesiana en los casos donde las distribuciones normales se aplican, requiere que sólo dos actos decisión sean evaluados en cierto momento.

La distribución de probabilidad *a priori* es descriptiva con respecto a la incertidumbre y asocia la estimación de probabilidad con la ocurrencia de un evento al azar. Por el contrario la distribución de probabilidad, más bien, es la estimación de un evento

dado que se basa en un juicio informado. Para situaciones de sentencia en la que una decisión es informada se debe ser consciente de una serie de factores de incertidumbre que podrían influir en el valor de los resultados finales.

El uso de la distribución normal puede ser una solución satisfactoria para hallar una aproximación de incertidumbre; la distribución normal se define mediante la identificación de la media y la desviación estándar de la distribución. La media de la distribución normal se puede obtener mediante la identificación el valor más probable al evento aleatorio. [9].

#### 1.1.2.2. Filtro de Kalman

El KF (del inglés *Kalman Filter*, o en español Filtro de Kalman) es un conjunto de ecuaciones matemáticas que proporcionan términos medios computacionales eficientes para estimar el estado de un proceso, de forma que minimiza el término medio del error. Este filtro es muy potente en varios aspectos: soporta estimaciones del pasado, presente e incluso estados del futuro, y también puede hacerlo cuando la precisión del sistema modelado es desconocida.

El KF es un medio para calcular la probabilidad que se representa mediante una distribución gaussiana. Formalmente, el KF estima la probabilidad condicional de que el sistema esté en el estado $x_t$ a partir de las acciones $a_1$ ... $a_t$ y las medidas $z_1$ ... $z_t$, es decir, estimar la función de densidad que representa la probabilidad *a posteriori*, [10].

$$p(x_t|z_1...z_t; a_1...a_{t-1}) \tag{1.1.9}$$

La aplicación del KF a la localización de robots móviles estima la posición $(x, y, \theta)$ del robot en el entorno mediante una distribución normal. La covarianza de esta distribución representa la incertidumbre local en la posición estimada. Las observaciones realizadas por los sensores se utilizan para actualizar la distribución de probabilidad de la localización, buscando la nueva distribución que maximiza la verosimilitud de las lecturas.

El KF resulta ser un excelente mecanismo recursivo para fusionar información redundante proveniente de diferentes sensores o sistemas lineales. Cuando la incertidumbre que caracteriza la información se distribuye de manera normal, el KF representa un filtro estadístico óptimo en el sentido que se minimiza la varianza del error. La aplicación de este filtro en sistemas no lineales se hace mediante una extensión conocida como EKF (del inglés, *Extended Kalman Filter*, o en español Filtro Extendido de Kalman), la cual surge de la linealización de las ecuaciones de estado mediante la expansión en Series de Taylor de primer orden alrededor del estado estimado. [8].

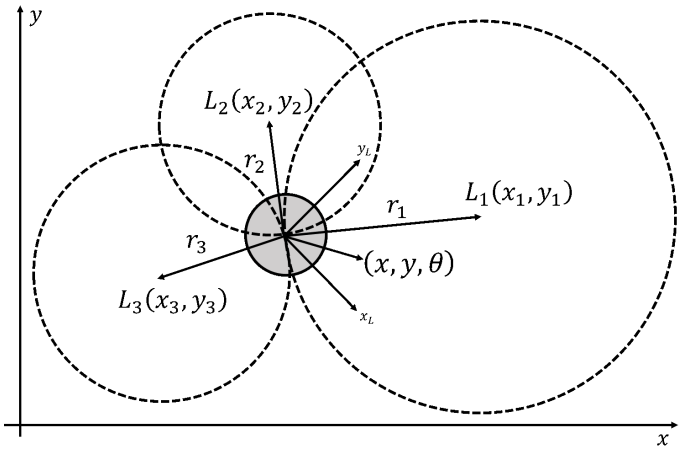
El filtro de Kalman es básicamente un conjunto de ecuaciones matemáticas mediante las cuales se estima el estado de un sistema lineal en forma e?ciente, a través de una solución recursiva del método de mínimos cuadrados. Estas ecuaciones permite implementar un estimador lineal, insesgado y óptimo, del estado de un proceso, basándose en la información disponible en el instante anterior, y corrigiendo este estado estimado a partir de la información disponible en el momento actual.

### 1.1.3. Métodos no Probabilísticos

Los métodos no probabilísticos comúnmente utilizados para determinar la posición de un robot móvil a partir de la detección de marcas, activas y pasivas, son dos: La trilateración, el cual se basa en las distancias desde el móvil a cada una de las landmarks, normalmente tres o más, y la triangulación, el cual se basa en los ángulos respecto a las diferentes landmarks.

#### 1.1.3.1. Trilateración

Figura 1.1.6: Esquema de localización por trilateración.




La trilateración es la determinación de localización $(x, y)$ de un robot móvil basado en la medición de las distancias $(r_1, r_2, r_3)$ a 3 o más *landmarks* cuyas coordenadas $(x_1, y_1)(x_2, y_2)(x_3, y_3)$ son conocidas, (ver Fig. 1.1.6). En un plano **2D**, la trilateración se puede definir como el problema de encontrar la intersección de tres circunferencias. Esto es encontrar la solución al sistema de ecuaciones cuadráticas expresadas en la ecuación (33).

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2 \tag{1.1.10}$$

$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$

$$(x - x_3)^2 + (y - y_3)^2 = r_3^2$$

Una forma simple de resolver el sistema dado, es sustrayendo la segunda y la tercera ecuación de la primera ecuación. Así, el sistema se reduce a un sistema de primer orden con dos ecuaciones y dos incógnitas según,

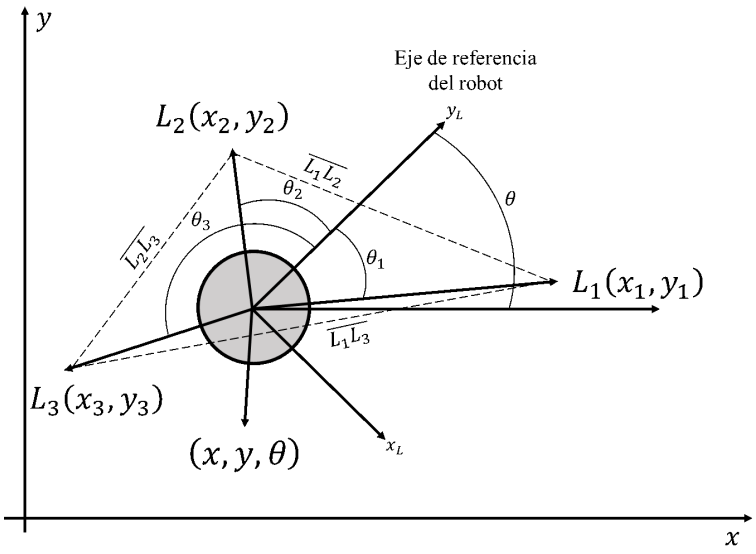
$$x_1^2 - x_2^2 - 2x(x_1 - x_2) + y_1^2 - y_2^2 - 2y(y_1 - y_2) = r_1^2 - r_2^2 \tag{1.1.11}$$

$$x_1^2 - x_3^2 - 2x(x_1 - x_3) + y_1^2 - y_3^2 - 2y(y_1 - y_3) = r_1^2 - r_3^2$$

La implementación práctica de la trilateración, usualmente, consiste en instalar 3 o más marcas en lugares conocidos del entorno, e instalar un único receptor aborde del móvil. A pesar de la simplicidad y buena precisión que ofrece el método, en robótica móvil, este planteamiento de localización es insuficiente, ya que sólo se determina las coordenadas cartesianas del robot $(x, y)$, no obteniendo ninguna información referente a la orientación del robot respecto al sistema de coordenadas global. [8].

#### 1.1.3.2. Triangulación

Figura 1.1.7: Esquema de localización por triangulación.




Por su parte, la triangulación es el proceso de determinar la localización del robot en el que, aparte de la posición $(x, y)$, también obtendríamos información acerca de la orientación $(\theta)$. Para ello se hace uso de tres landmarks $(L_1, L_2, L_3)$ cuyas respectivas localizaciones en el espacio cartesiano $(x_i, y_i)$ son conocidas. En general, el proceso requiere que el robot tenga capacidad sensorial para detectar estas marcas y para medir la orientación o ángulo de vista hasta cada una de ellas $(\theta_1, \theta_2, \theta_3)$, respecto al sistema de referencia de él mismo, (ver Fig. 1.1.7). Aplicando la ley de coseno, la distancia entre dos marcas cualesquiera se puede expresar como:

$$r_1^2 + r_2^2 - 2r_1 r_2 cos\theta_{12} = \overline{(L_1 L_2)} \tag{1.1.12}$$

$$r_1^2 + r_3^2 - 2r_1 r_3 cos\theta_{13} = \overline{(L_1 L_3)}$$

$$r_2^2 + r_3^2 - 2r_2 r_3 cos\theta_{23} = \overline{(L_2 L_3)}$$

Aquí, $(r_1, r_2, r_3)$ son las distancias desconocidas desde el robot hasta las *landmarks*, y $(\overline{(L_1 L_2)}, \overline{(L_1 L_3)}, \overline{(L_2 L_3)})$ son las distancias entre marcas que se consideran conocidas. Igualmente $(\theta_{12}, \theta_{13}, \theta_{23})$ surgen de sustraer los ángulos $\theta_1, \theta_2, \theta_3$, entre sí, por lo que también se consideran conocidos.

El sistema de ecuaciones 1.1.12 puede resolverse utilizando un método de aproximación por mínimos cuadrados para obtener así las distancias desconocidas $(r_1, r_2, r_3)$. Una vez obtenidas estas distancias, se pasa a resolver la localización $(x, y)$ del móvil usando el procedimiento descrito para las ecuaciones 1.1.10 y 1.1.11.

Una vez obtenida la localización del robot móvil, se puede computar el ángulo del segmento que desde el punto $(x, y)$ hasta cualquiera de las marcas conocidas $(x_i, y_i)$ y luego calcular el ángulo $\theta$ de orientación del móvil con la ecuación. [8].

$$\theta = \begin{cases} (x, y) \\ (x_i, y_i) \end{cases} - \theta_i(36) \tag{1.1.13}$$

# Capítulo 2

# Morfología Matemática

## 2.1. Procesamiento Morfológico de Imágenes

[11] La palabra morfología comúnmente denota una rama de la biología que se ocupa de la forma y estructura de los animales y las plantas. Usamos la misma palabra aquí en el contexto de la morfología matemática como una herramienta para la extracción de componentes de una imagen que son útiles en la representación y descripción de la región de forma, tales como técnicas de filtrado morfológicas de pre o pos procesamiento.

A lo largo de esta sección definimos dos operaciones morfológicas fundamentales, la dilatación y la erosión, en términos de la unión (o intersección) de una imagen con una forma traducida o llamada elemento estructurante, al igual que observaremos la combinación de la erosión y la dilatación para obtener operaciones morfologías más complejas.

### 2.1.1. Teoría de Conjuntos

Sea $Z$ el conjunto de los enteros reales. El proceso de muestreo utilizado para generar imágenes digitales puede ser visto como la partición del plano $xy$ en una cuadrícula, siendo la coordenada del centro de cada rejilla un par de elementos a partir del producto cartesiano, $Z^2t$. En la terminología de la teoría de conjuntos, una función $f(x, y)$ se dice que es una imagen digital si $(x, y)$ son números enteros de $Z^2$ y $f$ es un mapeo

que asigna un valor de la intensidad (es decir, un número real a partir del conjunto de los números reales, $R$) para cada par de coordenadas $(x, y)$. Si los elementos de $R$ son números enteros, una imagen digital se convierte en una función de dos dimensiones cuyos valores de coordenadas y amplitud (intensidad) son números enteros.

Sea $A$ un conjunto en $Z^2$, los elementos de cada pixel son las coordenadas $(x, y)$. Si $w(x, y)$ es un elemento de $A$, entonces escribimos

$$w \in A \tag{2.1.1}$$

Del mismo modo, si $w$ *no* es un elemento de $A$, escribimos

$$w \notin A \tag{2.1.2}$$

Un conjunto $B$ de coordenadas de píxeles que satisfacen una *condición* particular se escribe como

$$B = \{w|condicion\} \tag{2.1.3}$$

Por ejemplo, el conjunto de todas las coordenadas de píxeles que no pertenecen al conjunto $A$, está denotado por $A'$ y es llamado *complemento* de $A$.

$$A' = \{w|w \in A\} \tag{2.1.4}$$

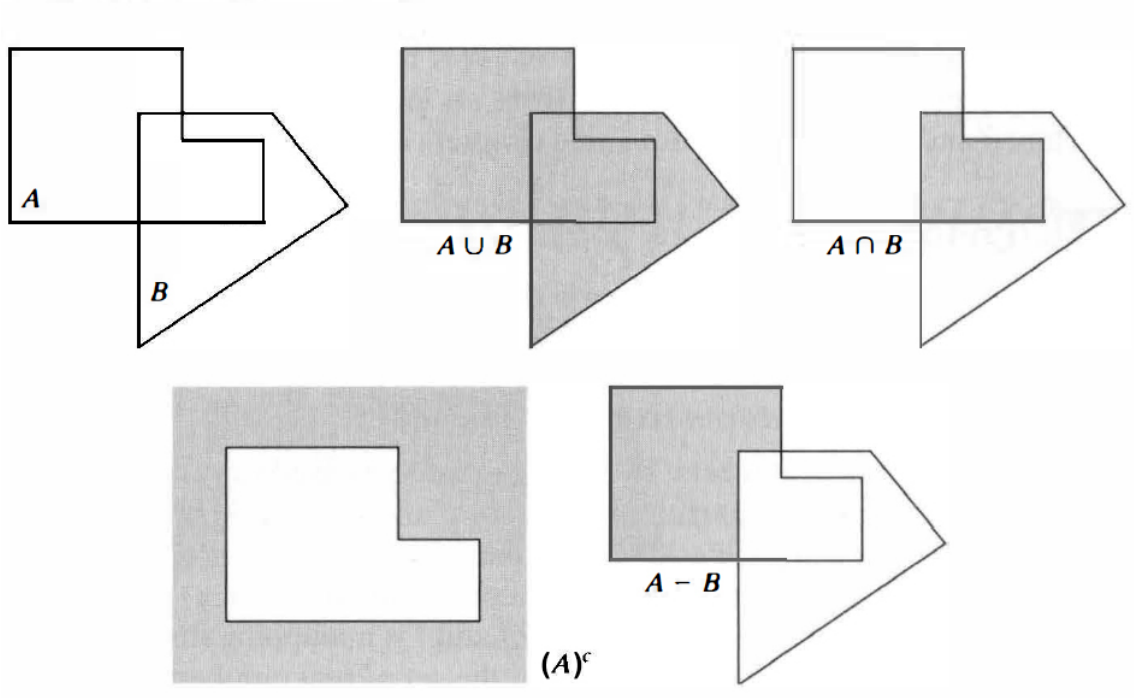
La *unión* de dos conjuntos $A$ y $B$, es el conjunto de todos los elementos que pertenecen a ambos conjuntos, denotado por

$$C = A \cup B \tag{2.1.5}$$

Del mismo modo, la *intersección* de los conjuntos $A$ y $B$, es el conjunto de todos los elementos que tienen en común ambos conjuntos, denotado por

$$C = A \cap B \tag{2.1.6}$$

Figura 2.1.1: (a) Conjuntos $A$ y $B$. (b) Unión de $A$ y $B$. (c) Intersección de $A$ y $B$. (d) Complemento de $A$. (e) Diferencia entre $A$ y $B$.




La *diferencia* de los conjuntos $A$ y $B$, denotada como $A - B$, es el conjunto de todos los elementos que pertenecen a $A$, pero no a $B$:

$$A - B = \{w | w \in A, w \notin B\} \tag{2.1.7}$$

Figura 2.1.1 ilustra las operaciones de conjuntos definidos hasta el momento. El resultado de cada operación se muestra en gris. Además de las operaciones básicas, las anteriores operaciones morfológicas a menudo requieren dos operadores que son específicos para conjuntos cuyos elementos son las coordenadas del píxel. La *reflexión* de un conjunto $B$, denotada por $B'$, se define como

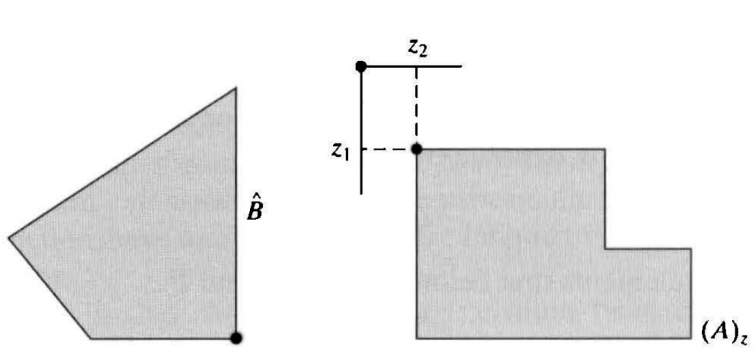
$$\hat{B} = \{w | w = -b \ para \ b \in B\} \tag{2.1.8}$$

La *traducción* del conjunto $A$ por punto $z = (z_1, z_2)_z$ denotado $(A)_z$, se define como

$$(A)_z = \{c | c = a + z \ para \ a \in A\} \tag{2.1.9}$$

Figura 2.1.2 ilustra estas dos definiciones que utilizan los conjuntos de la Figura 2.1.1 el punto negro indica el origen de los conjuntos (el punto de origen es una referencia definida por el usuario).

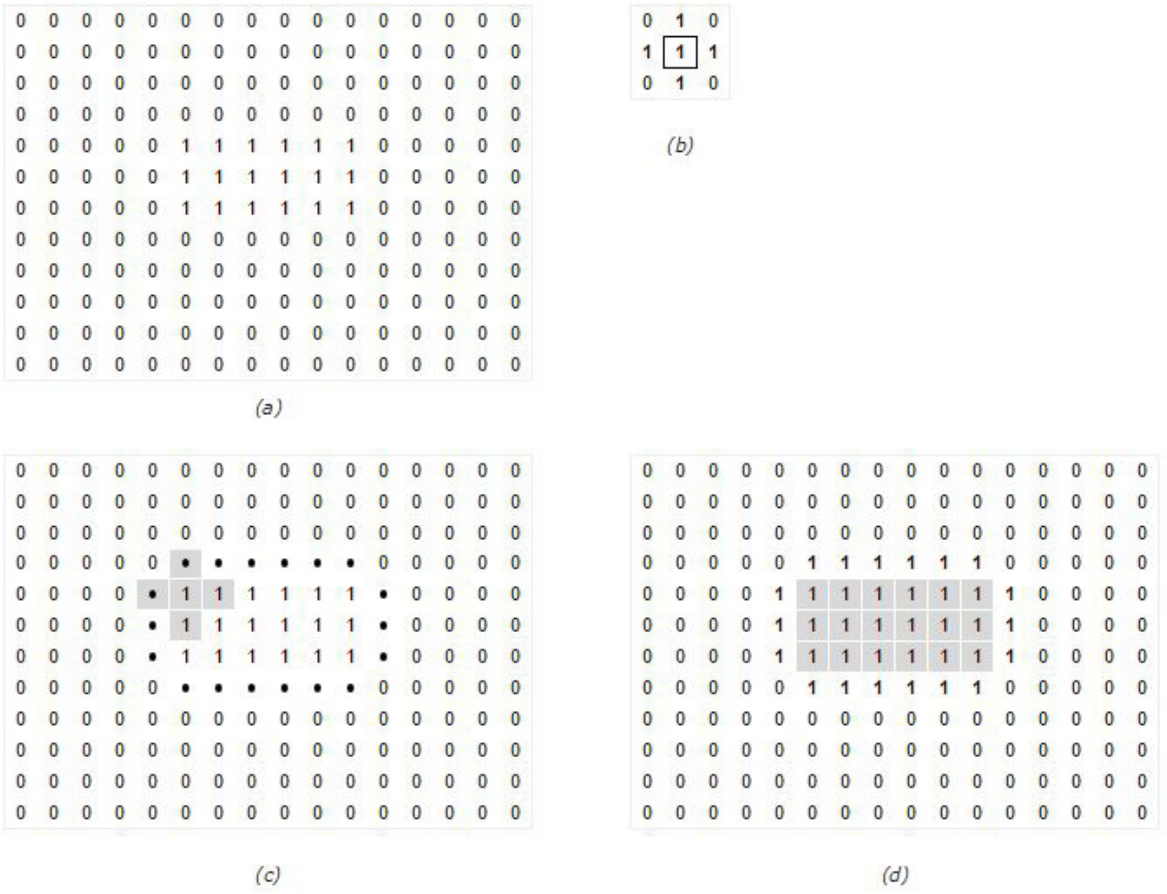
Figura 2.1.2: (a) Reflexión de $B$. (b) Traducción de $A$ por $z$. Los conjuntos $A$ y $B$ son de la Figura 2.1.1, y el punto negro denota su origen.




## 2.1.2. Dilatación

La dilatación es una operación que "*crece*" o "*engrosa*" objetos de una imagen. La manera específica y el alcance de este engrosamiento se controla por medio de un elemento estructurante. La figura 2.1.3 ilustra cómo funciona la dilatación. Figura 2.1.3 (a) muestra una imagen binaria que contiene un objeto rectangular. Figura 2.1.3 (b) es un elemento estructurante, en este caso una matriz de cinco píxeles. Gráficamente el elemento estructurante puede ser representado ya sea por una matriz de ceros y unos o como un conjunto de primer plano de píxeles de valor 1, en ambos casos se muestra el origen del elemento estructurante delineado por un recuadro negro. Figura 2.1.3 (c) representa la dilatación como un proceso que traslada el origen del elemento estructurante a el dominio de la imagen y el elemento estructurante se superponen por un valor de pixel 1. Figura 2.1.3 (d) imagen binaria de salida dilatada.

Figura 2.1.3: Figura de dilatación. (a) Imagen original con objeto rectangular. (b) Elemento estructurante con cinco píxeles dispuestos en forma de cruz, el origen o centro del elemento estructurante se muestran con un borde oscuro. (c) Elemento estructurante trasladado a varios lugares en la imagen. (d) Imagen de salida, la región sombreada muestra la ubicación de unos [1$s$] en la imagen original.




La dilatación de $A$ por $B$ denotado $A \oplus B$ , se define como la operación conjunto

$$A \oplus B = \{z|(\hat{B})_z \cap A \neq \oslash\} \tag{2.1.10}$$

donde $\oslash$ es el conjunto vacío y $B$ es el elemento estructurante. En palabras, la dilatación de $A$ por $B$ es el conjunto formado por todos los lugares de origen del ele-

mento estructurante donde el reflejado y traducido de $B$ se superpone al menos en un elemento de $A$. Es una convención en el procesamiento de imágenes para que el primer operando de $A \oplus B$ sea la imagen y el segundo operando sea el elemento estructurante, que por lo general es mucho más pequeño que la imagen. La traducción del elemento estructurante de la dilatación es similar a la mecánica de convolución espacial.

La dilatación es una operación matemática *asociativa:*

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C \tag{2.1.11}$$

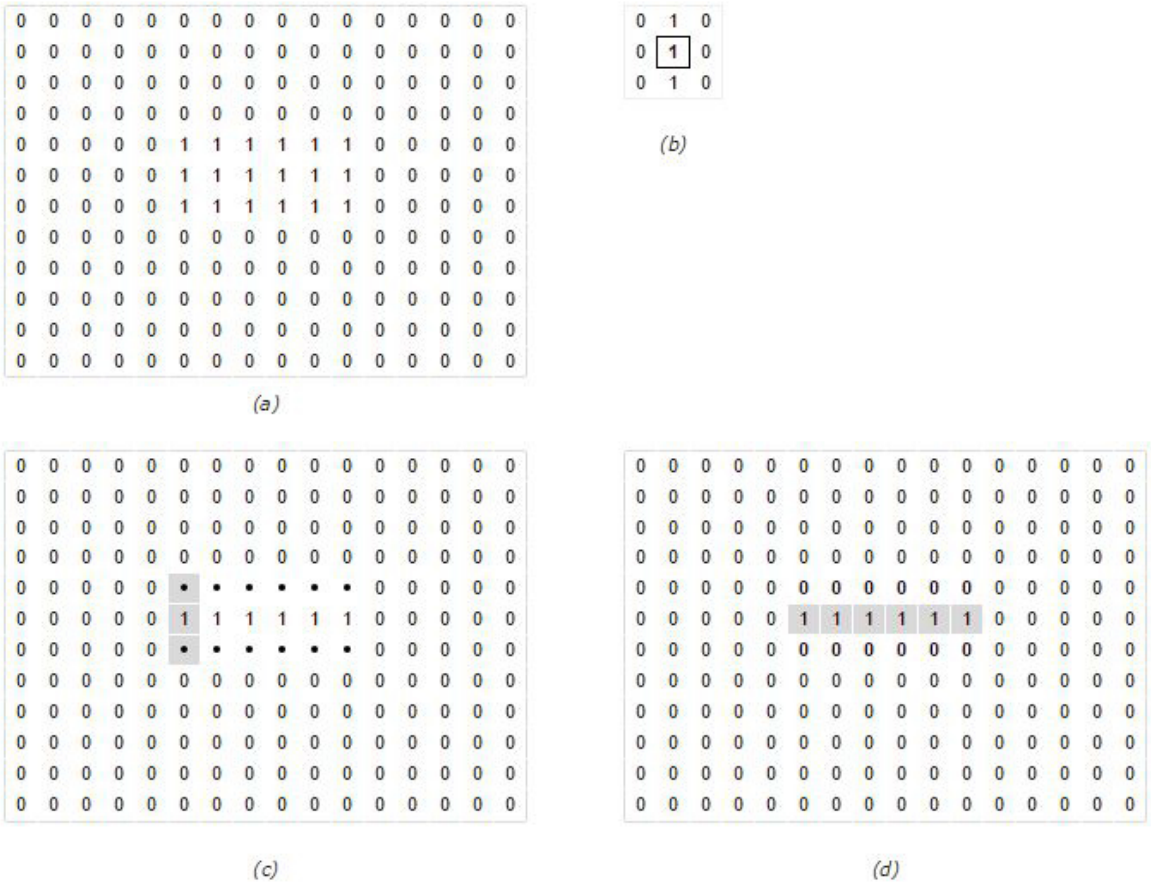
y *conmutativa:*

$$A \oplus B = B \oplus A \tag{2.1.12}$$

### 2.1.3. Erosión

La erosión es una operación que "*encoge*" o "*adelgaza*" objetos en una imagen binaria. Al igual que en la dilatación, la manera y el grado de adelgazamiento es controlado por un elemento estructurante. La figura 2.1.4 ilustra el proceso de erosión. Figura 2.1.4 (a) es la misma que la Figura 2.1.3 (a). 2.1.4 (b) es el elemento estructurante, una línea vertical de 3 píxeles. Figura 2.1.4 (c) representa erosión gráficamente como un proceso de traducción del elemento estructurante a través el dominio de la imagen y la comprobación para ver dónde encaja totalmente dentro el primer plano de la imagen. La imagen de salida en la Figura 2.1.4 (d) tiene un valor de 1 en cada ubicación del origen del elemento estructurante, de tal manera que el elemento se superpone en sólo 1 valor de píxel de la imagen de entrada (es decir, que no se superponga ninguna del fondo de la imagen).

Figura 2.1.4: Figura de erosión. (a) Imagen original con objeto rectangular. (b) Elemento estructurante con tres píxeles dispuestos en una línea vertical. El origen del elemento estructurante se muestra con un borde oscuro. (c) Elemento estructurante trasladado a varios lugares en la imagen. (d) Imagen de salida.




La erosión de $A$ por $B$, denotada por $A \ominus B$, se define como

$$A \ominus B = \{z|(B)_z \subseteq A\} \tag{2.1.13}$$

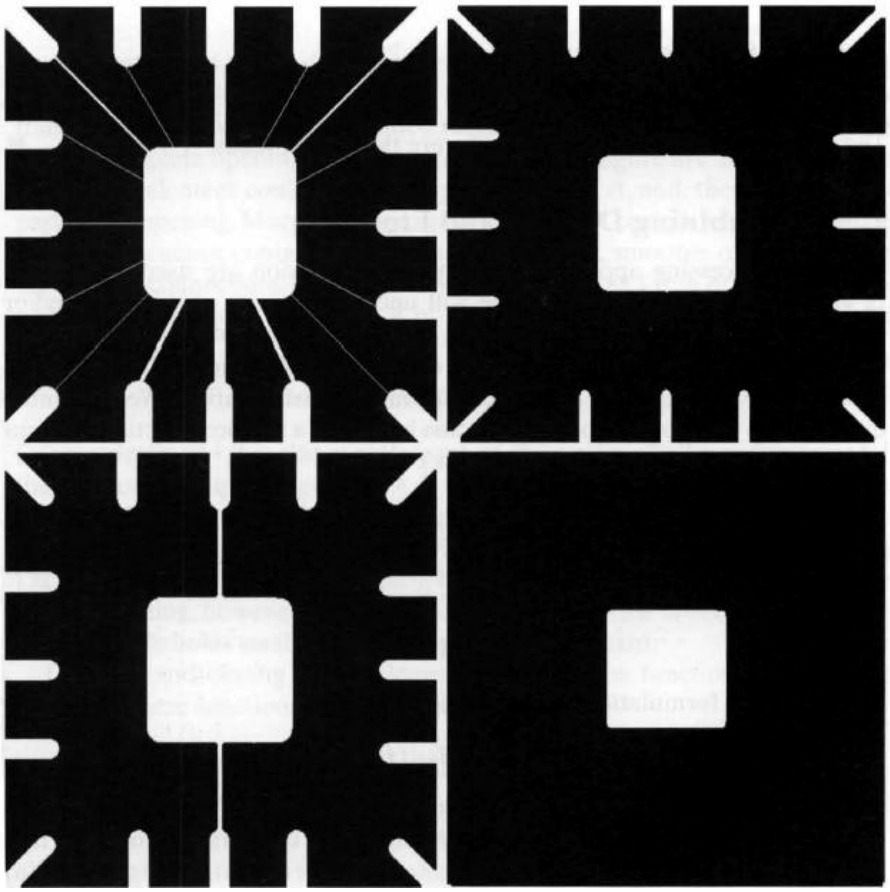
donde, como de costumbre, la notación $C \subseteq D$ significa que $C$ es un subconjunto de $D$. Esta ecuación dice que la erosión de $A$ por $B$ es el conjunto de todos los puntos $z$ tal que $B$, trasladada por $z$, esta contenida en $A$. Debido a la afirmación de que $B$ está contenido en $A$ es equivalente a que $B$ no comparte ningún elemento con el

fondo de $A$, podemos escribir la siguiente expresión equivalente según la definición de la erosión:

$$A \ominus B = \{z|(B)_z \cap A^c = \oslash\} \tag{2.1.14}$$

Aquí, la erosión de $A$ por $B$ es el conjunto formado por todos los lugares de origen del elemento estructurante donde ninguna parte de $B$ se superpone al fondo de $A$.

Figura 2.1.5: Ilustración de erosión. (a) Imagen original de tamaño 486 x 486 píxeles. (b) Erosión con un disco de radio 10. (c) La erosión con un disco de radio 5. (d) Erosión con un disco de radio 20.



### 2.1.4. Combinando dilatación y erosión

Las operaciones de dilatación y erosión son fundamentales para el procesamiento de imágenes morfológicas.

En las aplicaciones de procesamiento de imágenes, la dilatación y la erosión son los más utilizados en diversas combinaciones. Una imagen se someterá a una serie de dilataciones y erosiones utilizando los mismos, o en ocasiones diferentes elementos estructurantes. En esta sección consideramos dos de las combinaciones más comunes de dilatación y erosión: apertura (opening) y cierre (closing).

#### 2.1.4.1. Apertura

La *apertura morfológica* de $A$ por $B$, denotado $A \circ B$, se define como la erosión de $A$ por $B$, seguida de una dilatación del resultado por $B$:

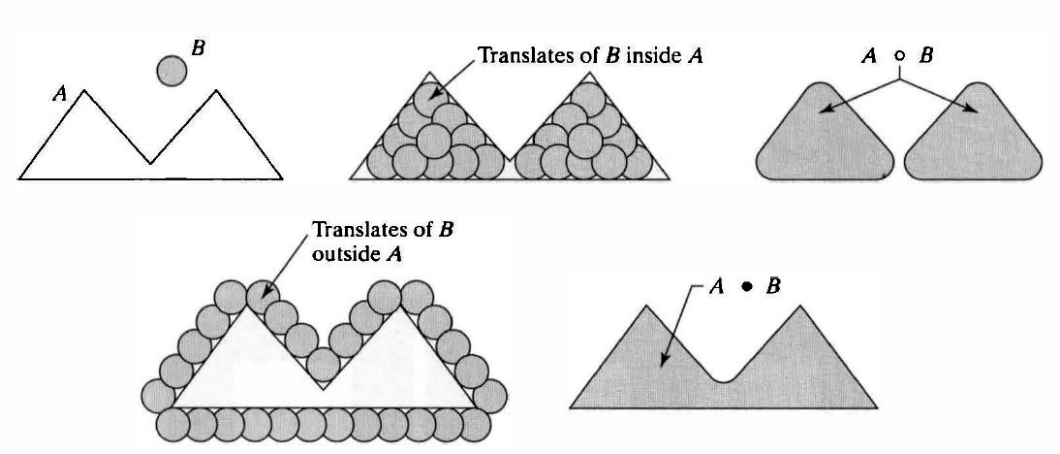
$$A \circ B = (A \ominus B) \oplus B \tag{2.1.15}$$

La fórmula equivalente de apertura es

$$A \circ B = \cup\{(B)_z | (B)_z \subseteq A\} \tag{2.1.16}$$

donde $\cup\{.\}$ denota la unión de todos los conjuntos dentro de las llaves. Esta formulación tiene una sencilla interpretación geométrica: $A$ o $B$ es la unión de todas las traducciones de $B$ que caben completamente dentro de $A$. Figura 2.1.6 ilustra esta interpretación. Figura 2.1.6 (a) muestra un conjunto $A$ y un elemento estructurante en forma de disco, $B$. Figura 2.1.6 (b) muestra algunas de las traducciones de $B$ que se ajustan completamente dentro de $A$. El resultado de la unión de todas las traducciones en las dos regiones sombreadas de la Figura 2.1.6 (c); estas dos regiones son la apertura completa. Las regiones blancas en esta figura son áreas en las que los elemento estructurante no podría encajar completamente dentro de $A$, y, por tanto, no son parte de la apertura. La apertura morfológica elimina completamente las regiones de un objeto que no puede contener el elemento estructurante, suaviza los contornos de objetos, rompe conexiones delgadas [como en la Figura 2.1.6 (c)], y elimina las protuberancias delgadas.

Figura 2.1.6: La apertura y cierre traducido como las uniones de elementos estructurantes. (a) Conjunto $A$ y elemento estructurante $B$. (b) Traducciones de $B$ que se ajustan completamente dentro del conjunto $A$. (c) Apertura completa (sombreado). (d) Las traducciones de $B$ fuera de la frontera de $A$. (e) Cierre completo (sombreado).



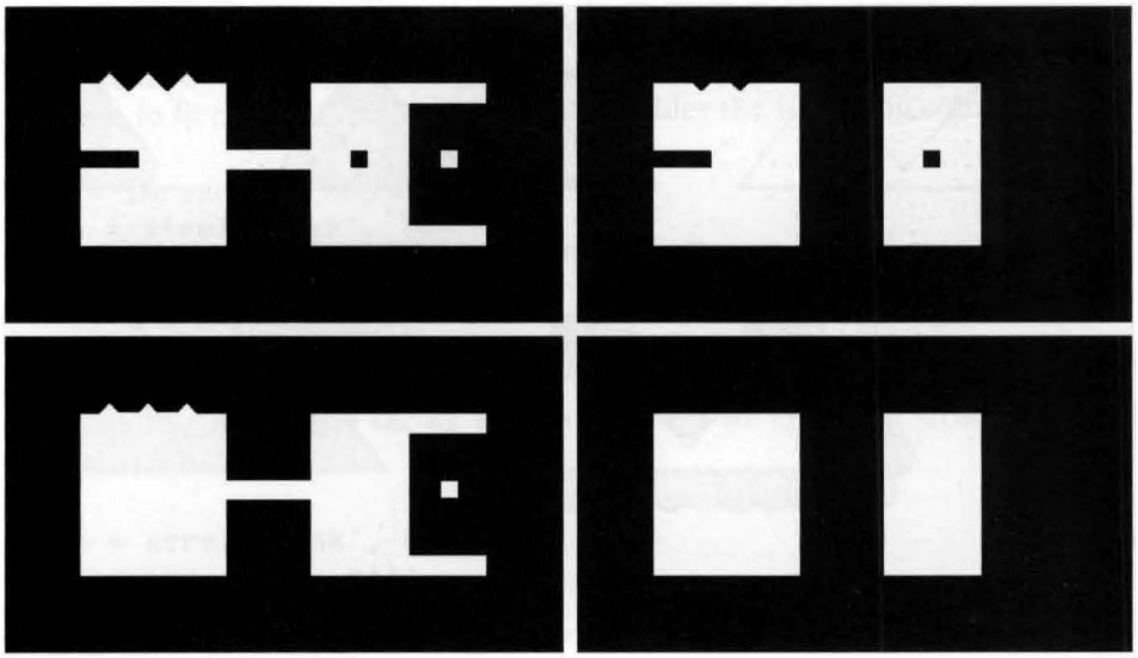

### 2.1.4.2. Cierre

El *cierre morfológico* de $A$ por $B$, denotado $A \cdot B$, se define como la dilatación de $A$ por $B$, seguida de por una erosión del resultado por $B$:

$$A \cdot B = (A \oplus B) \ominus B \tag{2.1.17}$$

Geométricamente, $A \oplus B$ es el complemento de la unión de todas las traducciones de $B$ que no se superponen en $A$. Figura 2.1.7 (d) ilustra varias traducciones de $B$ que no se superponen en $A$. Tomando el complemento de la unión de todas dichas traducciones, obtenemos la región sombreada Figura 2.1.7 (e), que es el cierre completo. Como la apertura, el cierre morfológico tiende a suavizar los contornos de los objetos. A diferencia de la apertura, sin embargo, el cierre generalmente une estrechas aberturas, llena golfos largos y delgados y los agujeros más pequeños que el elemento estructurante.

Figura 2.1.7: Ejemplo de apertura y cierre. (a) Imagen original. (b) Apertura. (c) Cierre. (D) Cierre de ($b$).



### 2.1.5. Imágenes binarias, conjuntos y operadores lógicos

Los operadores &, | y $\sim$ son los operadores de lógica *and*, *or* y *not* respectivamente. El resultado de $C = A\&B$ es una matriz cuyos elementos son unos donde $A$ y $B$ sean ambos distintos de cero, y ceros donde $A$ ó $B$ sean cero. $A$ y $B$ deben de ser matrices con las mismas dimensiones, a menos que una de ellas sea un escalar. El resultado de $C = A|B$ es una matriz cuyos elementos son unos donde $A$ ó $B$ tienen un elemento diferente de cero, y ceros donde ambas tienen elementos cero. $A$ y $B$ deben de ser matrices con las mismas dimensiones, a menos que una sea un escalar.

El resultado de $B = \sim A$ es una matriz cuyos elementos son uno donde $A$ tiene un elemento cero, y ceros donde $A$ tiene elementos diferentes de cero.

El lenguaje y la teoría de la morfología matemática a menudo presentan un doble (pero equivalente) vista de imágenes binarias. Conjuntos de operaciones como unión

e intersección puede ser aplicado directamente a conjuntos de imágenes binarias. Por ejemplo, si $A$ y $B$ son imágenes binarias, entonces $C = A \cup B$ es una imagen binaria también, cuando un píxel en $C$ es un píxel de primer plano, si uno o ambos de los píxeles correspondientes de $A$ y $B$ son píxeles de primer plano.

En la primera vista, el de una función, $C$ está dada por

$$C(x,y) = \begin{cases} 1 \textit{ si cualquiera } A(x,y) \textit{ ó } B(x,y) \textit{ es } 1 \textit{ ó si ambos son} 1 \\ 0 \textit{ de otra manera} \end{cases} \tag{2.1.18}$$

Por otra parte, utilizando el punto de vista de conjunto, $C$ está dada por

$$C(x,y) = \{(x,y) | (x,y) \in A \textit{ ó } (x,y) \in B \textit{ ó } \in (A \,\&\, B)\} \tag{2.1.19}$$

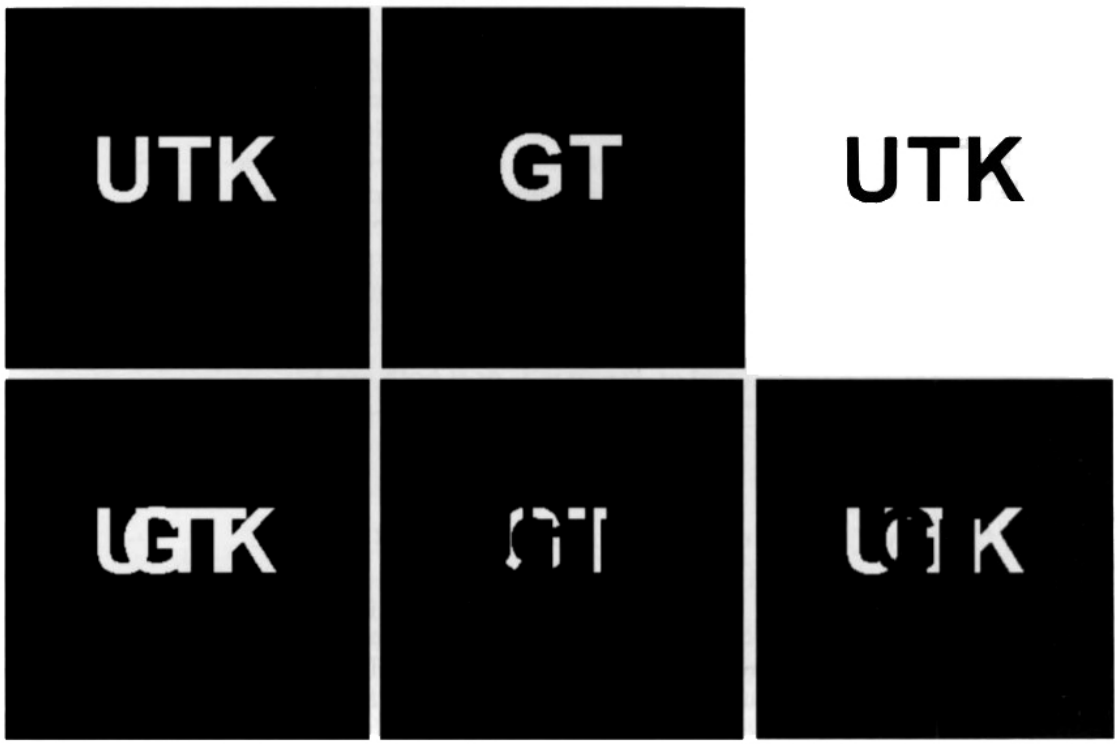
en este caso, como se ha mencionado anteriormente en relación con el punto de vista de conjuntos, los elementos de $A$ y $B$ tiene valor (1). Así, vemos que el punto de vista de la función se ocupa de el primer plano (1) y el fondo (0) de forma simultánea. El punto de ajuste de vista se ocupa sólo del primer plano de píxeles, y se entiende que todos los píxeles que no son píxeles de primer plano constituyen el fondo. Por supuesto, los resultados utilizando cualquiera de los puntos de vista son los mismos. Las operaciones de conjuntos definidos en la Figura 2.1.1 se puede realizar en imágenes binarias usando MATLAB mediante operadores lógicos OR (|), AND (&), y NOT (∼), como se muestra en el Cuadro 2.1.

La Figura 2.1.8 muestra los resultados de la aplicación de varios operadores lógicos a dos imágenes binarias que contienen texto. La imagen en la Figura 2.1.8 (d) es la unión (equivalente a OR) de las imágenes "UTK" y "GT"; que contiene todos los píxeles de primer plano de los dos. En contraste, la intersección (equivalente a AND) de las dos imágenes Figura 2.1.8 (e) muestra los píxeles en donde las letras "UTK" y "GT" se superponen. Finalmente, la imagen diferencia (equivalente a NOT) de conjuntos Figura 2.1.8 (f) muestra las letras en "UTK" eliminando los píxeles de "GT".

Cuadro 2.1: Uso de expresiones lógicas en MATLAB para realizar operaciones de conjuntos de imágenes binarias.

| **Operación Conjunto** | **MATLAB** | **Nombre** |
|---|---|---|
| $A \cap B$ | $A$ & $B$ | AND |
| $A \cup B$ | $A \mid B$ | OR |
| $A^c$ | $\sim A$ | NOT |
| $A - B$ | $A$ & $\sim B$ | DIFFERENCE |

Figura 2.1.8: (a) Imagen binaria $A$. (b) Imagen binaria $B$. (c) Complemento $\sim A$ . (d) Unión $A \mid \sim B$. (e) Intersección $A$ & $B$. (f) Diferencia $A$ & $\sim B$.



# Capítulo 3

# MATLAB

## 3.1. MATLAB y el Procesamiento Digital de Imágenes

### 3.1.1. ¿Qué es MATLAB?

MATLAB (abreviatura de MATrix LABoratory, "laboratorio de matrices") es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Mac OS X.

Entre sus prestaciones básicas se encuentran la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

El paquete de software de MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones: Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolbox) que cubren actualmente la mayoría de áreas principales del mundo de la ingeniería y la simulación, como por ejemplo

la toolbox de procesamiento digital de imágenes (IPT), la toolbox de procesamiento de señal, la toolbox de comunicaciones y así hasta más de 150 herramientas.

MATLAB proporciona un entorno de trabajo interactivo cuyo elemento básico de trabajo son las matrices, lo que permite la resolución numérica de problemas en un tiempo mucho menor que si se utilizan lenguajes de programación tradicionales como pueden ser los lenguajes Fortran, Basic o C, con la ventaja de que su lenguaje propio de programación es similar al de los lenguajes tradicionales. Al trabajar con matrices se pueden describir infinidad de variables de una forma altamente flexible y matemáticamente eficiente. Por ejemplo, una imagen se puede escribir como una matriz de píxeles, un sonido como una matriz de fluctuaciones, y en general se puede describir con una matriz cualquier relación lineal entre las componentes de un modelo matemático.

MATLAB empezó a utilizarse por investigadores y profesionales de Ingeniería de Control, expandiéndose rápidamente a diferentes disciplinas. Actualmente es un programa muy utilizado en educación, particularmente en la enseñanza del álgebra lineal y el análisis numérico, y es muy popular entre los científicos trabajando en procesamiento digital de imágenes. Los usuarios de MATLAB proceden de varios campos de la ingeniería, ciencia y economía, y pese a ser un paquete de software profesional y de precio elevado, ha conseguido instaurarse rápidamente en las aulas y despachos de muchos centros educativos de todo el mundo. [12]

## 3.2. Image Processing Toolbox

Para capturar imágenes con Matlab es indispensable utilizar *Image Processing Toolbox*, caja de herramienta que nos ofrece un conjunto completo de algoritmos estándar, funciones y aplicaciones para el procesamiento de imágenes, análisis, visualización y desarrollo de algoritmos. Puede realizar análisis, segmentación, mejora, reducción de ruido, transformaciones geométricas, y registro de imágenes.

Image Processing Toolbox admite un conjunto diverso de tipos de imágenes , incluyendo alto rango dinámico, resolución gigapixel, perfil ICC incrustado, y tomográfico. Funciones de visualización y aplicaciones que te permiten explorar las imágenes y vídeos, examinar una región de píxeles, ajustar el color y el contraste, crear contornos o histogramas y manipular las regiones de interés. La caja de herramientas es compatible con los flujos de trabajo para el procesamiento, visualización, y la navegación de grandes imágenes.

### 3.2.1. Funciones

Este Toolbox proporciona a MATLAB de un conjunto de funciones que extienden las capacidades del entorno de computación numérica para realizar desarrollo de aplicaciones y de nuevos algoritmos en el campo del procesamiento y análisis de imágenes. Este entorno matemático y de creación es ideal para dicho propósito ya que las imágenes se pueden representar mediante matrices.

La caja de herramientas es compatible con una amplia gama de operaciones de procesamiento de imágenes, algunas de las funciones más importantes incluidas dentro de este toolbox son las siguientes: [13]

- Operaciones geométricas
- Operaciones morfológicas.
- Transformación de imágenes.
- Diseño de filtros.
- Mejora y retoque de imágenes.
- Análisis y estadística de imágenes.
- Definición de mapas de colores.
- Proceso de bloques.

# Capítulo 4

# Procesamiento Digital de Imágenes

El procesamiento digital de imágenes (PDI) es entendiendo como el proceso de almacenamiento, transmisión y representación de información de imágenes digitales por medio de una computadora.

El término imagen se refiere a una función bidimensional de intensidad de luz $f(x, y)$ donde $x$ y $y$ denotan las coordenadas espaciales y el valor de $f$ en cualquier punto $(x, y)$ es proporcional a la intensidad de la imagen en ese punto. Una imagen digital puede escribirse como una matriz cuyos índices de fila y columna identifican un punto en la imagen y cuyo valor coincide con el nivel de intensidad de luz en ese punto. Cada elemento del array se corresponde con un elemento en la imagen y se le denomina pixel.

El interés en el procesamiento digital de imágenes se basa esencialmente en dos aspectos: la mejora de la información contenida en una imagen para la interpretación humana y el tratamiento de los datos de una escena para la favorecer la percepción autónoma por parte de una máquina.

Debido al amplio rango de tipos de imágenes empleadas en el PDI, no existe un límite claro respecto dónde se encuentra la línea divisoria entre el PDI y otras áreas afines, como el análisis de imágenes o la visión por computador, entre otras. El análisis de imágenes se refiere al proceso por el cual se extrae información cuantitativa de la

imagen y en donde el resultado del análisis es siempre una tabla de datos, una gráfica o cualquier representación de los datos numéricos.

El procesamiento de imágenes, sin embargo, siempre produce otra imagen como resultado de la operación, por lo que por lo general se pretende mejorar la calidad de una imagen para poder apreciar mejor determinados detalles. La visión por computador o visión artificial es un subcampo de la inteligencia artificial cuyo propósito es programar un computador para que .[en]tiendaüna escena o las características de una imagen, simulando la visión humana.

Atendiendo a los tipos de procesos implicados en estas disciplinas, se suele hacer una clasificación de tres niveles:

**Nivel bajo:** (procesamiento)

**Nivel medio:** (análisis)

**Nivel alto:** (interpretación)

Los procesos de nivel bajo incluyen la reducción de ruido, la mejora del contraste, y en general, la mejora de características de la imagen, en donde todas las entradas/salidas son imágenes. Los procesos de nivel medio analizan los niveles bajos e incluyen la segmentación (regiones, objetos), descripción de objetos, clasificación, etc. La entrada es una imagen y la salida son atributos de los objetos (bordes, contornos, identidades de objetos individuales). Por último, los procesos de nivel alto están generalmente orientados al proceso de interpretación de los elementos obtenidos en los niveles inferiores, entrando en juego el entendimiento y la toma de decisiones en función del contenido observado. Con esta clasificación, se le llama procesamiento digital de imágenes a los procesos cuyas entradas y salidas son imágenes (procesos de bajo nivel) y, además, a aquellos procesos que extraen atributos de imágenes, incluyendo el reconocimiento de objetos individuales (procesos de nivel medio). [12]

## 4.1. Clases de Procesamiento

Conviene clasificar los diferentes procesos involucrados en el procesamiento digital de imágenes para tener una visión general más estructurada. Podemos englobar la mayor parte de tareas en tres categorías, cada una de ellas con diferentes algoritmos involucrados:

- Mejora o realce de la imagen: procesamiento de la imagen para que el resultado sea más apropiado para una aplicación en particular. Tareas habituales: mejora de la nitidez o aclarado de imágenes desenfocadas, eliminación del ruido, mejora del contraste, mejora del brillo, detección de bordes, etc.

- Restauración de la imagen: se puede considerar como revertir el daño ocasionado a la imagen por una causa conocida. Tareas habituales: eliminar el desenfoque por movimiento, eliminar distorsiones ópticas, eliminar interferencia periódica, etc.

- Segmentación de la imagen: subdivide la imagen en partes o aísla ciertos objetos de una imagen. Tareas habituales: búsqueda y selección de formas determinadas en la imagen, máscaras de la imagen, etc.

## 4.2. Etapas del Procesamiento

La complejidad de la tarea determinará el número de procesos necesarios para resolver el problema, pero los pasos fundamentales y más habituales en el procesado digital de imágenes son los siguientes:

1. Adquisición de la imagen: Captura de la imagen.

2. Procesado del color: conversión de modelos de color para favorecer el procesamiento.

3. Operaciones morfológicas: herramientas para extraer componentes de la imagen útiles para la representación y descripción de formas. Procesos de manipulación de la imagen para lograr un resultado más adecuado que el original para una

aplicación específica (obtener detalles que no se veían, o simplemente destacar ciertas características de interés).

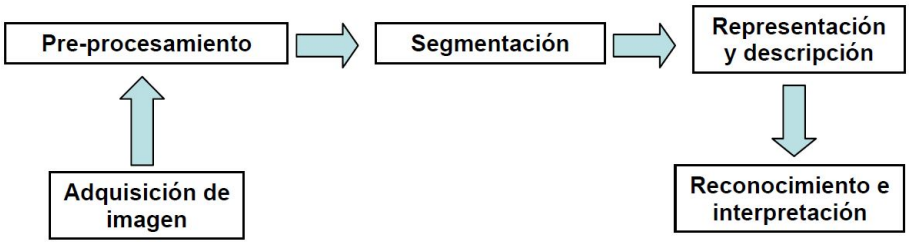
4. Segmentación: divide una imagen en sus partes constituyentes. Los objetos se extraen o aíslan del resto de la imagen para su posterior análisis. Es una de las tareas más difíciles del procesado digital de imágenes.

5. Representación y descripción: casi siempre recibe una imagen segmentada que consta solamente de fronteras o de regiones. Se toman decisiones tales como si la forma obtenida debe ser tratada como un frontera o una región, y se extraen atributos que resultan en información cuantitativa de interés o que son básicos para diferenciar clases de objetos.

6. Reconocimiento: el proceso que asigna una etiqueta a un objeto basándose en sus descriptores o bien le da un significado a un grupo de objetos ya reconocidos.

Para el procesamiento de la imagen se han seguido los pasos reflejados en el siguiente esquema:

Figura 4.2.1: Etapas del procesamiento digital de imágenes




## 4.3. Modelos de Color

Un modelo de color establece un conjunto de colores primarios a partir de los que, mediante mezclas, se pueden obtener otros colores hasta cubrir todo el espectro visible, además del propio blanco, negro y grises, y aún más. Por ejemplo, hay colores, como el marrón o el magenta, que no están presentes en el espectro visible, y es nuestro cerebro

el que lo interpreta a partir de la combinación de ondas con diferentes longitudes. Existen dos tipos de modelos de color, los aditivos y los sustractivos. Un modelo aditivo se basa en la adición o mezcla de los colores básicos como forma para obtener el blanco.

Un modelo sustractivo se basa en la mezcla de los colores primarios de dicho modelo para "sustraer la luz", es decir, para obtener el negro, es la ausencia de luz.

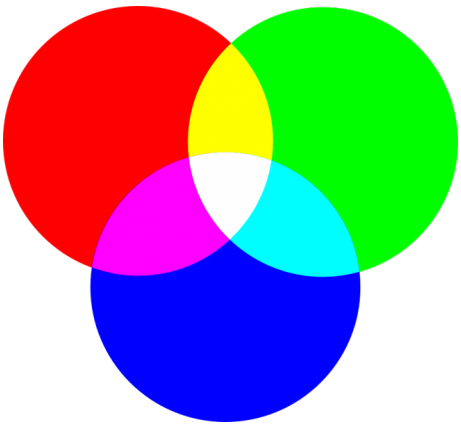
Cuando empleamos el término çolor.[en] realidad nos referimos al matiz o croma. Y junto a los colores también tenemos los tres casos especiales: el blanco, el negro y los grises. Los modelos de color más comunes son RGB (utilizado en monitores), CMYK (utilizado para impresión) y HSV (utilizado en procesamiento de imagenes). [14]

### 4.3.1. Modelo HSV vs RGB

La principal diferencia entre ambos es que el modelo HSV separa el luma, o la intensidad de la imagen, del croma o información del color. En la visión por computador esto nos permite aumentar la robustez del programa a los cambios de iluminación o eliminación de sombras, entre otros.

### 4.3.2. Modelo de Color RGB

Figura 4.3.1: Modelo RGB



El modelo RGB (Red, Green, Blue; Rojo, Verde, Azul) es el formado por la composición de los colores primarios de la luz. Con este modelo es posible representar un color mediante: la combinación de los tres genera blanco, la ausencia de los tres genera negro y las diferentes mezclas entre ellos representarían toda la gama de color. Un inconveniente que presenta es que no define lo que es exactamente con presicion un color (es facil confundir en amarillo con el naranja), por lo que los mismos valores RGB pueden mostrar diferentes colores en diferentes dispositivos, a pesar de usar este mismo modelo, ya que varían sus espacios de color. [15]

### 4.3.3. Modelo de Color HSV

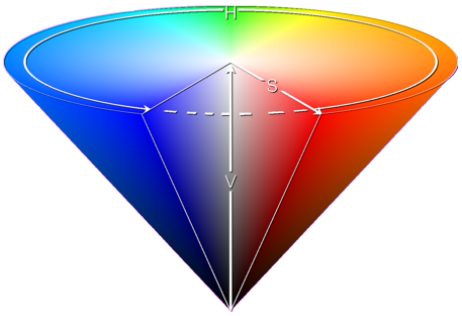
El espacio HSV representa uno de los espacios de coordenadas más clásicos e intuitivos existentes en la literatura. Su interpretación geométrica viene determinada por un cono de base quasi-hexagonal. Con esta representación del espacio de color, cada color trabaja con 3 componentes básicas: matiz(tono), saturación y brillo. El matiz, hHSV, hace referencia al valor de cromaticidad o clase de color. La saturación, sHSV , se refiere a las longitudes de onda que se suman a la frecuencia del color, y determina la cantidad de blanco que contiene un color.

Mientras menos saturado este un color, mayor es la cantidad de blanco, y mientras más saturado este un color, menor es la cantidad de blanco. En definitiva, la saturación representa la pureza e intensidad de un color. Así, la falta de saturación viene dada por la generatriz en la representación del cono HSV. Esa falta de saturación representa la gama de grises desde el blanco hasta el negro. La luminancia, vHSV, se corresponde con la apreciación subjetiva de claridad y oscuridad.

HSV (matiz-saturación-valor) y HSL (matiz-saturación-luminosidad) son las dos más comunes conversiones y representaciones de un modelo de color RGB. Desarrollado en la década de 1970 para los gráficos de aplicaciones de computadora, HSL y HSV se utilizan hoy en día en los selectores de color, en la edición de imágenes de software, y con frecuencia en el análisis de imágenes y visión por computador.

Normalmente este modelo es presentado en forma de cono o ruleta de color. El ángulo alrededor del eje vertical central corresponde a "matiz", y la distancia desde el eje corresponde a "saturación". Estos dos primeros valores dan los dos esquemas de la 'H' y 'S' en sus nombres. La altura corresponde a un tercer valor 'V' la representación del sistema de la percepción de luminancia en relación a la saturación.

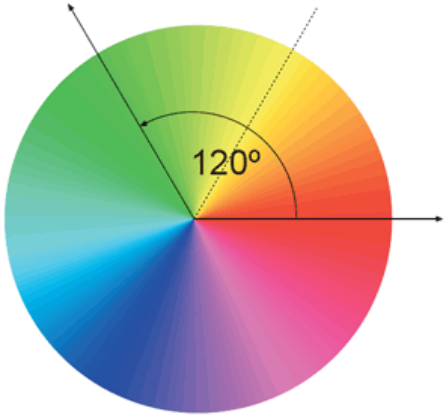
Figura 4.3.2: Modelo HSV




#### 4.3.3.1. Matiz

Se representa como un grado de angulo cuyos calores posibles van de 0 a 360° (aunque para algunas se normalizan del 0 al 100). Cada valor corresponde a un color. Ejemplos: 0 es rojo, 60 es amarillo y 120 verde

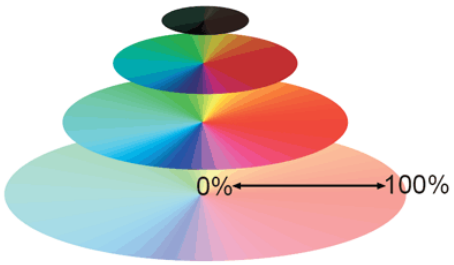
Figura 4.3.3: Hue



#### 4.3.3.2. Saturación

Se representa como la distancia al centro o eje de brillo negro-blanco. Se mide en porcentaje, siendo sus valores posibles entre 0 y 100 (De menos a más cantidad de color). A mayor saturación de un color mayor intensidad presentará su tonalidad.

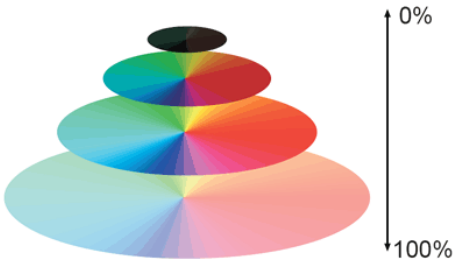
Figura 4.3.4: Saturation




#### 4.3.3.3. Valor

Representa la altura en el eje blanco-negro. Sus valores posibles varían del 0 al 100, siendo 0 siempre negro y 100 blanco o un color más o menos saturado (De totalmente oscuro a la máxima luminosidad).

Figura 4.3.5: Value

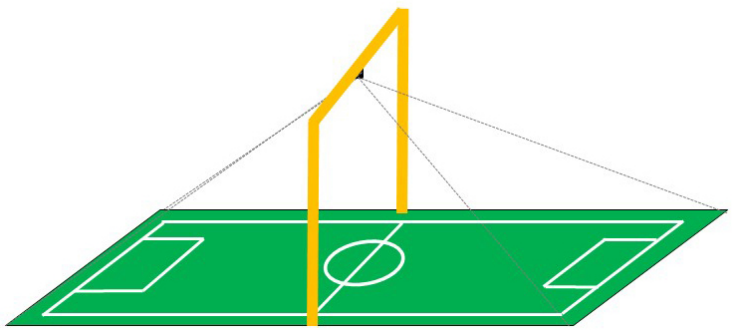


# Capítulo 5

# Desarrollo

Para la implementación y análisis del algoritmo desarrollado para la obtención de la posición y orientación del robot, se realiza un prototipo del entorno, utilizando un soporte para fijar la cámara a una distancia determinada del suelo, cubriendo de esta forma toda el área de la cancha. Posteriormente se realizan una serie de pruebas que evalúan el desempeño del algoritmo y las cuales se explicaran paso a paso a lo largo de este capitulo.

Figura 5.0.1: Prototipo del modelo

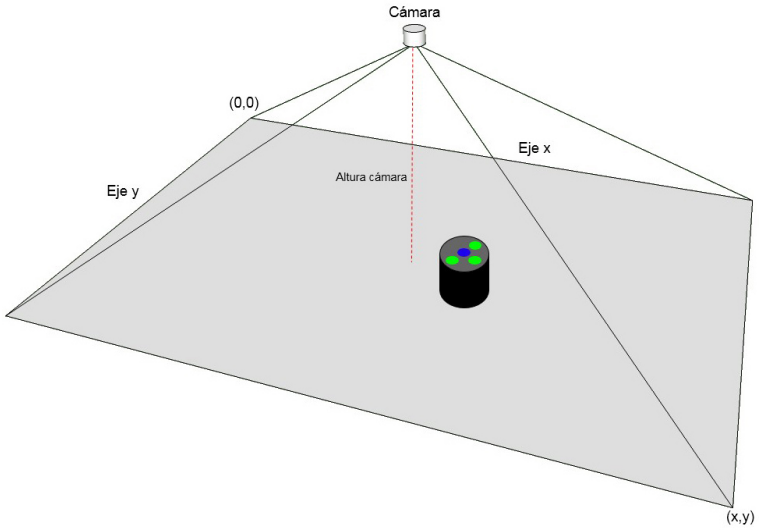


## 5.1. Adquisición de Imágenes

El desarrollo del algoritmo para la estimación de la posición y orientación se realiza utilizando una webcam que captura imágenes de una dimension (720x1280) píxeles en formato RGB. Como se observa en la Figura 5.1.1, el entorno de navegación es

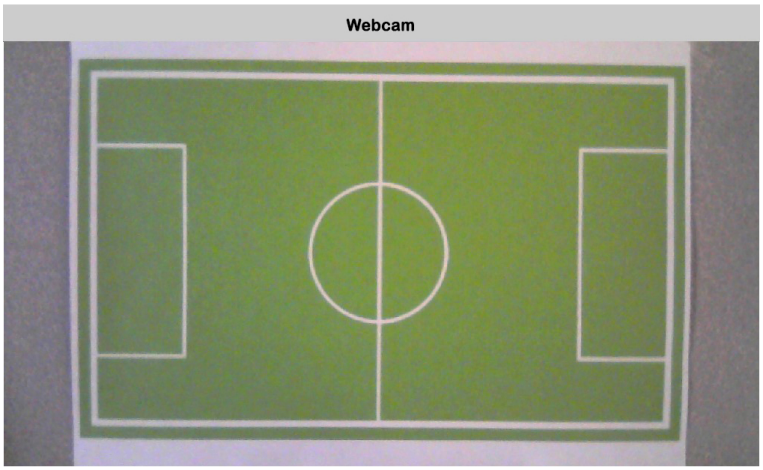
un área determinada por el alcance de visión de la cámara. La cámara es conectada mediante conexión USB a un computador, de esta forma es posible capturar la imagen del entorno en el que se encuentra el robot en tiempo real, para posteriormente realizar un tratamiento de ella en MATLAB para obtener la posición y orientación del robot.

Figura 5.1.1: Área de alcance de visión de la cámara




A continuación tenemos un ejemplo de una imagen obtenida en tiempo real realizada por la webcam.

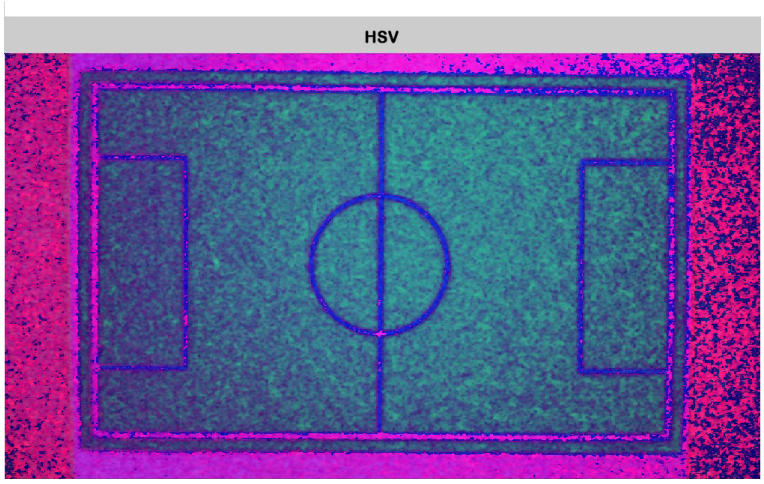
Figura 5.1.2: Adquisición de imagen en formato RGB



## 5.2. Conversión de Modelo de Color

Como se ha mencionado en el Capitulo 4, es necesario realizar la conversión del modelo de color RGB al modelo de color HSV. Lo anterior para aumentar la robustez del programa a los cambios de iluminación o eliminación de sombras, entre otros.

Figura 5.2.1: Conversión de modelo de color RGB a HSV




## 5.3. Extracción Binaria de Colores

A continuación se observan las gráficas de la extracción binaria de cada color del resultado de la conversión de modelo de color RGB a HSV.

Figura 5.3.1: Extracción binaria de color blanco.

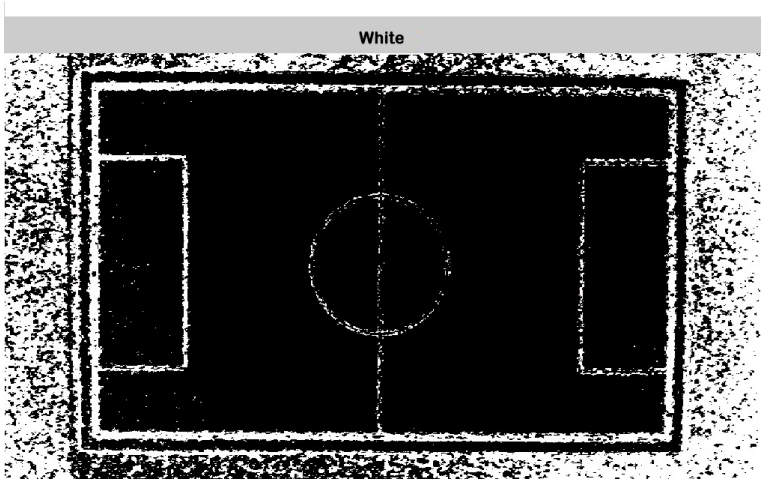


Figura 5.3.2: Extracción binaria de color verde.

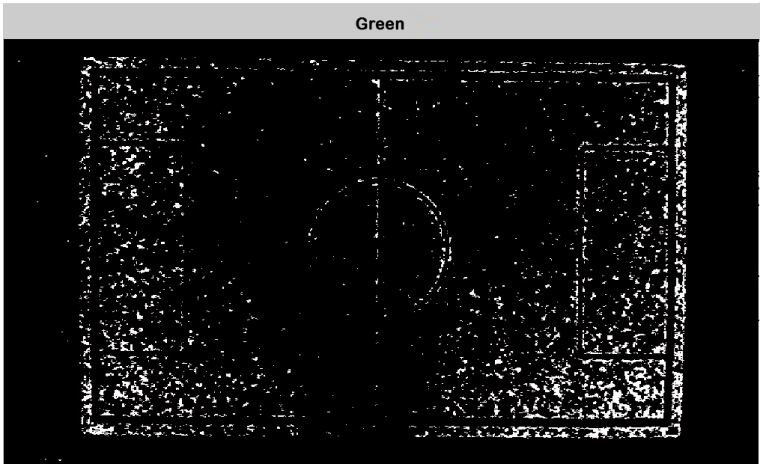
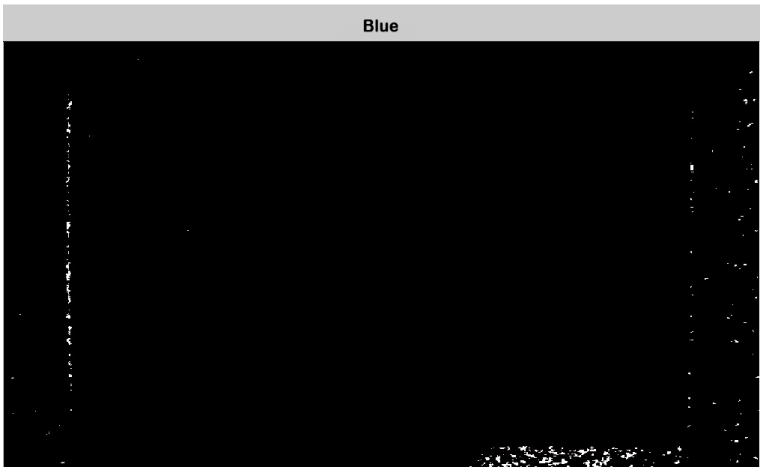



Figura 5.3.3: Extracción binaria de color azul.



## 5.4. Detección del Campo de Juego

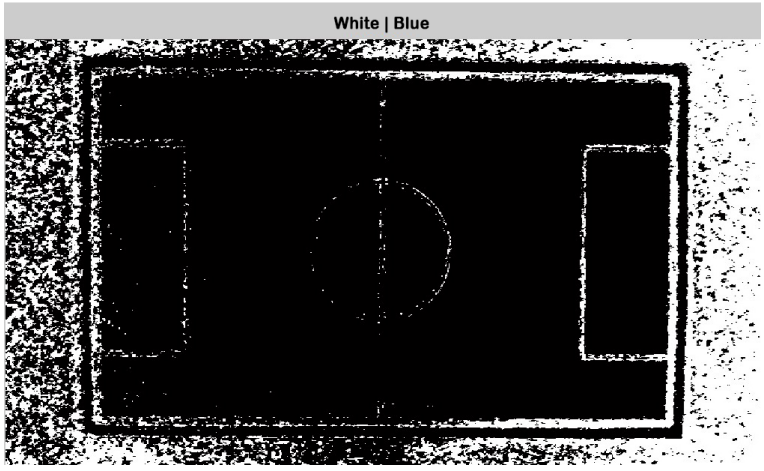
En esta sección se realizara la detección del campo de juego o segmentación de la imagen para extraer nuestra área de interés del resto de la imagen y cuya extracción se lleva a cabo para no perder procesamiento en áreas de no importancia.

Para determinar únicamente el área de la cancha, en donde el robot se desplazara, se realiza una captura de toda la cancha y se aplica una serie de filtros. A partir del área delimitada por las líneas de campo se crea una nueva ventana que representa el área total por donde el robot puede desplazarse.

### 5.4.1. Aplicación de filtros morfológicos

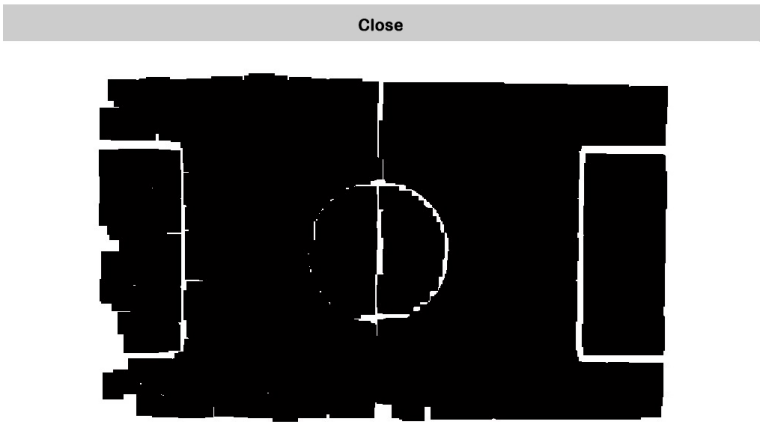
En la siguiente imagen observamos la aplicación del operador lógico OR (|), entre la extracción de la matriz binaria de color blanco y azul, o lo que en teoría de conjuntos se conoce como la unión entre dos conjuntos $A \cup B$ esto para mejorar la detección del área que comprende el campo de juego.

Figura 5.4.1: Aplicación de operadores lógicos.



En la siguiente Figura 5.4.2 se evidencia la aplicación del filtro morfológico close o de cierre sobre la Figura 5.4.1, el cual es una dilatación seguida de una erosión, este filtro se aplica con la finalidad de limpiar o borrar aquellas áreas de color negro (0 binario) de menor o igual área que el elemento estructurante.

Figura 5.4.2: Aplicación de filtro morfológico close.

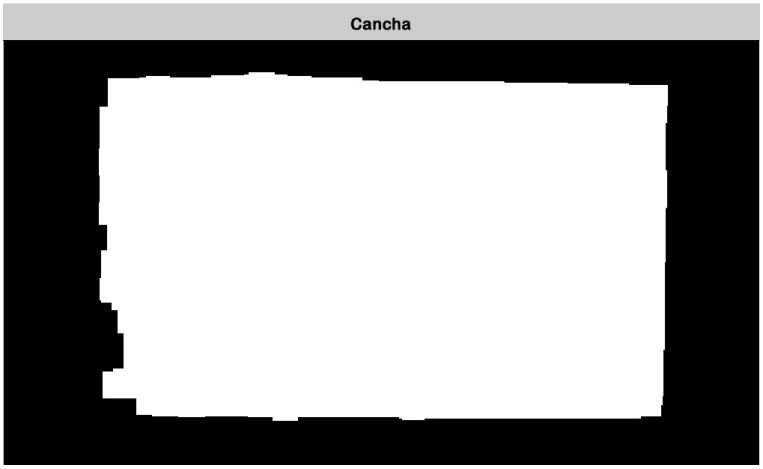



Seguido al filtro close, se aplica el filtro morfologico open o de apertura sobre la Figura 5.4.2, el cual es una erosion seguida de una dilatacion, este filtro se aplica con la finalidad de unir aquellas áreas de color negro (0 binario) de menor o igual área que el elemento estructurante.

Figura 5.4.3: Aplicación de filtro morfológico open.

La detección del área de interés en una imagen binaria se realiza detectando el color blanco (1 binario), para lo cual en este caso es necesario aplicar un filtro de complemento a la imagen.

Figura 5.4.4: Filtro complemento.




Teniendo el complemento de la imagen binaria, procedemos a trazar un recuadro sobre el área de interés, el cual parte desde la coordenada $x_i$ que se traduce como la primera columna donde se detecto un (1) y en la coordenada $y_i$ como la primera fila donde se detecto un (1), finalizando en la coordenada $x_f$ que es la ultima columna donde se detecto un (1) y en la coordenada $y_f$, ultima fila donde se detecto un (1).

Figura 5.4.5: Detección del campo de juego en matriz binaria.

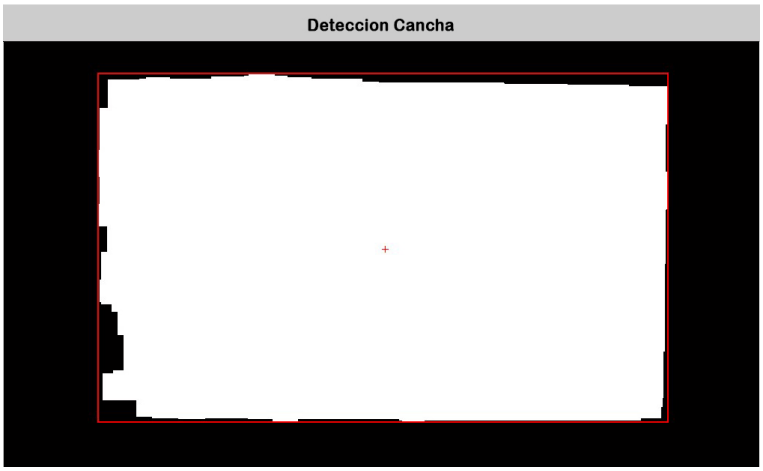
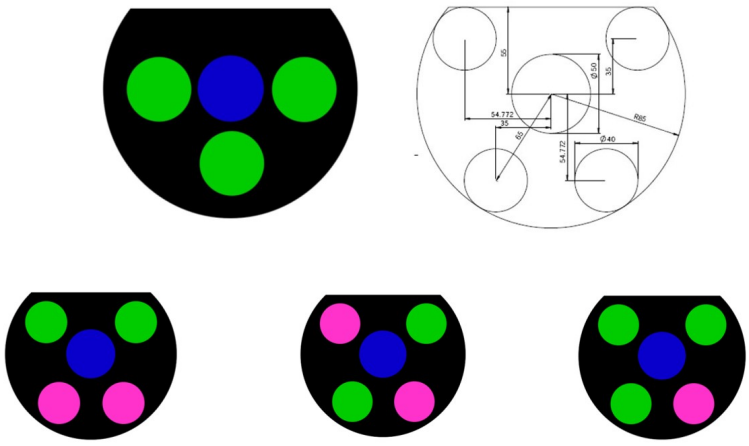


Figura 5.4.6: Detección del campo de juego sobre modelo RGB.


## 5.5. Reconocimiento del Patrón de Punto (SSL)

El cálculo de las coordenadas exactas en las que se encuentra el robot en un instante determinado, se realiza a partir del modelo cinemático de la parte superior del robot con respecto al espacio en el que se encuentra, utilizando un patrón de colores SSL, (Ver Fig. 5.5.1) claramente visible por la cámara desde la parte superior. Este patrón es comúnmente utilizado en la liga de tamaño pequeño en RoboCup, el campeonato de fútbol robótico (Ver Fig. 5.5.2) celebrado anualmente.

Figura 5.5.1: Patrón de punto SSL.



Cada robot compite en un juego de fútbol con reglas establecidas, los robots deben llevar este patrón de colores en la parte superior que los identifica.

Figura 5.5.2: Competición tamaño pequeño Robocup.

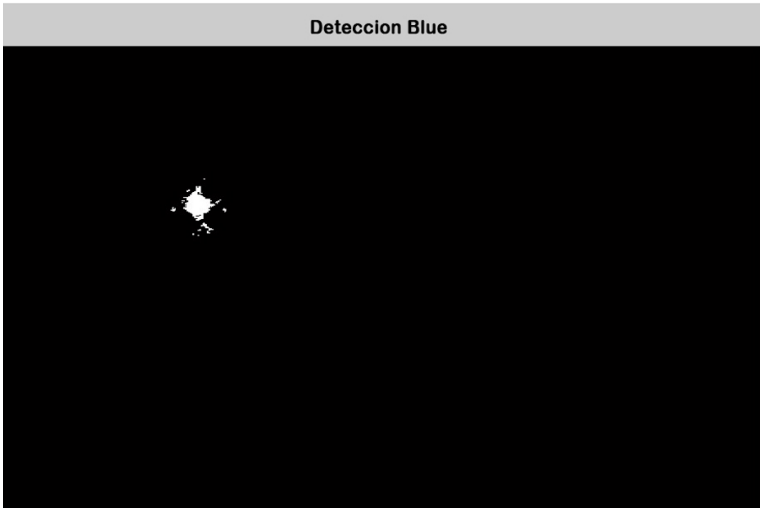
### 5.5.1. Obtención de posición del robot

Considerando un sistema de referencia global cartesiano bidimensional, el estado de la posición del robot dentro del entorno se determinado realizando la detección del centro del patrón SSL (color azul) dentro del entorno o área de interés anteriormente extraída:

Figura 5.5.3: Patrón de punto SSL en tiempo real.

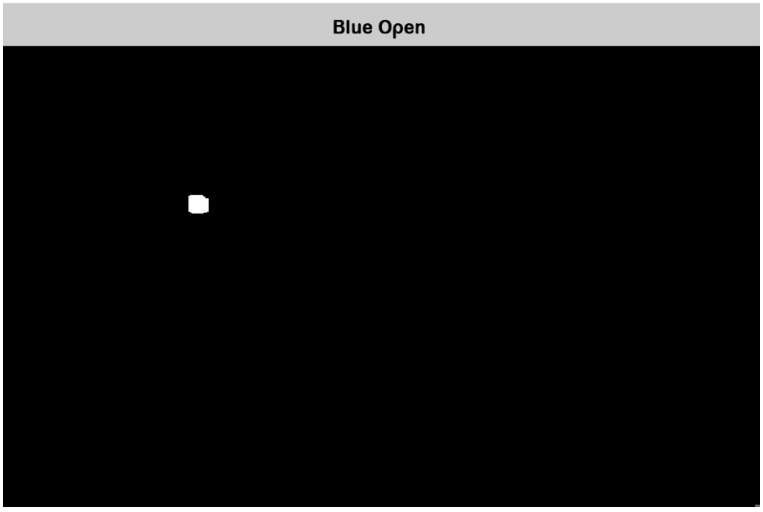
En la siguiente imagen se observa la extracción de la matriz binaria correspondiente al color azul de la conversión HSV, dicha matriz es del mismo tamaño que el área de interés o campo de juego.

Figura 5.5.4: Matriz binaria (HSV) color azul.



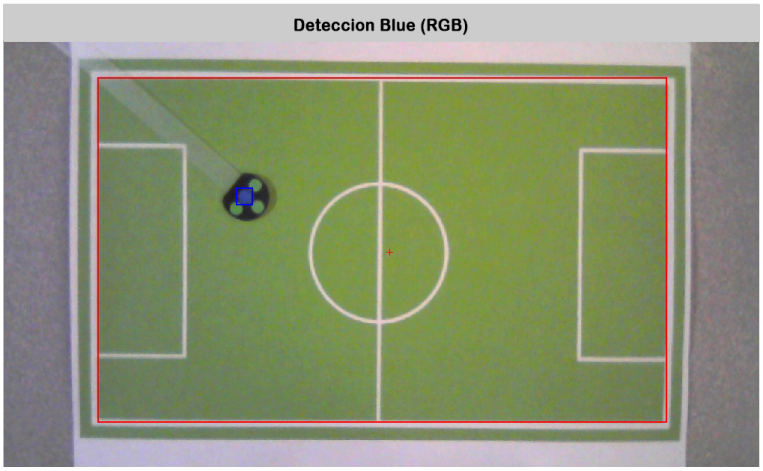

Como se puede apreciar es necesario realizar la aplicación de un filtro de apertura (Erosión, Dilatación) para mejorar la imagen y favorecer o facilitar la detección del punto azul del patrón SSL.

Figura 5.5.5: Filtro de apertura a matriz binaria del color azul.



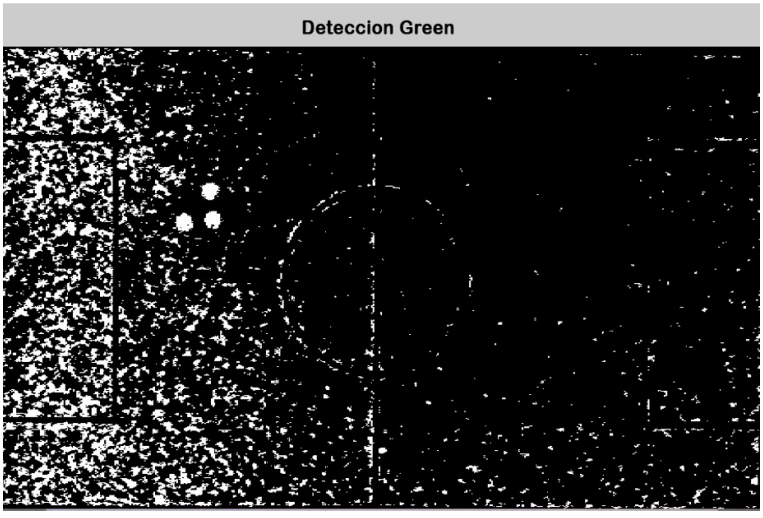
Con nuestra imagen ya filtrada, se procede a realizar la detección del color azul, región que es delimitada por un recuadro del mismo color, así como se puede observa en la siguiente Figura 5.5.6, donde el recuadro se encuentra ubicado o sobrepuesto sobre la imagen en tiempo real.

Figura 5.5.6: Detección del color azul en patrón SSL en tiempo real.



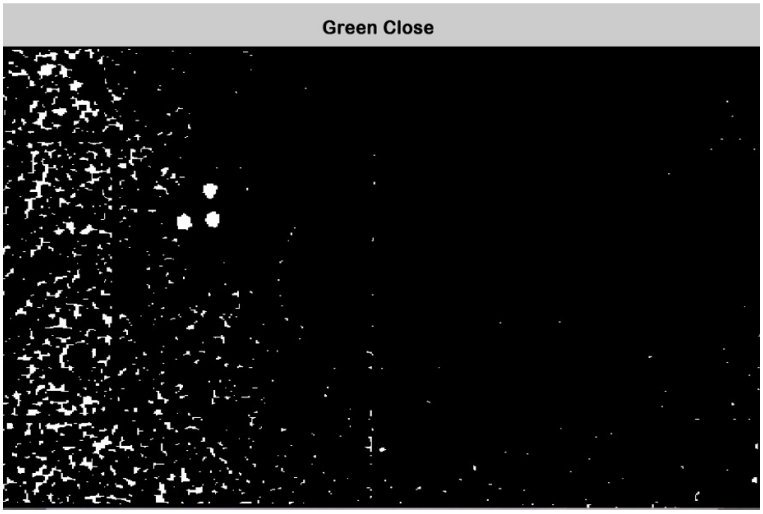

Al igual que en el caso anterior, para la detección de las regiones de color verde del patrón de punto SSL, es necesario realizar la extracción de la matriz binaria correspondiente al color verde de la conversión HSV.

Figura 5.5.7: Matriz binaria (HSV) color verde.



Es necesario la aplicación de un filtro de cierre (Dilatación, Erosión) para limpiar regiones de la imagen y favorecer o facilitar la detección de las regiones verdes del patrón SSL.

Figura 5.5.8: Filtro de cierre a matriz binaria del color verde.



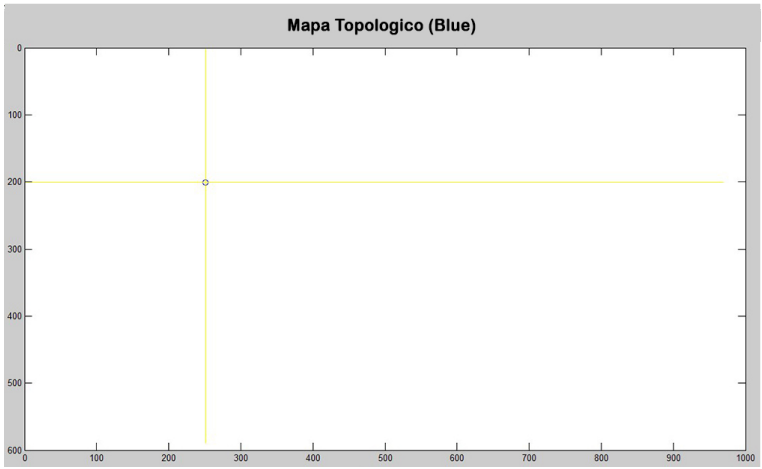

Posterior a la imagen filtrada, se puede realizar la detección del color verde sobre el patrón SSL, regiones que son delimitadas por recuadros del mismo color, así como se puede observar en la siguiente Figura, donde los recuadros se encuentran ubicados o sobrepuestos sobre la imagen en tiempo real.

Figura 5.5.9: Detección del color verde en patrón SSL en tiempo real.

## 5.6. Construcción de Mapa Topológico

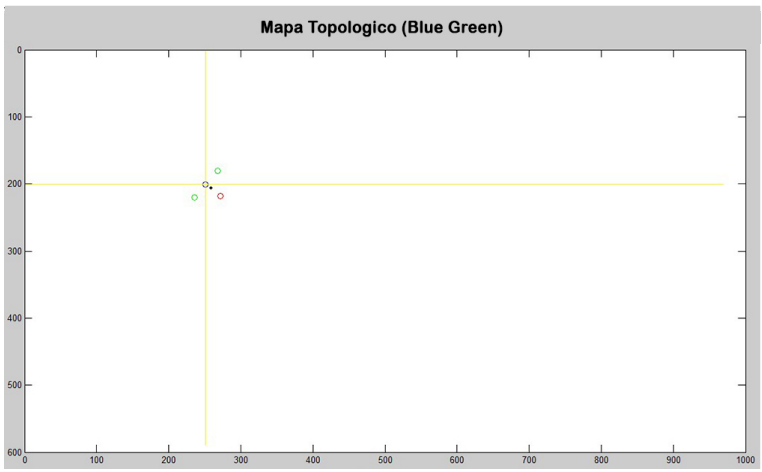
A partir de los datos y coordenadas obtenidos a través del procesamiento de imágenes se realiza la construcción de un mapa topológico que representa las características del entorno y la localización del robot móvil dentro de él. El primer paso es ubicar el centro del robot o el color azul del patrón de punto SSL.

Figura 5.6.1: Localización de patrón SSL (azul) en mapa topológico.



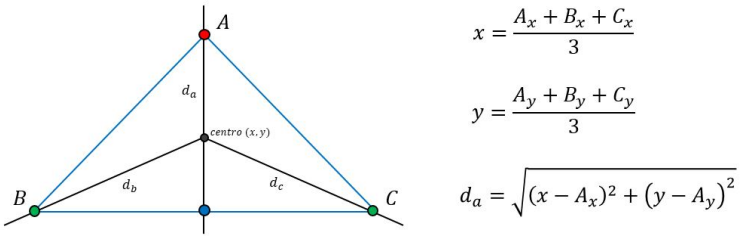

El segundo paso es ubicar las regiones de color verde del patrón de punto SSL. Con las cuales posteriormente se realizara la obtención de la orientación o dirección del robot.

Figura 5.6.2: Localización de patrón SSL (verde) en mapa topológico.



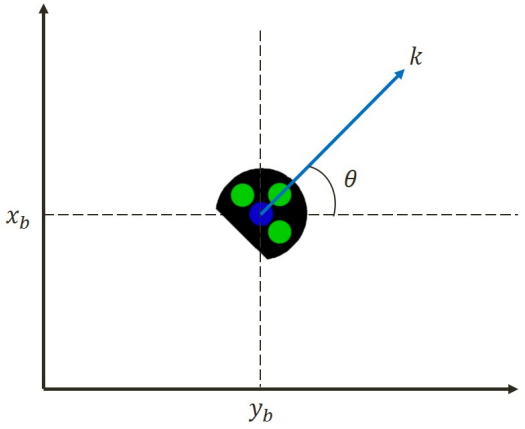
Calculando el centro del triangulo formado por las regiones de color verde del patrón SSL y hallando la distancia desde el centro de este triangulo a cada una de las regiones se logro identificar cual de estas regiones era la que indicaba la dirección del robot, cuya región es la de menor distancia al centro del triangulo.

Figura 5.6.3: Calculo de la dirección del robot.



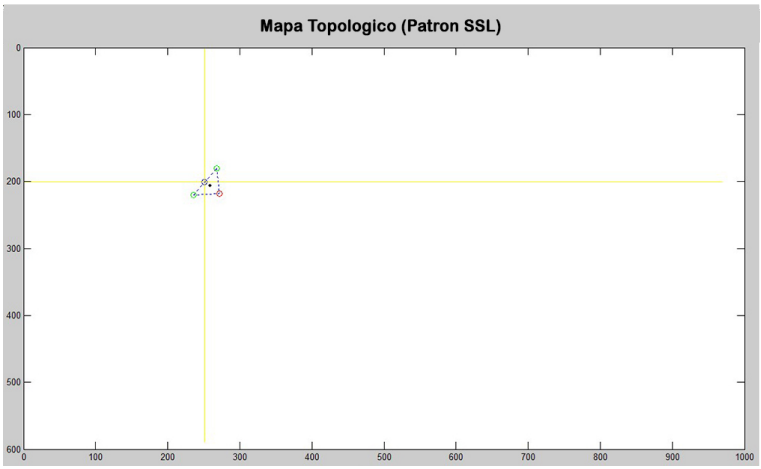

Obteniendo la región que indica la dirección del robot se calculo el ángulo que se formaba entre el eje $x$ de la región azul del patrón SSL y una linea que parte desde la coordenada del centro del patrón (azul) y cruza por el centro del triangulo formado por las demás regiones (verdes).

Figura 5.6.4: Calculo del angulo al cual se encuentra el robot.




La orientación del robot se ha definido como el ángulo $\theta$ formado por la recta $k$ con respecto al eje $x_b$ tomando como positivo el giro en el sentido contrario a las agujas del reloj. El origen del sistema de referencia global se debe elegir arbitrariamente en algún punto del espacio cartesiano.

Figura 5.6.5: Dirección del robot.




## 5.7. Planificación de Trayectorias

### 5.7.1. Definición de las coordenadas de los obstáculos y robot

La coordenada de la posición del robot se obtiene del reconocimiento del patrón SSL en tiempos real y almacenada en un vector, el robot puede moverse en cualquier dirección dentro del área definida de la cancha.

Figura 5.7.1: Definición de obstáculos y destino.

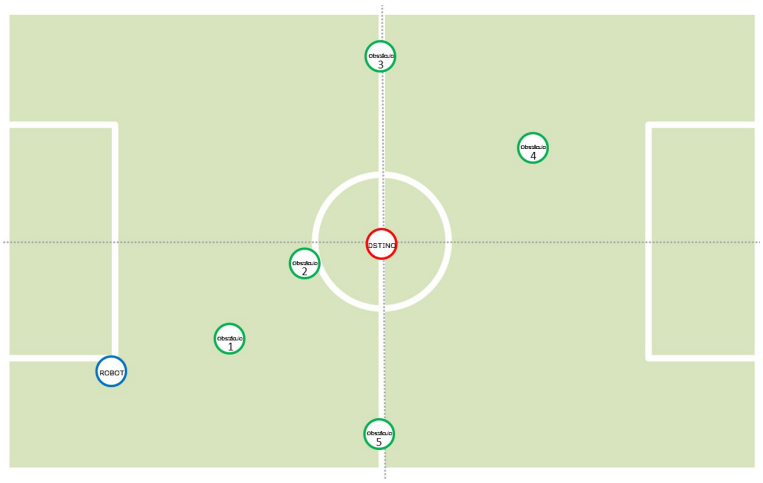
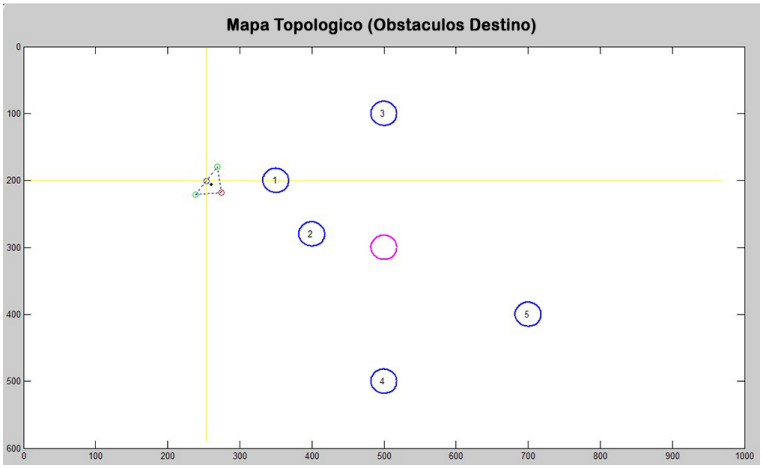


Figura 5.7.2: Definición de obstáculos y destino Mapa Topológico.

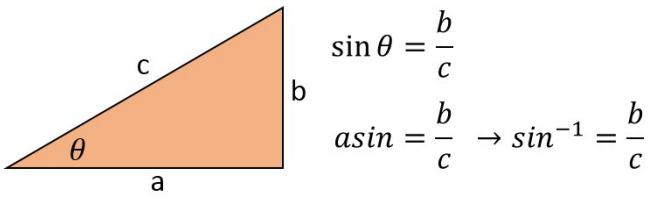



El algoritmo de navegación está dividido en dos partes, que denominamos algoritmo No. 1 y algoritmo No. 2. En cada instante de tiempo se calcula el ángulo que se forma entre el robot y el destino, este ángulo determina cuál de estos dos algoritmos se ejecuta. (ver Fig. 5.7.3).

Figura 5.7.3: Medición de ángulo entre Robot y Destino.

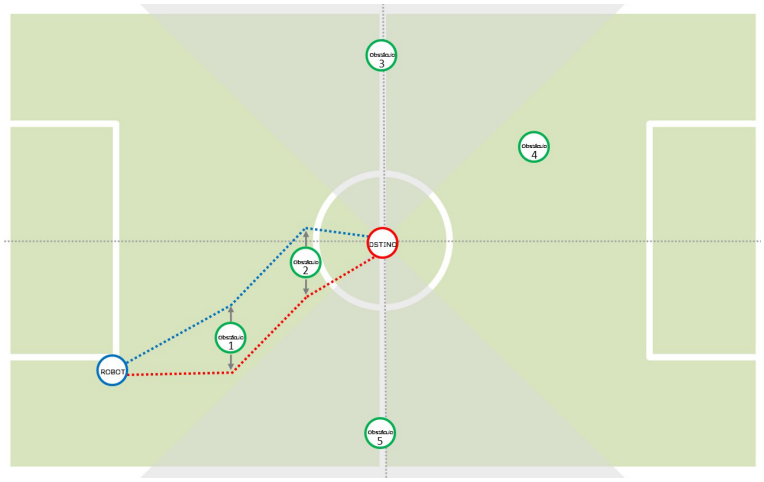

Para determinar el ángulo se utiliza la relación trigonométrica sobre el triángulo rectángulo que se forma entre el robot y el destino en cada uno de los cuatro cuadrantes que componen el área de la cancha por donde el robot puede moverse.

Figura 5.7.4: Relacion trigonometrica.



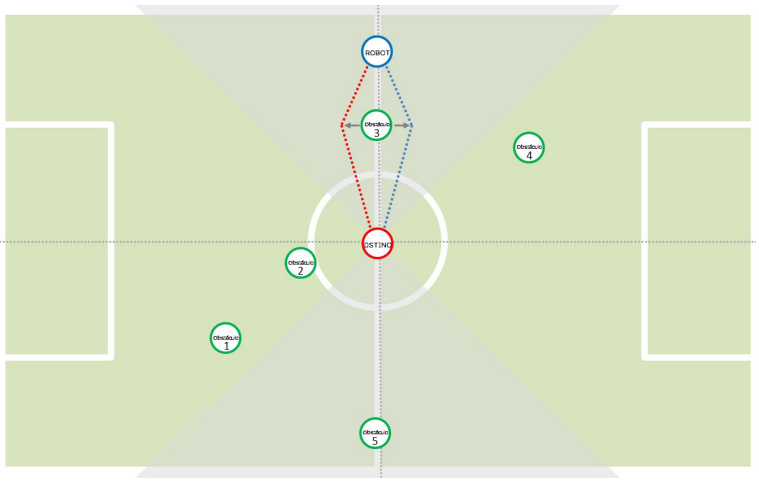

El algoritmo No. 1, realiza el proceso de elección de la ruta sobre cada uno de los obstáculos que se encuentran entre el robot y el destino. Como se aprecia en la Figura 5.7.5. El algoritmo No. 1 abre el espacio necesario para que el robot no choque con el obstáculo de forma vertical, este procedimiento funciona tanto de derecha a izquierda como izquierda a derecha.

Figura 5.7.5: Resultado del algoritmo No. 1.




El algoritmo No. 2, realiza el proceso de elección de la ruta sobre cada uno de los obstáculos que se encuentran entre el robot y el destino. Como se aprecia en la Figura 5.7.6. El algoritmo No. 2 abre el espacio necesario para que el robot no choque con el obstáculo de forma horizontal, este procedimiento funciona tanto de arriba hacia abajo como de abajo hacia arriba.

Figura 5.7.6: Resultado del algoritmo No. 2.




### 5.7.2. Obstáculos entre robot y destino

Después de definir los obstáculos y el destino, se traza un área determinada por la posición del robot y el destino. Los obstáculos que se encuentren dentro de esta área (ver Fig. 5.7.7), serán los que se tengan en cuenta para operar en el siguiente paso del algoritmo, que consiste en determinar la distancia del robot a cada uno de estos obstáculos. (ver Fig. 5.7.8)

Figura 5.7.7: Obstáculos dentro del área de visión del robot.

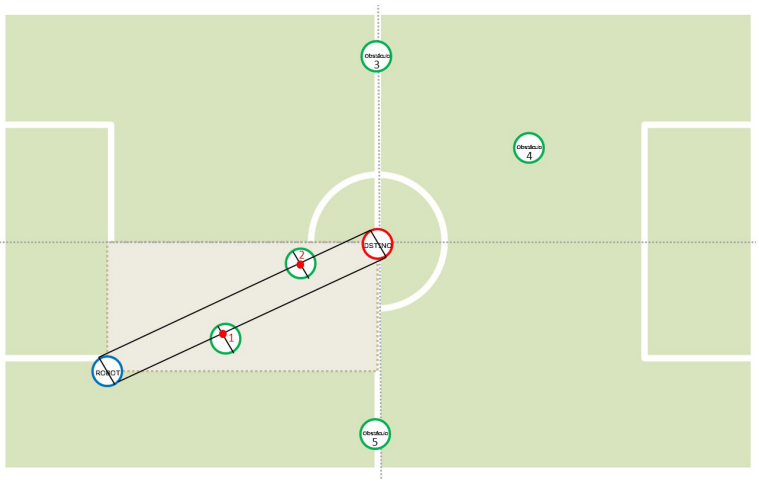
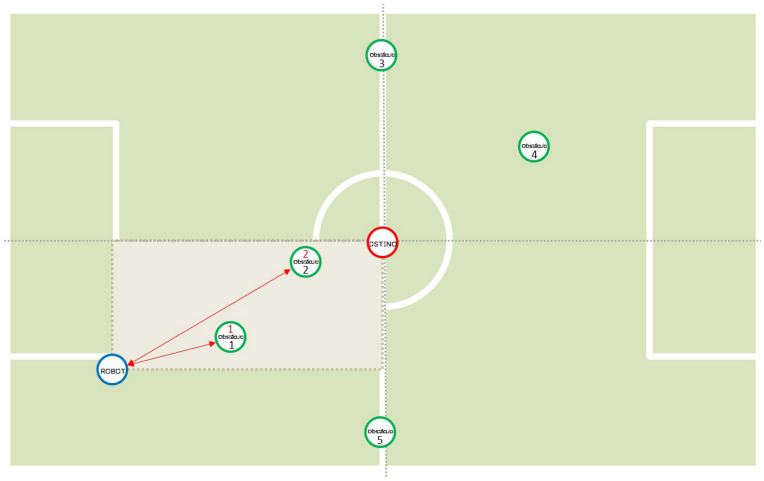


Figura 5.7.8: Distancia del Robot a cada uno de los obstáculos dentro del área de interés.




Para determinar los obstáculos que se encuentran dentro de la línea de visión del robot y el destino, se trazan dos rectas (desde el robot hasta el destino) y se determina la intersección con las rectas perpendiculares asociada a cada obstáculo. Una intersección indica que hay presente un obstáculo y habrá que realizar un procedimiento para calcular los dos puntos de apertura para que el robot no choque con el obstáculo.

Ecuación de la recta $A$ que pasa por dos puntos (recta entre robot y destino).

**Punto 1:** $(Rx1, Ry1)$

**Punto 2:** $(Dx2, Dy2)$

En base a estos dos puntos conocidos de la recta se calcula su ecuación:

$$\frac{(y - Ry_1)}{(1x - Rx_1)} = \frac{(Dy_2 - Ry_1)}{(Dx_2 - Rx_1)} \tag{5.7.1}$$

$$y - Ry_1 = \frac{(Dy_2 - Ry_1)}{(Dx_2 - Rx_1)}(x - Rx_1) \tag{5.7.2}$$

La ecuación se debe acercar a la ecuación general de la recta $y = mx + b$

donde, $m$ es la pendiente:

$$m = \frac{(Dy_2 - Ry_1)}{(Dx_2 - Rx_1)} \tag{5.7.3}$$

Ecuación de la recta B que pasa por dos puntos (recta asociada a cada obstáculo).

**Punto 1:** $(Ox_1, Oy_1)$

**Punto 2:** $(Ox_2, Oy_2)$

En base a estos dos puntos conocidos de la recta se calcula su ecuación:

$$\frac{(y - Oy_1)}{(x - Ox_1)} = \frac{(Oy_2 - Oy_1)}{(Ox_2 - Ox_1)} \tag{5.7.4}$$

$$y - Oy_1 = \frac{(Oy_2 - Oy_1)}{(Ox_2 - Ox_1)}(x - Ox_1) \tag{5.7.5}$$

La ecuación se debe acercar a la ecuación general de la recta $y = mx + b$

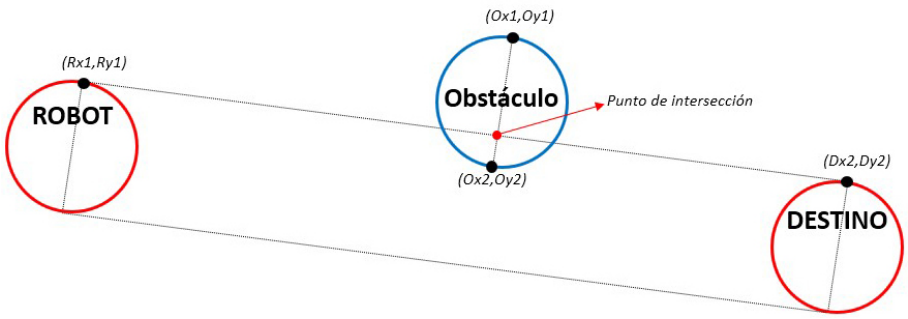
Donde, $m$ es la pendiente:

$$m = \frac{(Oy_2 - Oy_1)}{(Ox_2 - Ox_1)} \tag{5.7.6}$$

#### 5.7.2.1. Intersección entre dos rectas

Para determinar la intersección entre dos rectas, se igualan las dos ecuaciones de las rectas y se determinan la variable $x$ y la variable $y$. La coordenada $(x, y)$ es el punto en el que las dos rectas se interceptan.

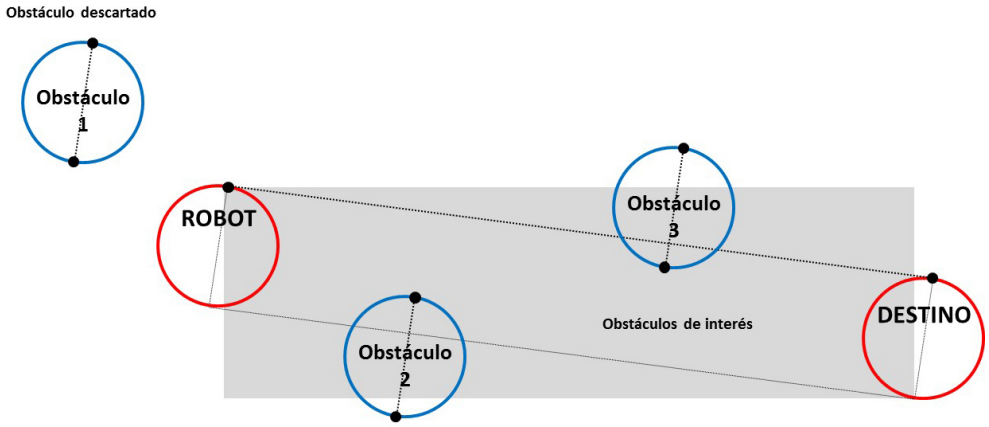
Figura 5.7.9: Detección de obstáculos entre robot y destino.




Luego de determinar la intersección de estas dos rectas, se mide la distancia del punto de intersección a cada uno de los obstáculos que se encuentran por delante del robot (obstáculos de interés), Sobre el primer obstáculo más cercano se calculan los

dos puntos de apertura (arriba y abajo) (ver Fig. 5.7.13 y Fig. 5.7.14 respectivamente), determinando una línea tangente a las dos circunferencias.

Figura 5.7.10: Obstáculos de interés entre robot y destino.




### 5.7.2.2. Intersección de una recta tangente entre dos circunferencias

Se resuelve el siguiente sistema:

$$C : x^2 + y^2 + Ax + By + C = 0 \tag{5.7.7}$$

$$s : y = mx + n \tag{5.7.8}$$

Sustituyendo $y = mx + n$ en la primera ecuación, resulta una ecuación de segundo grado en $x : ax^2 + bx + c = 0$, donde $a, b$ y $c$ son coeficientes conocidos. Esta ecuación podría tener dos soluciones, una o ninguna, según el valor del discriminante:

$$\triangle = b2 - 4ac \tag{5.7.9}$$

Figura 5.7.11: Tipo de rectas sobre circunferencias.

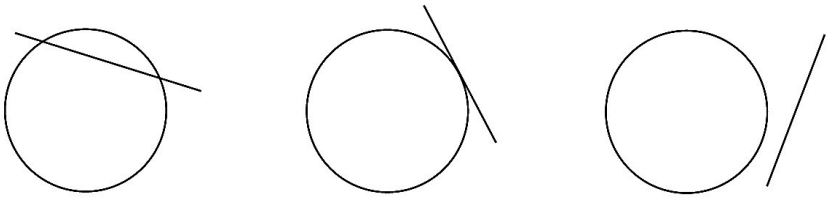


Figura 5.7.12: Punto de apertura sobre un obstáculo.

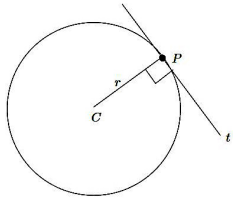



Figura 5.7.13: Punto de apertura superior al obstáculo, la línea verde indica la ruta que debe seguir el robot para llegar al destino.

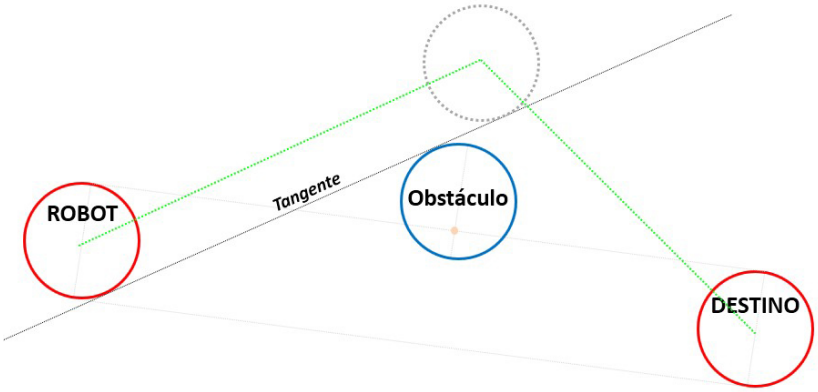
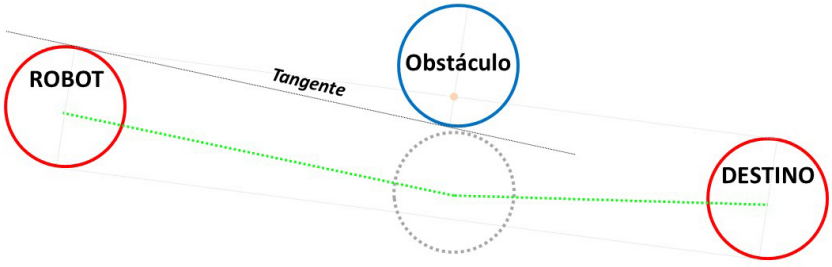



Figura 5.7.14: Punto de apertura inferior al obstáculo, la línea verde indica la ruta que debe seguir el robot para llegar al destino.




Teniendo las dos rutas, se calcula su longitud respectivamente y se dibuja la de menor distancia. Este proceso se aplica de nuevo tomando como punto inicial, la coordenada del punto de apertura seleccionado, lo que nos permite obtener la ruta desde el robot hasta el destino.

Figura 5.7.15: Obtención del espacio adecuado sobre el primer obstáculo a partir del método de la tangente entre dos circunferencias.

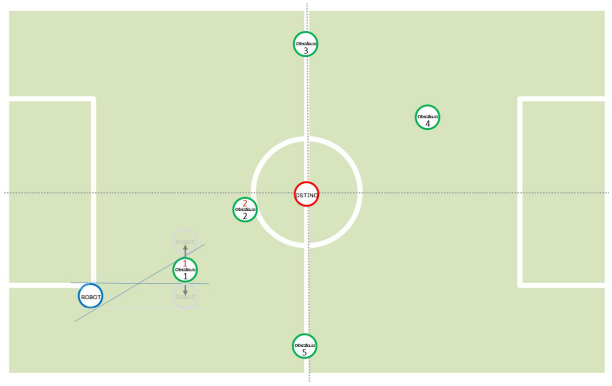


Figura 5.7.16: Trayectorias sobre el primer obstáculo.

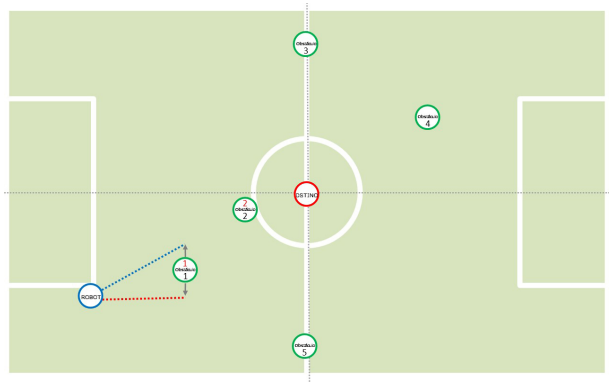


Figura 5.7.17: Trayectorias sobre el primer obstáculo Mapa Topológico.

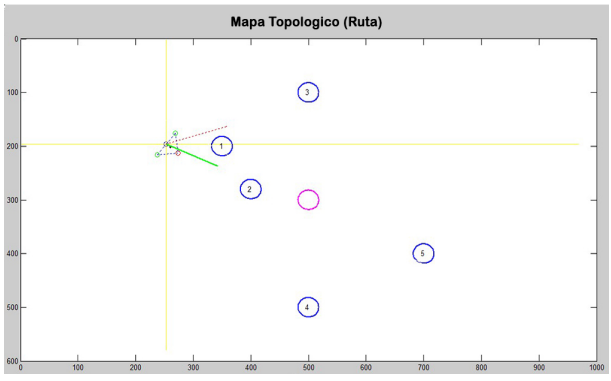


Figura 5.7.18: Obtención del espacio adecuado sobre el segundo obstáculo a partir del método de la tangente entre dos circunferencias.

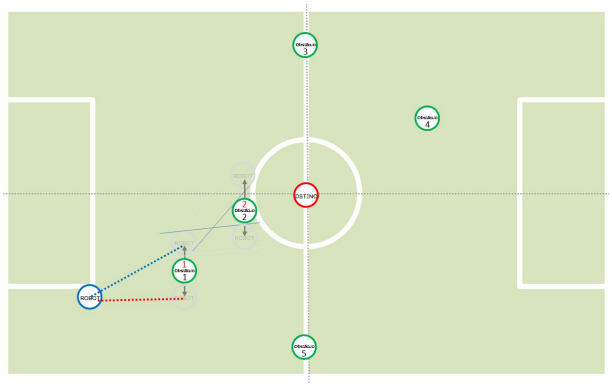


Figura 5.7.19: Trayectorias sobre el segundo obstáculo.

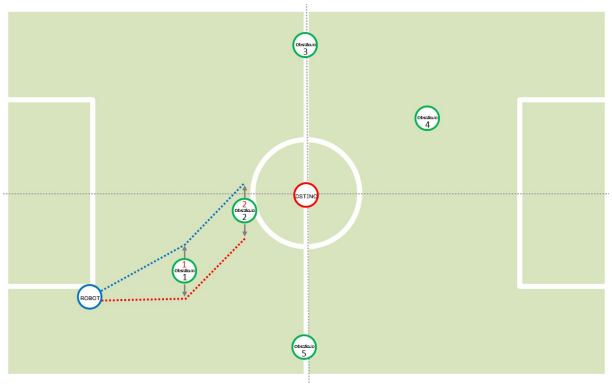


Figura 5.7.20: Trayectorias sobre el segundo obstáculo Mapa Topológico.

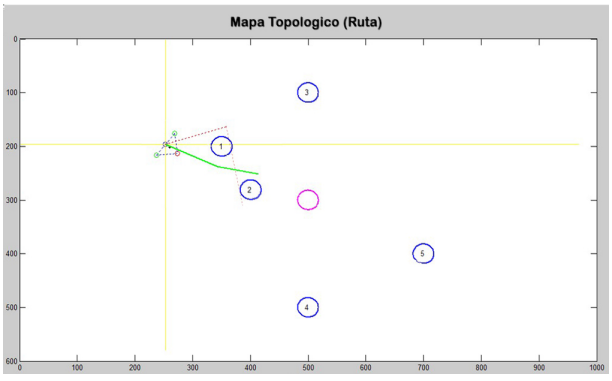


Figura 5.7.21: Trayectorias que puede seguir el robot para llegar al destino, esquivando los obstáculos.

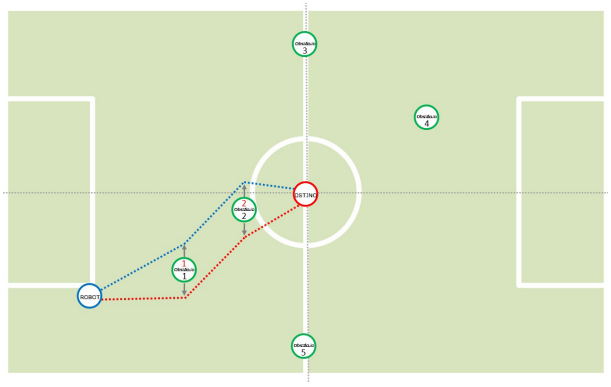



Figura 5.7.22: Trayectorias más corta que el robot puede seguir para llegar al destino.

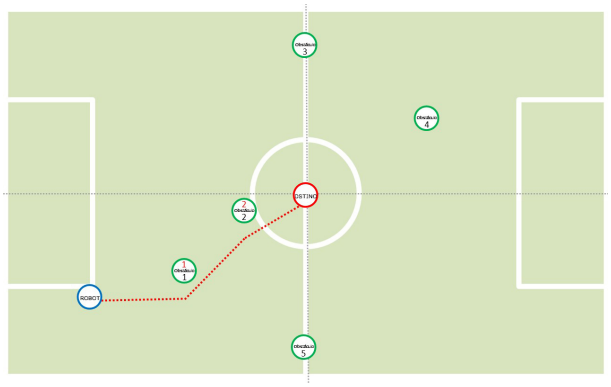



Figura 5.7.23: Trayectoria más corta sobre el segundo obstáculo Mapa Topológico.

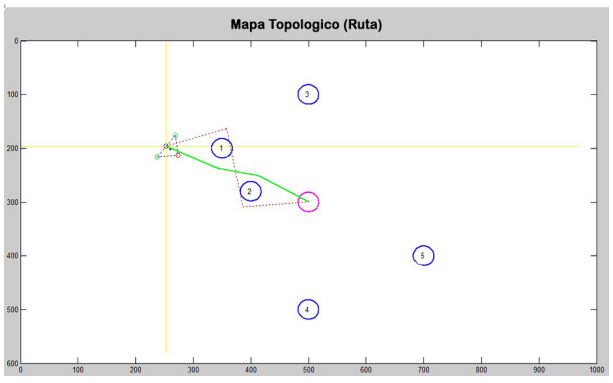


Figura 5.7.24: Trayectoria más corta sobre el segundo obstáculo Mapa Topológico.

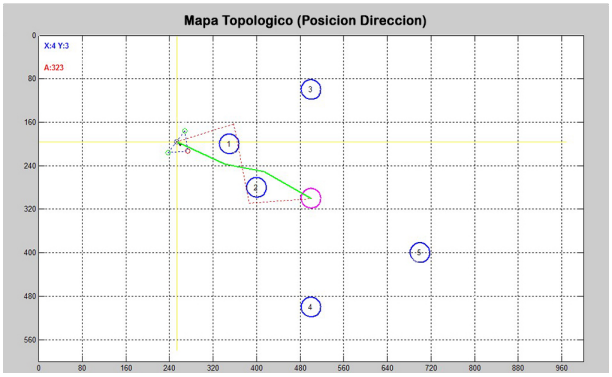
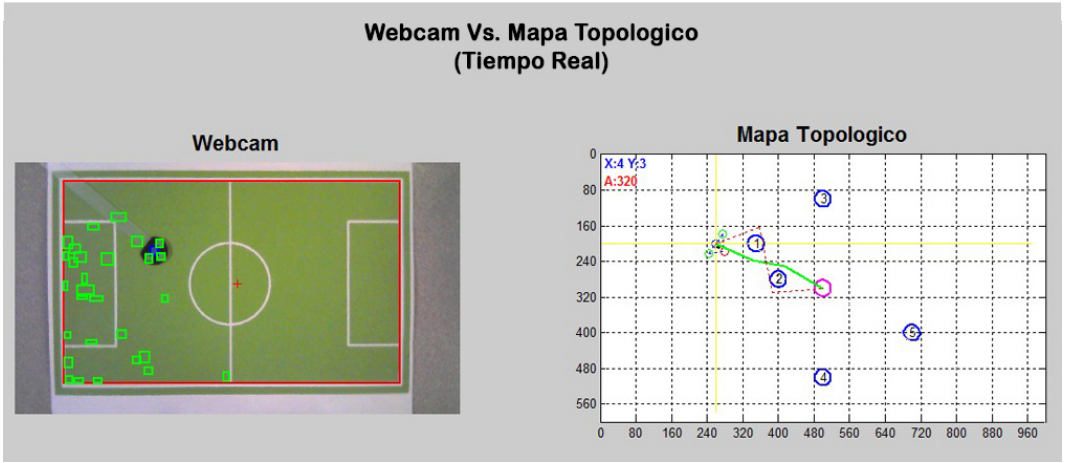



Figura 5.7.25: Trayectorias más corta sobre el segundo obstáculo Mapa Topológico.



# GLOSARIO

**INFERENCIA:** evidencias u observaciones empleadas para actualizar o inferir la probabilidad de que una hipótesis pueda ser cierta.

**LANDMARKS:** localizaciones particulares fácilmente identificables por el sistema sensorial del robot.

**LOCALIZACIÓN:** cálculo de la posición del robot en el entorno de trabajo.

**MAPEO ROBÓTICO:** adquisición y elaboración de modelos espaciales de entornos físicos a través de sensores que permitan crear una abstracción del entorno.

**MODELO PROBABILÍSTICO:** forma que pueden tomar un conjunto de datos obtenidos de muestreos de datos con comportamiento que se supone aleatorio.

**NAVEGACIÓN:** metodología que permite guiar el curso de un robot móvil a través de un entorno, con la capacidad de reaccionar ante situaciones inesperadas.

**ROBOT MÓVIL:** máquina autónoma capaz de moverse en un entorno.

**VEROSIMILITUD:** parámetros de un modelo estadístico que permite realizar inferencias acerca de su valor a partir de un conjunto de observaciones.

# PRIMER ANEXO

Listing 1: Codigo Matlab

```
%% Limpiar

close all                                   % Cerrar ventanas figuras
clear all                                   % Limpiar variables
clc                                         % Limpiar consola

%% Variables

size_grid = 80;                             % Tamaño division grilla
box_white(1) = 0;   box_white(2) = 0;       % Ajuste 0 coordenadas campo
area_W = 500000;    rang_W = 75000;         % Tamaño region campo
area_B = 600;       rang_B = 150;           % Tamaño region azul
area_G = 225;       rang_G = 100;           % Tamaño region verde
D_min  = 22;        D_max  = 30;            % Distancia patron

%% Obstaculos

obstaculosDefinidos = {[600,350],...        % Obstaculo No.1
                       [600,250],...        % Obstaculo No.2
                       [500,300],...        % Obstaculo No.3
                       [400,300],...        % Obstaculo No.4
                       [300,300],...        % Obstaculo No.5
                       [1,1]};

%% Opciones de adquision de video

% imaqtool                                  % Ver webcams y sus propiedades
```

```
28 vid = videoinput('winvideo', 1, 'YUY2_1280x720'); ...
                                        % Selección de webcam
29 set(vid, 'FramesPerTrigger', Inf);
30 set(vid, 'ReturnedColorspace', 'rgb')    % Visualización a color
31 vid.FrameGrabInterval = 5;               % Cuadros por segundo
32 start(vid)                               % Inicia visualización
33
34 %% Webcam
35
36 while(vid.FramesAcquired ≤ 250) ...
                                                        % Números de ...
    cuadros para salir del loop
37
38 figure(1); pause(0.25) ...
                                                                % ...
    Abre nueva ventana
39 subplot(1,2,1) ...
                                                                    ...
    % Figura webcam
40 set(gcf, 'Position', get(0, 'ScreenSize')); ...
                                            % Maximiza ventana
41
42 data = getsnapshot(vid); ...
                                                              % ...
    Capturas del video
43 uicontrol('Style', 'pushbutton', 'String', 'Salir',... ...
                                    % Boton de salida
44 'Position', [5 5 90 30], 'Callback', 'break')
45 imshow(data); hold on; ...
                                                                % ...
    Mostrar video
46 title('Webcam', 'FontSize', 16, 'FontWeight', 'bold') ...
                                     % Titulo figura 1
47
```

```
48 %% HSV
49
50 val = 100;  sat = 100; ...
                                                                    % ...
      Valores de (SV) hsv
51
52 if nargin > 0
53     if nargin < 3
54         sat = [0.1 1]; val = [0.1 1];
55     else
56         sat = [min(sat) max(sat)];
57         val = [min(val) max(val)];
58     end
59
60     dataHSV = rgb2hsv(data); ...
                                                                % ...
         Conversion formato RGB a HSV
61     H = round(dataHSV(:,:,1)*360); ...
                                                            % ...
         Componente H (tono) °
62     S = dataHSV(:,:,2); ...
                                                                ...
         % Componente S (saturacion)
63     V = dataHSV(:,:,3); ...
                                                                ...
         % Componente V (valor)
64     threshI = (S≥sat(1)) & (S≤sat(2)) & (V≥val(1)) & (V≤val(2)); ...
                        % Umbral
65
66     white   = (S < sat(1)) & (V ≥ val(1)); ...
                                                % Color blanco en HSV
67     green   = ((H >  90) & (H ≤ 150)) & threshI; ...
                                            % Color verde en HSV
68     blue    = ((H > 215) & (H ≤ 255)) & threshI; ...
```

```
                                                % Color azul en HSV
69
70  %% White detection
71
72      WB = white|blue; ...
                                                                    ...
            % Color blanco OR color azul
73      ma = ones(35); ...
                                                                      ...
            % Mascara
74      CL = imclose(WB,ma); ...
                                                                  ...
            % Filtro para cerrar areas
75      cancha = imcomplement(imopen(CL,ma)); ...
                                                  % Area del campo ...
            de juego
76
77      data_white = regionprops(cancha, 'Area', 'BoundingBox', ...
            'Centroid');              % Información del campo
78      area_white = [data_white.Area]; ...
                                                       % Are del campo
79      pos_area_white = find(area_white>(area_W-rang_W) & ...
            area_white<(area_W+rang_W)); % Posicion del area
80
81  for object_white = 1:size(pos_area_white,2) ...
                                            % Número de regiones ...
        detectadas
82      box_white = ...
            data_white(pos_area_white(object_white)).BoundingBox; ...
                          % Coordenadas de box de regiones
83      centroid_white = ...
            data_white(pos_area_white(object_white)).Centroid; ...
                        % Coordenadas de centro de regiones
84      rectangle('Position', box_white, 'EdgeColor', 'r', 'LineWidth', ...
```

```
        2)              % Deteccion de color
85      plot(centroid_white(1), centroid_white(2), '-r+') ...
                                        % Graficar centro de regiones
86
87  %% Campo de juego
88
89      m = box_white(3); m = ceil(m/10)*10; ...
                                                    % Tamaño m del campo
90      n = box_white(4); n = ceil(n/10)*10; ...
                                                    % Tamaño n del campo
91
92      if(max(size(box_white)) == 4) ...
                                                        % SI se ...
          detecto campo de juego
93          xi = ceil(box_white(1)); ...
                                                            % ...
              Coordenada X inicial del campo
94          yi = ceil(box_white(2)); ...
                                                            % ...
              Coordenada Y inicial del campo
95          xf = fix(box_white(1)) + fix(box_white(3)); ...
                                            % Coordenada X final ...
              del campo
96          yf = fix(box_white(2)) + fix(box_white(4)); ...
                                            % Coordenada Y final ...
              del campo
97
98          blue  = blue(yi:yf,xi:xf); ...
                                                        % ...
              Matriz del color azul del tamaño de campo
99          mas   = ones(15); ...
                                                                ...
              % Mascara
100         blue  = imopen(blue,mas); ...
```

```
                                                                    % ...
            Filtro de apertura
101         green = green(yi:yf,xi:xf); ...
                                                                    % ...
            Matriz del color verde del tamaño de campo
102         mas   = ones(3); ...
                                                                    ...
            % Mascara
103         green = imerode(green,mas); ...
                                                                    % ...
            Filtro para erosionar areas
104         mas   = ones(5); ...
                                                                    ...
            % Mascara
105         green = imclose(green,mas); ...
                                                                    % ...
            Filtro para cerrar areas
106     end
107 end
108 end
109
110 %% Blue detection
111
112     data_blue = regionprops(blue, 'Area', 'BoundingBox', ...
            'Centroid');                % Información de elementos azules
113     area_blue = [data_blue.Area]; ...
                                                          % Areas ...
            azules
114     pos_area_blue = find(area_blue>(area_B-rang_B) & ...
            area_blue<(area_B+rang_B));     % Posicion de area azul
115     clearvars center_blue ...
                                                                % ...
            Borrar variable center_blue
116
```

```
117 for object_blue = 1:size(pos_area_blue,2) ...
                                                % Número de regiones ...
       detectadas
118     box_blue = data_blue(pos_area_blue(object_blue)).BoundingBox; ...
                              % Coordenadas de box de regiones
119     box_blue(1) = box_blue(1) + box_white(1); ...
                                                % Indexar regiones en ...
          campo
120     box_blue(2) = box_blue(2) + box_white(2); ...
                                                % Indexar regiones en ...
          campo
121     centroid_blue = data_blue(pos_area_blue(object_blue)).Centroid; ...
                            % Coordenadas de centro de regiones
122     center_blue(:,:,object_blue) = centroid_blue; ...
                                             % Estructura con las ...
          coordenadas de centro de regiones
123     rectangle('Position', box_blue, 'EdgeColor', 'b', 'LineWidth', ...
          2)                % Deteccion de color
124 end
125
126 %% Green detection
127
128     data_green = regionprops(green, 'Area', 'BoundingBox', ...
          'Centroid');              % Información de elementos
129     area_green = [data_green.Area]; ...
                                                          % Areas verdes
130     pos_area_green = find(area_green>(area_G-rang_G) & ...
          area_green<(area_G+rang_G)); % Posicion de area verde
131     clearvars center_green ...
                                                                  % ...
          Borrar variable center_green
132
133 for object_green = 1:size(pos_area_green,2) ...
                                               % Número de regiones ...
```

```
    detectadas
134     box_green = ...
            data_green(pos_area_green(object_green)).BoundingBox; ...
                        % Coordenadas de box de regiones
135     box_green(1) = box_green(1) + box_white(1); ...
                                                % Indexar regiones en campo
136     box_green(2) = box_green(2) + box_white(2); ...
                                                % Indexar regiones en campo
137     centroid_green = ...
            data_green(pos_area_green(object_green)).Centroid; ...
                    % Coordenadas de centro de regiones
138     center_green(:,:,object_green) = centroid_green; ...
                                        % Estructura con las ...
            coordenadas de centro de regiones
139     rectangle('Position', box_green, 'EdgeColor', 'g', 'LineWidth', ...
            2)              % Deteccion de color
140 end
141
142 %% Posicion y direccion robot
143
144     subplot(1,2,2) ...
                                                                        ...
          % Figura mapa topologico
145
146 %% Deteccion de patron 1 BGGG
147
148     if (object_blue≥1 & object_green≥3) ...
                                                % Deteccion de ...
          areas para 1 patron
149          cont_B = 0; ...
                                                                      ...
              % Contador de regiones azules = 0
150          clearvars BG ...
                                                                    ...
```

```
            % Borrar variable BG
151
152         for CB = 1:2:(object_blue*2)-1 ...
                                                    % Total de ...
                coordenadas azules
153             cont_G = 0; ...
                                                                ...
                % Contador de regiones verdes = 0
154             cont_B = cont_B+1; ...
                                                            ...
                % Contador de regiones azules
155             XB = CB; ...
                                                                ...
                % Coordenada en X de regiones azules
156             YB = CB+1; ...
                                                            ...
                % Coordenada en Y de regiones azules
157
158             for CG = 1:2:(object_green*2)-1 ...
                                                  % Total de ...
                    coordenadas verdes
159                 cont_G = cont_G+1; ...
                                                            ...
                    % Contador de regiones verdes
160                 XG = CG; ...
                                                                ...
                    % Coordenada en X de regiones verdes
161                 YG = CG+1; ...
                                                            ...
                    % Coordenada en Y de regiones verdes
162                 BG(cont_G,cont_B,:) = ...
                        abs(sqrt(((center_blue(XB)-center_green(XG))^2) ...
                        + ((center_blue(YB)-center_green(YG)))^2));
163             end
```

```
164         end
165
166         DMIN1    = min(min(BG)); ...
                                                                    % ...
               Distancia minima 1 entre region azul y verde
167         [f1 c1] = find(BG==DMIN1); ...
                                                                % ...
               Posicion de distancia minima 1
168         M_BLUE   = sort(BG(:,c1)); ...
                                                                 % ...
               Ordenar columna donde se encuentran distancias minimas
169         DMIN2    = M_BLUE(2,1); ...
                                                                    % ...
               Distancia minima 2 entre region azul y verde
170         [f2 c2] = find(BG==DMIN2); ...
                                                                % ...
               Posicion de distancia minima 2
171         DMIN3    = M_BLUE(3,1); ...
                                                                    % ...
               Distancia minima 3 entre region azul y verde
172         [f3 c3] = find(BG==DMIN3); ...
                                                                % ...
               Posicion de distancia minima 3
173
174         if(DMIN1≥D_min & DMIN1≤D_max & DMIN2≥D_min & DMIN2≤D_max && ...
               DMIN3≥D_min && DMIN3≤D_max)
175
176         XB_1 = center_blue(c1*2-1); ...
                                                               % ...
               Coordenada X region azul del patron
177         YB_1 = center_blue(c1*2); ...
                                                                % ...
               Coordenada Y region azul del patron
178         X_blue = center_blue(c1*2-1); ...
```

```
                                                                % ...
            Coordenada X region azul del patron
179         Y_blue = center_blue(c1*2); ...
                                                                  % ...
            Coordenada Y region azul del patron
180
181         axis([0 m 0 n]); hold off ...
                                                                    % ...
            Dimension de ejes
182         plot(XB_1, YB_1, 'bo'); hold on ...
                                                               % Graficar ...
            region azul del patron
183
184         set(gca, 'yDir', 'reverse'); ...
                                                                % ...
            Invertir eje Y
185         plot([XB_1,m], [YB_1,YB_1], 'y'); plot([XB_1,XB_1], ...
            [YB_1,0], 'y')          % Trazar cruz a patron detectado
186         plot([XB_1,0], [YB_1,YB_1], 'y'); plot([XB_1,XB_1], ...
            [YB_1,n], 'y')          % Trazar cruz a patron detectado
187
188         XG_1 = center_green(f1*2-1); ...
                                                                % ...
            Coordenada X region verde 1 patron
189         YG_1 = center_green(f1*2); ...
                                                                  % ...
            Coordenada Y region verde 1 patron
190         XG_2 = center_green(f2*2-1); ...
                                                                % ...
            Coordenada X region verde 2 patron
191         YG_2 = center_green(f2*2); ...
                                                                  % ...
            Coordenada Y region verde 2 patron
192         XG_3 = center_green(f3*2-1); ...
```

```
                                                                    % ...
            Coordenada X region verde 3 patron
193         YG_3 = center_green(f3*2); ...
                                                                      % ...
            Coordenada Y region verde 3 patron
194
195 %% Distancias centro patron
196
197         X = (XG_1+XG_2+XG_3) / 3; ...
                                                                     % ...
            Coordenada X centro del patron
198         Y = (YG_1+YG_2+YG_3) / 3; ...
                                                                     % ...
            Coordenada Y centro del patron
199         plot(X, Y, 'k.') ...
                                                                        ...
            % Grafica centro del patron
200
201         A = abs(sqrt(((X-XG_1)^2) + ((Y-YG_1)^2))); ...
                                            % Distancia del centro ...
            del patron a verde 1
202         B = abs(sqrt(((X-XG_2)^2) + ((Y-YG_2)^2))); ...
                                            % Distancia del centro ...
            del patron a verde 2
203         C = abs(sqrt(((X-XG_3)^2) + ((Y-YG_3)^2))); ...
                                            % Distancia del centro ...
            del patron a verde 3
204         x = XB_1; y = YB_1; ...
                                                                    ...
            % Coordenadas XY azul del patron
205
206 %% Direccion del robot (0 - 360°)
207
208         if(A<B & A<C) ...
```

```
                                                                    ...
        % Frente del robot G1
209     plot(XG_1, YG_1, 'ro') ...
                                                                % ...
           Frente del robot
210     plot(XG_2, YG_2, 'go') ...
                                                                % ...
           Lateral del robot
211     plot(XG_3, YG_3, 'go') ...
                                                                % ...
           Lateral del robot
212     x_pos = XG_1 - XB_1; ...
                                                                  ...
           % Definen si angulo es mayor > 180°
213     y_pos = YB_1 - YG_1; ...
                                                                  ...
           % Definen si angulo es mayor > 180°
214     A1  = sqrt(((x-XG_1)^2) + ((y-YG_1))^2);
215     B1  = sqrt(((x-(A1+x))^2) + ((y-y))^2);
216     C1  = sqrt(((XG_1-(A1+x))^2) + ((YG_1-y))^2);
217     ang = acosd(((A1^2)+(B1^2) - (C1^2))/(2*(A1*B1))); ...
                                % Angulo direccion robot  0  ...
           <-> 180°
218     if(y_pos<0); ang = 360-ang; end ...
                                                  % Angulo ...
           direccion robot 181 <-> 360°
219
220   elseif(B<A & B<C) ...
                                                                ...
        % Frente del robot G2
221     plot(XG_1, YG_1, 'go') ...
                                                                % ...
           Lateral del robot
222     plot(XG_2, YG_2, 'ro') ...
```

```
                                                                    % ...
                Frente del robot
223             plot(XG_3, YG_3, 'go') ...
                                                                    % ...
                Lateral del robot
224             x_pos = XG_2 - XB_1; ...
                                                                      ...
                % Definen si angulo es mayor > 180°
225             y_pos = YB_1 - YG_2; ...
                                                                      ...
                % Definen si angulo es mayor > 180°
226             A1  = sqrt(((x-XG_2)^2) + ((y-YG_2))^2);
227             B1  = sqrt(((x-(A1+x))^2) + ((y-y))^2);
228             C1  = sqrt(((XG_2-(A1+x))^2) + ((YG_2-y))^2);
229             ang = acosd(((A1^2)+(B1^2) - (C1^2))/(2*(A1*B1))); ...
                                        % Angulo direccion robot  0  ...
                <-> 180°
230             if(y_pos<0); ang = 360-ang; end ...
                                                          % Angulo ...
                direccion robot 181 <-> 360°
231
232         else ...

            % Frente del robot G3
233             plot(XG_1, YG_1, 'go') ...
                                                                    % ...
                Lateral del robot
234             plot(XG_2, YG_2, 'go') ...
                                                                    % ...
                Lateral del robot
235             plot(XG_3, YG_3, 'ro') ...
                                                                    % ...
                Frente del robot
236             x_pos = XG_3 - XB_1; ...
```

```
                                                                    ...
                % Definen si angulo es mayor > 180°
237             y_pos = YB_1 - YG_3; ...
                                                                    ...
                % Definen si angulo es mayor > 180°
238             A1  = sqrt(((x-XG_3)^2) + ((y-YG_3))^2);
239             B1  = sqrt(((x-(A1+x))^2) + ((y-y))^2);
240             C1  = sqrt(((XG_3-(A1+x))^2) + ((YG_3-y))^2);
241             ang = acosd(((A1^2)+(B1^2) - (C1^2))/(2*(A1*B1))); ...
                                    % Angulo direccion robot  0  ...
                <-> 180°
242             if(y_pos<0); ang = 360-ang; end ...
                                                        % Angulo ...
                direccion robot 181 <-> 360°
243         end
244
245         plot([XG_1 XG_2], [YG_1 YG_2], 'b:') ...
                                                % Triangulo patron
246         plot([XG_1 XG_3], [YG_1 YG_3], 'b:') ...
                                                % Triangulo patron
247         plot([XG_2 XG_3], [YG_2 YG_3], 'b:') ...
                                                % Triangulo patron
248
249 %% Constantes posicion elementos
250
251         robot = [X_blue,Y_blue];                            % ...
                posicion del robot (mouse)
252
253         Otext = {'1','2','3','4','5','6'};                  % ...
                numeros de cada obstaculo
254         destino = [800,500];                                % ...
                Posición destino
255
256         obstaculosDefinidos = {[600,350],...                % ...
```

```
            Obstaculo No.1
257                                 [600,250],...                   % ...
                                       Obstaculo No.2
258                                 [500,300],...                   % ...
                                       Obstaculo No.3
259                                 [400,300],...                   % ...
                                       Obstaculo No.4
260                                 [300,300],...                   % ...
                                       Obstaculo No.5
261                                 [1,1]};
262
263 %% Ordenar obstaculos
264
265         dis = {}; ...
                                                                        ...
            % Inicializo celda
266         for i = 1:(length(obstaculosDefinidos)) ...
                                                % Iteracion 1 - hasta el ...
            numero de obstaculos
267             dis{i} = distancia(robot, obstaculosDefinidos{i}); ...
                                % Calcula distancia entre robot y cada ...
                obstaculo
268         end
269
270         A = [1 dis{1}; 2 dis{2}; 3 dis{3}; 4 dis{4}; 5 dis{5}]; % ...
            Guardo cada distancia con un ID
271         [B,k] = sort(A(:,2)); ...
                                                              % Ordena de ...
            forma descendente ...
272         B = [A(k) B]; ...
                                                                    % ...
            las distancias
273         B(:,1); ...
                                                                       ...
```

```
            % ID de cada obstaculo
274         obstaculos = {obstaculosDefinidos{B(:,1)}}; ...
                                % Obstaculos ordenados segun ...
            distancia a ROBOT
275
276 %% Numero de cada obstaculo
277
278         for i=1:1:5
279             text(obstaculos{i}(1)-5,obstaculos{i}(2),Otext(i),'fontsize',10)
280         end
281         for j=1:1:5
282             text(obstaculosDefinidos{j}(1)-3,obstaculosDefinidos{j}(2)+15,Otext(j),'fontsi
283         end
284 %% Constantes para dibujo circulos
285
286         d = 40;                                     % Diametro del ...
            robot
287         r = d/2;                                    % Radio del robot
288         theta = linspace(0,2*pi,600);               % Theta
289         rho = ones(1,600)*r;                        % Rho
290
291         plot(robot(1),robot(2),'Ob');               % Grafica ...
            posicion inicial
292         plot(destino(1),destino(2),'Ob');           % Grafica ...
            posicion destino
293
294 %% Grafica de robot y obstaculos - Circulos
295
296         robotCirculo = [destino(1) destino(2)];                    ...
            % DESTINO
297         [robotCirculoX, robotCirculoY] = pol2cart(theta,rho);      ...
            % Conversion de coordenadas cartesianas a polares
298         robotCirculoX = robotCirculoX + robotCirculo(1);           ...
            % Coordenada X obstaculo
```

```
299         robotCirculoY = robotCirculoY + robotCirculo(2);            ...
                % Coordenada Y obstaculo
300         plot(robotCirculoX, robotCirculoY, 'm', 'linewidth', 2);
301
302         for l=1:1:5  %% 6
303             robotCirculo = [obstaculos{l}(1) obstaculos{l}(2)];     ...
                    % Obstaculo No. 1
304             [robotCirculoX, robotCirculoY] = pol2cart(theta,rho);   ...
                    % Conversion de coordenadas cartesianas a polares
305             robotCirculoX = robotCirculoX + robotCirculo(1);        ...
                    % Coordenada X obstaculo
306             robotCirculoY = robotCirculoY + robotCirculo(2);        ...
                    % Coordenada Y obstaculo
307             plot(robotCirculoX, robotCirculoY, 'b', 'linewidth', 2);
308         end
309
310         m2 = 0:0.01:1000;                       % variables para ...
                dibujo linea diagonal...
311         t2 = 0.6*m2;                            % dimensiones cancha
312         hold on                                 % diguja sobre ...
                figure la linea
313         plot(m2,t2,'color', 'white');           % color blanca para ...
                que no se vea
314
315 %% Triangulo rectangulo
316
317     % Linea A
318     Ap1 = [robot(1),destino(2)];
319     Ap2 = [robot(1),robot(2)];
320     A = distancia(Ap1,Ap2);
321
322     % Linea C
323     Cp1 = [destino(1),destino(2)];
324     Cp2 = [robot(1),destino(2)];
```

```
325     C = distancia(Cp1,Cp2);
326
327     % Linea B
328     Bp1 = [destino(1),destino(2)];
329     Bp2 = [robot(1),robot(2)];
330     B = distancia(Bp1,Bp2);
331
332     angulo = asind(A/B);
333
334 %% Angulo>50
335
336     if angulo>50
337
338         flag = 1;                                % Bandera que ...
               establece cual es el proximo destino
339         k = 1;                                   % Indice de la ruta
340         ruta = {k};                              % Celda ruta - ...
               almacena la ruta a seguir por el robot
341         ruta{k} = robot;                         % La primera ...
               posicion corresponde al ROBOT
342         ruta2 = {k};                              % Celda ruta - ...
               almacena la ruta a seguir por el robot
343         ruta2{k} = robot;                         % La primera ...
               posicion corresponde al ROBOT
344
345         while (flag ~= 0)
346
347             obstaculos3 = {};
348             i = 1;
349             j = 1;
350                 d1 = distancia(obstaculos{i},destino);
351                 d2 = distancia(robot,destino);
352
353                 if (d2<d1)                       % no hay obstaculos ...
```

```
                    delante del robot
354                  plot([robot(1) destino(1)],[robot(2) ...
                         destino(2)],'r','LineWidth',2);
355                  obstaculos{i}=destino;
356              end
357           while (i<length(obstaculos))       %obstaculos= ordenados
358              if (((obstaculos{i}(1)<destino(1)+200 && ...
                    obstaculos{i}(1)>destino(1)-200) && ...
                    (obstaculos{i}(2)>destino(2)+50 && ...
                    obstaculos{i}(2)<robot(2)-50)) || ...
                    obstaculos{i}(1)==destino(1) || ...
                    ((obstaculos{i}(1)<destino(1)+200 && ...
                    obstaculos{i}(1)>destino(1)-200) && ...
                    (obstaculos{i}(2)<destino(2)+50 && ...
                    obstaculos{i}(2)>robot(2)-50)));
359
360              obstaculos3{j} = obstaculos{i};
361
362              j = j+1;
363              end
364              i = i+1;
365          end
366
367      if isempty(obstaculos3)        % Si no hay obstaculos
368          flag=0;
369      else                           % Si no, busca nuevo destino
370
371      puntoArriba = 0;
372      puntoAbajo = 0;
373      flag = 0;
374      n1 = 0;                                % indice obstaculos
375      toque = 0;                             % flag "SI" hay ...
             toque o "NO"
376
```

```
377         while(toque == 0) && n1<length(obstaculos3)
378
379             n1 = n1 + 1;
380
381             dx1=robot(1)- 25;
382             dy1=robot(2);
383             dx2=robot(1)+ 25;
384             dy2=robot(2);
385
386             robotx={[dx1,dy1,dx2,dy2]};
387
388             px1=destino(1)- 25;
389             py1=destino(2);
390             px2=destino(1)+ 25;
391             py2=destino(2);
392
393             destinox={[px1,py1,px2,py2]};
394
395             ox1=obstaculos3{n1}(1)-25;
396             oy1=obstaculos3{n1}(2);
397             ox2=obstaculos3{n1}(1)+ 25;
398             oy2=obstaculos3{n1}(2);
399
400             obstaculox={[ox1,oy1,ox2,oy2]};
401             ruta{k} = robot;
402             ruta{k+1}=destino;
403             toque = ...
                    obstaculo(obstaculos3{n1},robotx,destinox,obstaculox,angulo);
404         end
405
406         if (toque == 1)
407
408             ox1 = obstaculos3{n1}(1)- 25;
409             oy1 = obstaculos3{n1}(2);
```

```
410
411             ox2 = obstaculos3{n1}(1)+ 25;
412             oy2 = obstaculos3{n1}(2);
413
414             obstaculox = {[ox1,oy1,ox2,oy2]};
415
416             j = 1;
417
418                 puntoArriba=[obstaculos3{n1}(1)+50,obstaculos3{n1}(2)];
419                 puntoAbajo=[obstaculos3{n1}(1)-50,obstaculos3{n1}(2)];
420
421                 dUP = distancia( ruta{k},puntoArriba);
422                 dDOWN = distancia( ruta{k},puntoAbajo);
423
424                 ruta{k} = robot;
425                 ruta{3}=destino;
426
427         if dUP<dDOWN
428               k = k + 1;
429               ruta{k} = puntoArriba;
430               ruta{3}=destino;
431         end
432             if dUP>dDOWN
433                   k = k + 1;
434                   ruta{k} = puntoAbajo;
435                   ruta{3}=destino;
436             end
437         end
438         end
439         end
440
441 %% Angulo<50
442
443     if angulo<50
```

```
444
445         flag = 1;                                   % Bandera que ...
               establece cual es el proximo destino
446         k = 1;                                      % Indice de la ruta
447         ruta = {k};                                 % Celda ruta - ...
               almacena la ruta a seguir por el robot
448         ruta{k} = robot;                            % La primera ...
               posicion corresponde al ROBOT
449         ruta2{k} = robot;                            % La primera ...
               posicion corresponde al ROBOT
450         ruta{k} = robot;                            % La primera ...
               posicion corresponde al ROBOT
451         while (flag ~= 0)
452               obstaculos2 = obsEntrePuntos(obstaculos,ruta{k},destino);
453               if isempty(obstaculos2)               % Si no hay obstaculos
454                   flag = 0;
455               else                                  % Si no, busca nuevo ...
                     destino
456                    [puntoArriba,puntoAbajo,flag] = ...
                          rutas(ruta{k},destino,obstaculos2,r,angulo);
457               end
458
459               k = k + 1;                            % Aumenta indice de ruta
460               if flag == 0                          % Si no hay obstaculos
461                   ruta{k} = destino;                % Destino final
462                   ruta2{k} = destino;                % Destino final
463               end
464               if flag == 1                          % Abre por Arriba
465                   ruta{k} = puntoArriba;            % Proximo destino - ...
                        parte superior obstaculo
466                   ruta2{k} = puntoAbajo;            % Destino final
467               end
468               if flag == 2                          % Abre por Abajo
469                   ruta{k} = puntoAbajo;             % Proximo destino - ...
```

```
                    parte inferior obstaculo
470                 ruta2{k} = puntoArriba;        % Destino final
471             end
472         end
473
474     end
475         for i = 1:(length(ruta)-1)          % Recorre cada posicion ...
               de ruta
476             plot([ruta{i}(1) ruta{i + 1}(1)],[ruta{i}(2) ruta{i + ...
                   1}(2)],'g','LineWidth',2); % Grafica la ruta ...
                   uniendo cada coordenada
477             plot([ruta2{i}(1) ruta2{i + 1}(1)],[ruta2{i}(2) ...
                   ruta2{i + 1}(2)],':r','LineWidth',1); % Grafica la ...
                   ruta uniendo cada coordenada
478         end
479
480 %% Mapa Topologico
481
482         title('Mapa Topologico', 'FontSize', 16, 'FontWeight', ...
               'bold')              % Titulo figura 2
483         axis image; grid on; axh = gca; ...
                                                        % Ejes y grilla
484         set(axh, 'XTick', 0:size_grid:m) ...
                                                       % Division ...
               de grilla en eje X
485         set(axh, 'YTick', 0:size_grid:n) ...
                                                       % Division ...
               de grilla en eje Y
486
487 %% Mostrar informacion en figura 2
488
489         coord_blue_map = text((10), (20),... ...
                                                    % Posicion XY ...
               del robot
```

```
490         strcat('X: ', num2str(ceil(X_blue/size_grid)),...
491         ' Y: ', num2str(ceil(Y_blue/size_grid))));
492         set(coord_blue_map, 'FontName', 'Arial', 'FontWeight',...
493         'bold', 'FontSize', 10, 'Color', 'blue');
494
495         coord_ang = text((10), (60), strcat('A: ', ...
               num2str(ceil(ang))));            % Angulo direccion del ...
               robot
496         set(coord_ang, 'FontName', 'Arial', 'FontWeight',...
497         'bold', 'FontSize', 10, 'Color', 'red');
498         end
499     end
500
501 end
502
503  %% Detener y cerrar video
504
505 stop(vid); ...
                                                                          ...
       % Parar video
506 flushdata(vid); ...
                                                                        ...
       % Elimina todos los cuadros almacenados en la memoria
507 delete(vid); ...
                                                                     ...
       % Borrar video almacenados en la memoria
508 close all ...
                                                                          ...
       % Cerrar ventanas figuras
```

# CONCLUSIONES

El proceso utilizado para el reconocimiento del patrón de punto a partir del procesamiento digital de imágenes hace posible realizar tareas de navegación básicas de una manera más rápida que implementando la información de los sensores propios del robot; se logra cumplir con el objetivo básico de planificar una trayectoria que le permita al robot moverse desde una posición inicial a una final en un entorno.

El algoritmo propuesto tiende a ser lento, debido a derivados de la propia arquitectura del lenguaje utilizado en cuanto a la implementación de sus funciones. Esto se hace evidente en la captura de la imagen para el reconocimiento del patrón de punto que representa al robot. El tiempo estimado entre cada imagen es de $200\ ms$, lo que aleja al sistema de funcionar en tiempo real.

La planificación de la ruta no considera ninguna característica del vehículo, ya que el objetivo inicial del algoritmo es la de garantizar la continuidad en posición. Por ello, debe recibir un tratamiento que la adecúe para su seguimiento. Este tratamiento es llevado a cabo en la etapa de control del robot.

# RECOMENDACIONES

Para mejorar la captura de la imagen del entorno donde se encuentra el robot se hace necesario utilizar un tipo de cámara que no posea auto compensación de ilumina-ción y proporcional al tamaño del entorno seleccionar la resolución de la cámara para tener una imagen de mayor calidad.

Para mejorar la velocidad de procesamiento del algoritmo es conveniente migrar a un lenguaje más óptimo con mayor rendimiento. Al pensar en una implementación para un desarrollo comercial lo más acertado es utilizar software libre, por motivos de licencias.

El método de tangentes utilizado para determinar las trayectorias está basado en líneas rectas, lo que hace que la trayectoria sea muy directa sobre cada obstáculo. Para mejorar esta consideración se puede trabajar con este mismo método, tomando una mayor cantidad de puntos de referencia sobre el obstáculo, logrando una trayectoria más curva.

En futuros trabajos es posible adaptar este proyecto a otras plataformas móviles y cámaras simplemente variando algunos parámetros en el software y algunas adaptacio-nes en el prototipo real.

# REFERENCIAS

[1] J, Pulido Fentanes. *Exploración y reconstrucción tridimensional de entornos mediante robots móviles*. 2012. [En línea]. Disponible: http://uvadoc.uva.es/handle/10324/2023.

[2] E. R. C. Hernández. *Desarrollo de un sistema de visión para la localización y navegación de robots humanoides*. 2011. [En línea]. Disponible: http://homepage.cem.itesm.mx/aaceves/Bogobots/seminario/Tesis_Erick_Cruz.pdf.

[3] D. G. López. *Aplicación del muestreo bayesiano en robots móviles: estrategias para localización y estimación de mapas del entorno*. 2000. [En línea]. Disponible: http://rua.ua.es/dspace/handle/10045/9887.

[4] B. Kuipers, Y-T. Byun. *A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations*. 1991. [En línea]. Disponible: http://scholar.google.com/citations?view_op=view_citation&hl=es&user=H1XVLwsAAAAJ&citation_for_view=H1XVLwsAAAAJ:d1gkVwhDpl0C.

[5] E. R. González. *Localización mediante differential evolution de un robot móvil*. 2011. [En línea]. Disponible: http://hdl.handle.net/10016/13590.

[6] J. L. B. Domínguez. *Reconstrucción del entorno mediante rejillas de ocupación*. 2012. [En línea]. Disponible: http://e-archivo.uc3m.es/bitstream/handle/10016/14368/memoria%20PFC%20Jose_Luis__Buedo_Dominguez.pdf?sequence=1.

[7] A. S. M. V. A. V. M. M. J. Gómez-Moreno. *Impletación de un algoritmo de localización basado en un método de montecarlo para un robot movil omidirecccional*. 2012. [En línea]. Disponible: http://wks.gii.upv.es/cobami/files/JGomez_JJAA2012.pdf.

[8] D. A. N. García. *Contribución a la autolocalización de robots móviles basada en la fusión de información multisensorial*. 2009. [En línea]. Disponible: http://www.upv.es/.

[9] M. R. L. J. A. R. D. Lesley Ofelia Mesa Páez. *Descripción general cde la inferencia bayesiana y sus aplicaciones*. 2011. [En línea]. Disponible: http://www.urosario.edu.co/urosario_files/38/38e60ea0-497e-4197-913d-e156ae0bb084.pdf.

[10] J. A. C. Suárez. *Localización y seguimiento de trayectorias con robots caminantes en entornos naturales*. 2007. [En línea]. Disponible: http://eprints.ucm.es/7967/.

[11] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins. *Digital Image Processing Using MATLAB*. (Cáp. 10). 2009. [Libro 2nd ed]. Disponible: ISBN: 978-0-9820854-0-0.

[12] Eduardo L. F.. *Descripcion, comparacion y ejemplos de usos de los toolbox de procesado digital de imagenes de MATLAB*. 2012. [En línea]. Disponible: http://oa.upm.es/14016/2/PFC_EDUARDO_LAORDEN_FITER_B.pdf.

[13] Pentagono Univerdsidad de los Andes. *Tutorial de MATLAB*. [En línea]. Disponible: http://pentagono.uniandes.edu.co/index.php?option=com_content&view=article&id=17&Itemid=132.

[14] Antonio Herrera. *Modelos de color (RGB, CMYK, HSV/HSL)*. 2014. [En línea]. Disponible: http://ahenav.com/2014/04/09/modelos-de-color/.

[15] Alejandro S. M.. *Control Visual Para Brazo Robotnik*. 2013. [En línea]. Disponible: http://repositorio.bib.upct.es:8080/jspui/bitstream/10317/3355/1/pfc5192.pdf