DISEÑO Y CONSTRUCCIÓN DE UNA PLATAFORMA ROBÓTICA PARA EL CONTROL DE FORMACIÓN Y DISTRIBUCIÓN DE TAREAS

DANIEL FELIPE LEÓN CARDONA

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍAS
PROGRAMA DE INGENIERÍA MECATRÓNICA
BUCARAMANGA
2018

DISEÑO Y CONSTRUCCIÓN DE UNA PLATAFORMA ROBÓTICA PARA EL CONTROL DE FORMACIÓN Y DISTRIBUCIÓN DE TAREAS

DANIEL FELIPE LEÓN CARDONA

Trabajo de grado para optar el título de Ingeniero Mecatrónico

Director
M.Sc. Carlos Adolfo Forero González

Asesores
M.Sc. Hernando González Acevedo
M.Sc. Hernán González Acuña

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍAS
PROGRAMA DE INGENIERÍA MECATRÓNICA
BUCARAMANGA
2018

Nota de aceptación:

5.0

Cinco punto Cero.

M.Sc. Carlos Adolfo Forero González
Director de proyecto

M.Sc. Darío José Hernández Bolívar
Evaluador de proyecto

Bucaramanga, 6 de agosto de 2018

Dedicado a la memoria de Adelina, Rosalba y Adriana;
causales de todo lo bello que he encontrado en mi vida.

AGRADECIMIENTOS

Es irónico que, siendo este el primer capítulo del libro, sea este el último en ser redactado, no obstante, tiene todo el sentido ya que, no es sino hasta el final del tramo del sendero de la vida que hemos recorrido, el instante en el que se presenta la oportunidad de hacer una pausa y así, en retrospectiva, evocar a cada persona y suceso que de alguna u otra forma se cruzaron o encontramos a lo largo y ancho de nuestro camino, para este caso, una muy extensa avenida.

En primera instancia, he de agradecer a las Tres personas de la Santísima Trinidad, Dios Padre, Dios Hijo y Dios Espíritu Santo, quienes jamás han dejado de acompañar y guiar mi caminar. A la Santísima Virgen María, quien con su manto me continúa cobijando en su amor de Madre. A mi Ángel custodio, quien seguramente ha tenido que trabajar incontables horas extras.

He de agradecer a Elvira Cardona Arboleda, quien, con su desmesurado apoyo e incondicional amor, me ha demostrado en cada día de mi vida la existencia de un Dios verdadero, un Ser supremo que, apiadándose de mí, me concedió la gracia, el honor y la dicha de poder llamar madre a ese maravilloso ángel. A mi padre, Pedro Alonso León Rueda, quien fue mi primer maestro en la senda de la investigación, quien sembró y cultivó en mí la semilla de la curiosidad, quien me enseñó la importancia de soñar y perseverar por esas metas, quien junto a mi madre me heredaron el gusto por la lectura, el deporte y las artes. A mi hermano Germán Darío, quien fue mi primer cómplice en la vida y mi más destacado alumno de picardías, quien, pese a carecer de la contextura de Sancho, también hacia, y hace hoy día, de mi de escudero.

También he de agradecer eternamente a Dios por otorgarme la alegría y la responsabilidad de atesorar a la criatura más bella de su creación, con quien un día

entrelazamos caminos y desde ese entonces permanecemos juntos, con quien comparto mis sueños, temores, alegrías y angustias, quien con su ejemplo me enseña cada día a perseverar y junto a quien he aprendido que avanzar en la vida, no siempre significa andar hacia adelante, agradezco a Ana María por la indescriptible alegría y orgullo que me produce ver su sonrisa, a ella, todo mi amor, en palabras de Rosemonde Gérard “Pues ya ves que cada día te amo más, hoy más que ayer y mucho menos que mañana”.

Agradezco al Centro de Investigación en Ingeniería y Organizaciones (CIIO) de la Universidad Autónoma de Bucaramanga por el apoyo económico recibido a través del proyecto: “COMPARACIÓN DE 2 TÉNICAS DE CONTROL APLICADAS A LA FORMACIÓN DE ROBOTS MÓVILES COOPERATIVOS”. También a Leidy Camacho, quien en representación del CIIO, realizó oportunamente las gestiones requeridas por el proyecto.

Agradezco inmensamente a los docentes, auxiliares y demás personal de la Universidad Autónoma de Bucaramanga quienes de una u otra forma me brindaron su apoyo, conocimiento y comprensión en el transcurso de mi desarrollo profesional en la institución, especialmente, al M.Sc. Carlos Forero, quien fue mi director de proyecto y compañero de aprendizaje en el tema; al Ph.D. Sebastián Roa, quien fue mi segundo mentor en la senda de la investigación; al M.Sc. Hernán González, sin quien me hubiese graduado dos años antes, pero sin los valiosos conocimientos que adquirí en este proceso; al M.Sc. Hernando González, a quien considero un verdadero maestro; a la MBA-Ing. Nayibe Chio, quien me recibió y acompañó durante su dirección; al M.Sc. Darío Hernández, quien me aconsejó desde su experiencia y fue justo en la evaluación de la tesis; al Ing. Mg. Silvio Cuello, quien me enseñó a “echarle candela” a los desafíos que plantea nuestra vida profesional; al M.Sc. Víctor Ardila, quien me enseñó a remontar pese a las “ostias” que te de la vida; a Marvin Torres, maestro de generaciones; a Jegnny Roció Pabón Ortega, a

ella, mi eterna gratitud, mi profunda admiración y mi más sincera amistad; a Gabriel Franco, por su apoyo en los procesos de manufactura; a María Eugenia, Angie y Krystel, por la extraordinaria paciencia que me brindaron; a don Omar y su equipo de seguridad, gracias por mantener segura nuestra institución; a María y su equipo de servicios generales, es gracias a su arduo trabajo que la estancia en la institución se hiciera más llevadera.

Agradezco al doctor Magnus Egerstedt por su modelo matemático, es gracias a ese "truco inteligente" que se me facilitó la comprensión de los conceptos desarrollados en este proyecto.

Agradezco al doctor Elías Rigoberto Ledesma Orozco, director del Departamento de Ingeniería Mecánica de la Universidad de Guanajuato (división de ingenierías, campus Irapuato-Salamanca, México), quien mediante el proyecto: "DESARROLLO DEL PROCESO DE DIGITALIZACIÓN E IMPRESIÓN TRIDIMENSIONAL (3D)", me abrió las puertas a una nueva cultura, una nueva institución, un nuevo país y una gran experiencia como estudiante investigador.

Agradezco a aquellas personas con las que tuve el honor de compartir espacios de aprendizaje, a los ingenieros Andrés Velásquez y José García, por su amistad y asesoramiento en el proyecto; al "dotó de la ingeniería", Edwin Solano, a quien admiro y considero un gran ser humano y profesional; al Ing. Cristian Jaimes, a quien le auguro muchos éxitos en ese viaje que acaba de emprender; a los ingenieros Julián Mantilla, Raúl Di Marco, Jorge Rangel, Jesús Monsalve, Julián Serrano, Olmer Villamizar, a Jessica Paola y Martha Patricia, gracias por dejarme entrar en sus vidas; a las tres ingenieras "Powerpuff", Dalya, Luisa y Verónica, sin olvidar a la cuarta, Marien, gracias por compartir buenos y no tan buenos momentos.

Agradezco a esas personas que, con tiempo y cariño, han gravado su nombre en mi corazón y me han apoyado desde siempre, a Gloria, Silvia y toda la familia Quintero Medina; a Elsa, Felipe, Marcela y toda la familia Pérez Arciniegas; a Diego Fernando y toda la familia Díaz Prada; a los demás “compadres”, Aura, Iván, Miguel, Sandra, María Elena, Yezica y Manuel, gracias por las palabras de aliento; a toda la Familia León Cardona, quienes jamás perdieron la esperanza; a las familias Rosso Cerón y Lineros Pinto, gracias por estar en las buenas y en las difíciles; al CIDES, por permitirnos tantos momentos a Ana y a mí; a la PJ Jesed, gracias por mantenerme en sus oraciones, estoy seguro que fueron escuchadas.

En general, agradezco por todos los maestros, por los eventos, por los lugares, por aquellas personas que hoy no están con nosotros y por aquellas que están por venir, por todo lo que ha contribuido a edificar la persona soy hoy día, por todo lo recibido, muchas gracias.

CONTENIDO

Pág.

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

1. MARCO GENERAL

En este primer capítulo se realizará una contextualización del proyecto, en donde se expondrán los antecedentes del mismo, la motivación que conllevó a su realización y la metodología aplicada a su desarrollo.

1.1 INTRODUCCIÓN

En el presente proyecto se describe el paso a paso del desarrollo, construcción e implementación de una plataforma robótica móvil diferencial destinada al control de formaciones y la distribución de tareas. Para ese fin, el documento se ha dividido en tres temáticas principales, 1) la información concerniente a la temática del proyecto, 2) la descripción detallada y progresiva tanto del desarrollo de la plataforma robótica como de su entorno de trabajo y, 3) la divulgación de los resultados obtenidos; con el propósito de introducir gradualmente las temáticas desarrolladas.

En los siguientes capítulos se abordan temas como la robótica cooperativa, la visión artificial, el análisis por elementos finitos, el control de formaciones, la cinemática directa e inversa de un robot móvil diferencial, el control de velocidad de un motor de corriente continua, entre otros. Al hacer la sinergia de esos contenidos, se obtiene el cumplimiento de los objetivos propuestos, resultando en una plataforma robótica de experimentación al alcance de la comunidad educativa. Todo, con la esperanza de posicionar a la Universidad en este campo de la robótica.

Los resultados obtenidos, son un reflejo de los conocimientos adquiridos durante la formación profesional en el programa de Ingeniería Mecatrónica de la Universidad Autónoma de Bucaramanga. Conocimientos que hoy día, se han consolidado mediante la investigación y cursos electivos adicionales al pensum académico.

1.2 OBJETIVOS

A continuación, se enlistan las directrices del proyecto en un marco tanto general como específico.

1.2.1 Objetivo general

Diseñar y construir una plataforma robótica para el control de formación y distribución de tareas.

1.2.2 Objetivos específicos

- Diseñar la plataforma robótica, determinando sus especificaciones técnicas.
- Seleccionar la instrumentación, sensores y actuadores necesarios que permitan el desplazamiento, posicionamiento y orientación espacial de la plataforma robótica.
- Seleccionar e implementar un sistema de comunicación inalámbrica para comunicación entre los robots y el sistema central de control.
- Determinar la estrategia de control aplicada al control de formaciones.
- Realizar la simulación del sistema de control de formaciones.
- Diseñar una interfaz gráfica de usuario para monitorear y graficar en tiempo real la posición y orientación de los robots.
- Realizar la validación del sistema de formación con al menos seis (6) robots.

1.3 PLANTEAMIENTO DEL PROBLEMA

Para cumplir con una tarea o una función preestablecida, la robótica presenta dos enfoques de solución, uno de ellos es la construcción de un robot con la capacidad de realizar todas las actividades requeridas para cumplir el objetivo planteado, mientras que el otro enfoque, es el de distribuir la misma tarea entre varios robots.

Si bien, el primer enfoque presenta mayor facilidad en cuanto a configuración, control y manufactura, este presenta desventajas operacionales. Esto, dado que si el robot presenta un malfuncionamiento que detenga su operación, la tarea establecida no se llevará a cabo ya que el único operador disponible se encuentra fuera de funcionamiento. Es por eso que el segundo enfoque, propósito de este proyecto, aunque con un grado mayor de dificultad en su programación, operación y comunicación, resulta una mejor opción ante el desarrollo de esa misma tarea mediante la distribución de labores entre un conjunto de operadores. Si falla uno de los miembros del conjunto, existirá un suplente que lo releve en su tarea, garantizando con ello la realización del objetivo propuesto.

1.4 JUSTIFICACIÓN

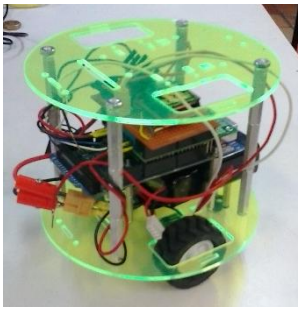
Dado que este tipo de investigación no tiene precedentes en la Universidad, el resolver el problema de la coordinación de las $N \geq 6$ plataformas robóticas para llevar a cabo una tarea específica, representaría la apertura de una nueva línea de investigación en el programa de Ingeniería Mecatrónica, aportando las bases teóricas que rigen este campo del conocimiento, además de suministrar la planta física de experimentación con la cual se podrán probar nuevas teorías de control de formación. La metodología a seguir durante el desarrollo de este trabajo de grado se presenta en la Figura 5 del subcapítulo 1.7 (DISENO METODOLÓGICO).

1.5 ANTECEDENTES

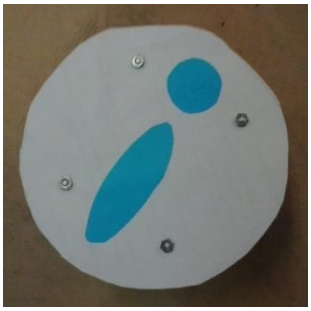
Este proyecto parte de la iniciativa de investigación propuesta en la asignatura ROBÓTICA MÓVIL Y VISIÓN ARTIFICIAL, en donde se construyen plataformas robóticas (Ver Figura 1) capaces de interactuar con su entorno para realizar tareas definidas por el usuario. Para profundizar los conceptos y la práctica desarrollada

en esa materia, se plantea este proyecto como un primer paso en términos de control de formación y distribución de tareas.

Figura 1. Plataforma robótica realizada en el curso electivo Robótica Móvil y Visión Artificial (segundo semestre de 2015). (a) Prototipo. (b) Vista superior de la plataforma. (c) Interfaz gráfica desarrollada. (d) Escenario construido

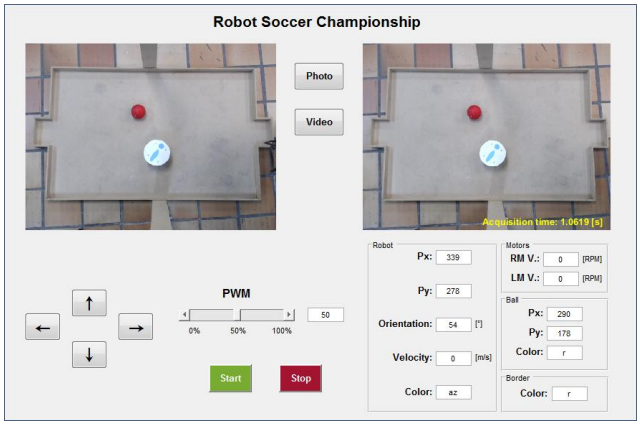


(a)



(b)

(d)



(c)

Fuente: Autor

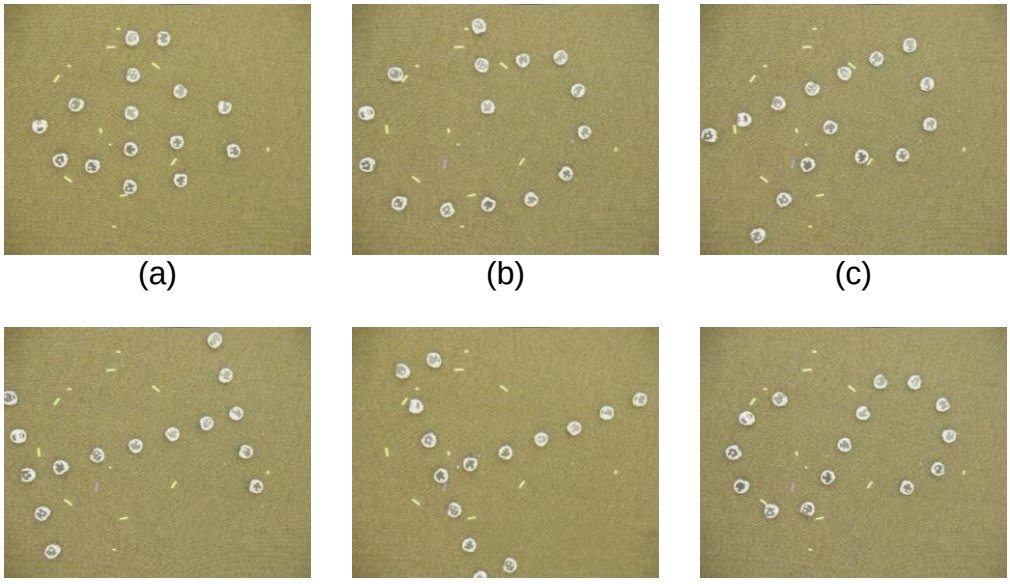
1.6 ESTADO DEL ARTE

Si se parte del hecho que una plataforma robótica es en sí un soporte estructural para los elementos que en ella se depositen, se entenderá entonces que una plataforma robótica no es más que un conjunto de elementos electrónicos dispuestos sobre una estructura física diseñada para cumplir una función específica.

Con esta premisa, es de esperarse que existan en la industria, en la academia y en los diversos saberes de la humanidad, un sinfín de dispositivos que se encuentren dentro del término “plataforma robótica”. Es por eso que, aunque se limite el estado del arte a la comprensión de aquellos robots que han sido diseñados exclusivamente para el estudio de la robótica cooperativa y el control de formación, se hace menester reducir aún más el grupo objetivo, dado que para dicho campo de investigación se han desarrollado todo tipo de plataformas para entornos de trabajo en tierra, agua y aire. Con ese objetivo, se procederá entonces a describir algunas de las plataformas robóticas que han sido concebidas para investigar el control de formación en un ambiente académico bajo un entorno de operación terrestre en el plano cartesiano.

Si se observa la línea de tiempo de la robótica, existen dos fechas significativas para esta investigación. La primera de ellas es en 1986, cuando R. A. Brooks[1] introduce el paradigma del control basado en comportamientos, iniciando con ello la investigación de robots coordinados. La segunda es en el 2001, cuando Yamaguchi[2] (esquema de control distribuido) y Fierro[3] (esquema de control jerárquico) plantearon la solución al problema de coordinación de movimiento con la inclusión de la generación de formaciones específicas de un grupo de robots[4]. Es a partir de las publicaciones de la segunda fecha cuando el control de formaciones toma vigencia y se desarrolla continuamente hasta el presente. Prueba de ello, son los resultados expuestos por Edward A. Macdonald en su trabajo de maestría (2011)[5], en dónde se describe como 15 robots Khepera III ejecutan el algoritmo de asignación multi-robot y control de formación que él desarrolló, formando las letras G, R, I, T y S (por el *Georgia Robotics and Intelligent Systems Laboratory* – GRITSlab). En la Figura 2 puede constatarse la ejecución del algoritmo por parte de los robots Khepera III.

Figura 2. Control de formación de robots Khepera III (GRITSLab). (a) Posición aleatoria inicial. Letras (b) G, (c) R, (d) I, (e) T y (f) S



(a) (b) (c)

(d) (e) (f)

Fuente: GRITSLAB[6]

En la actualidad, el Instituto de Tecnología de Georgia (Estados Unidos) inauguró el *Robotarium*, un laboratorio de robótica cooperativa fundado por la Fundación Nacional para la Ciencia (*NSF*, por sus siglas en inglés) junto a la Oficina de Investigación Naval (*ONR*, por sus siglas en inglés), ambas agencias norteamericanas. Este laboratorio está a disposición de toda la comunidad internacional bajo el concepto de “robots para todos”, esto es, los investigadores de todo el mundo pueden escribir sus propios algoritmos, enviarlos para ser probados y obtener los resultados de los experimentos[7]. Lo anterior se realiza en un ambiente controlado y se hace mediante el empleo de los GRITSBots (Ver Figura 3), plataforma robótica desarrollada por el GRITSLab.

A nivel nacional, son las instituciones universitarias, sean estas públicas o privadas, las entidades que mayormente se encuentran involucradas en la investigación de la robótica cooperativa. Prueba de ello, son los diversos resultados publicados en los

artículos, libros, tesis de posgrado y pregrado que se encuentran a disposición de la comunidad investigativa local e internacional. A continuación, algunos ejemplos por categoría:

Figura 3. Enjambre robótico de 15 GRITSBots

Fuente: PICKEM, Daniel; et al.[8]

- Artículo: Sistemas multiagente robóticos: Revisión de metodologías (2009)[9]. Este artículo expone un estado del arte generalizado de los sistemas multiagente robóticos, describiendo algunas de las metodologías empleadas en esos diseños.
- Libro: Robótica cooperativa. Experiencias de sistemas multiagente (SMA) (2012)[10]. Mediante la integración de temas de mecánica, electrónica e informática, este libro aborda algunos de los aspectos más relevantes del desarrollo de la robótica cooperativa, sus problemas, fundamentos, y funcionamiento.
- Tesis de posgrado: Comparación de estrategias de navegación colaborativa para robótica móvil (2015)[11]. En esta tesis se realizan diversos análisis y comparaciones con respecto a los sistemas multiagentes aplicados a diversos objetivos, resultando en una propuesta de arquitectura y un estudio estadístico a partir de simulaciones.

- Tesis de pregrado: Flotilla de robots para trabajos en robótica cooperativa (2014)[12]. Este trabajo expone los resultados de simular e implementar la estrategia líder-seguidor bajo un sistema de control descentralizado como una solución a la generación de formaciones en un sistema multiagente.

Si bien, para algunos temas de investigación todo culmina con la simulación del objetivo propuesto y su posterior publicación, existen también aquellos que, además de la simulación, requieren de la validación experimental de la teoría, como es el caso del fútbol robótico; rama de la robótica cooperativa que, al igual que el fútbol tradicional, goza de buena aceptación, al punto de tener su propio mundial, la *RoboCupSoccer*[13]. En las últimas versiones de ese certamen (2011 en adelante), Colombia ha sido representada en la categoría de robots pequeños por la Universidad Santo Tomás[14] con su equipo STOx's (Ver Figura 4).

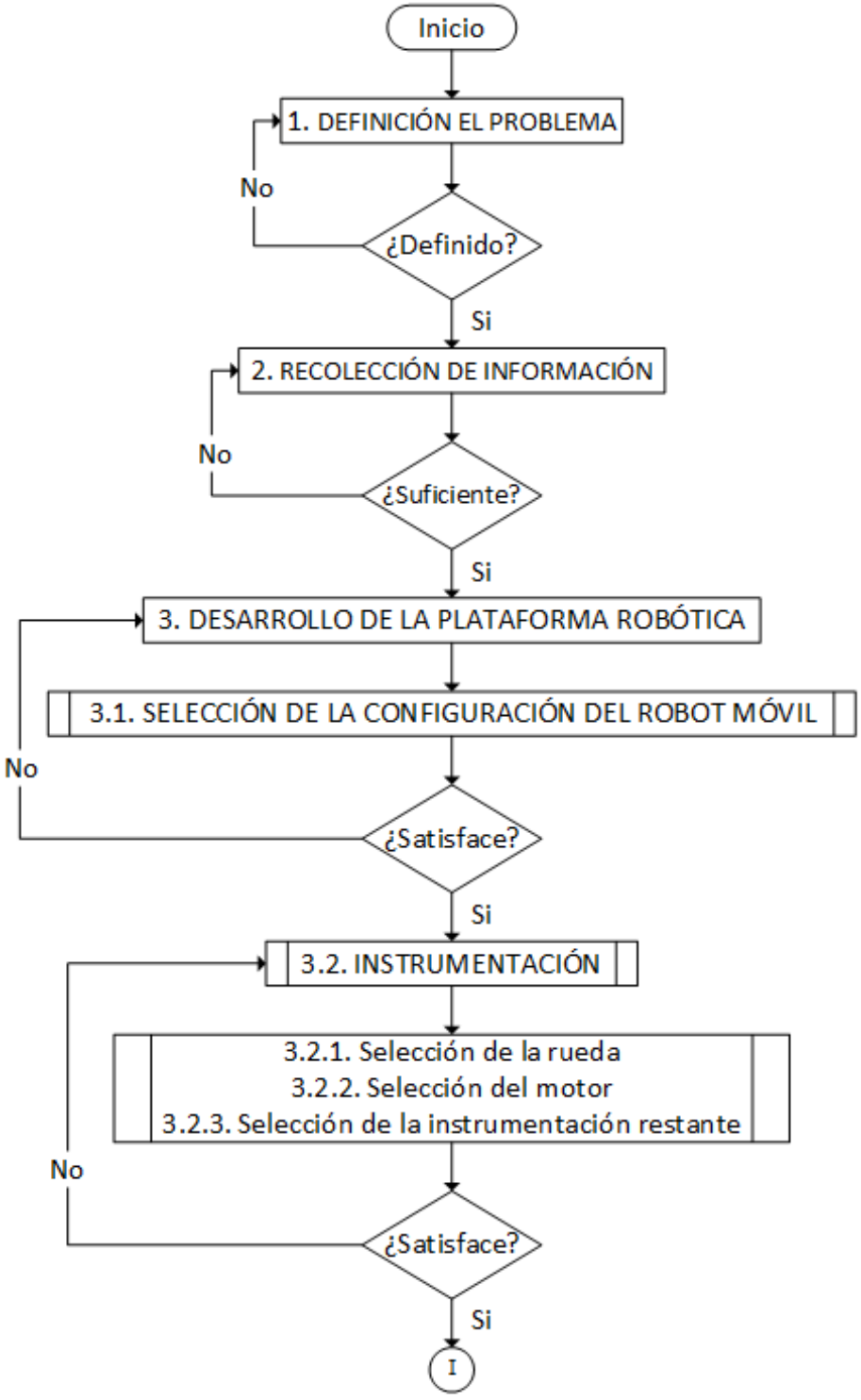
Figura 4. STOx's – Selección de fútbol robótico de la Universidad Santo Tomás (RoboCup Brasil, 2014)

Fuente: IEEE[15]

1.7 DISENO METODOLÓGICO

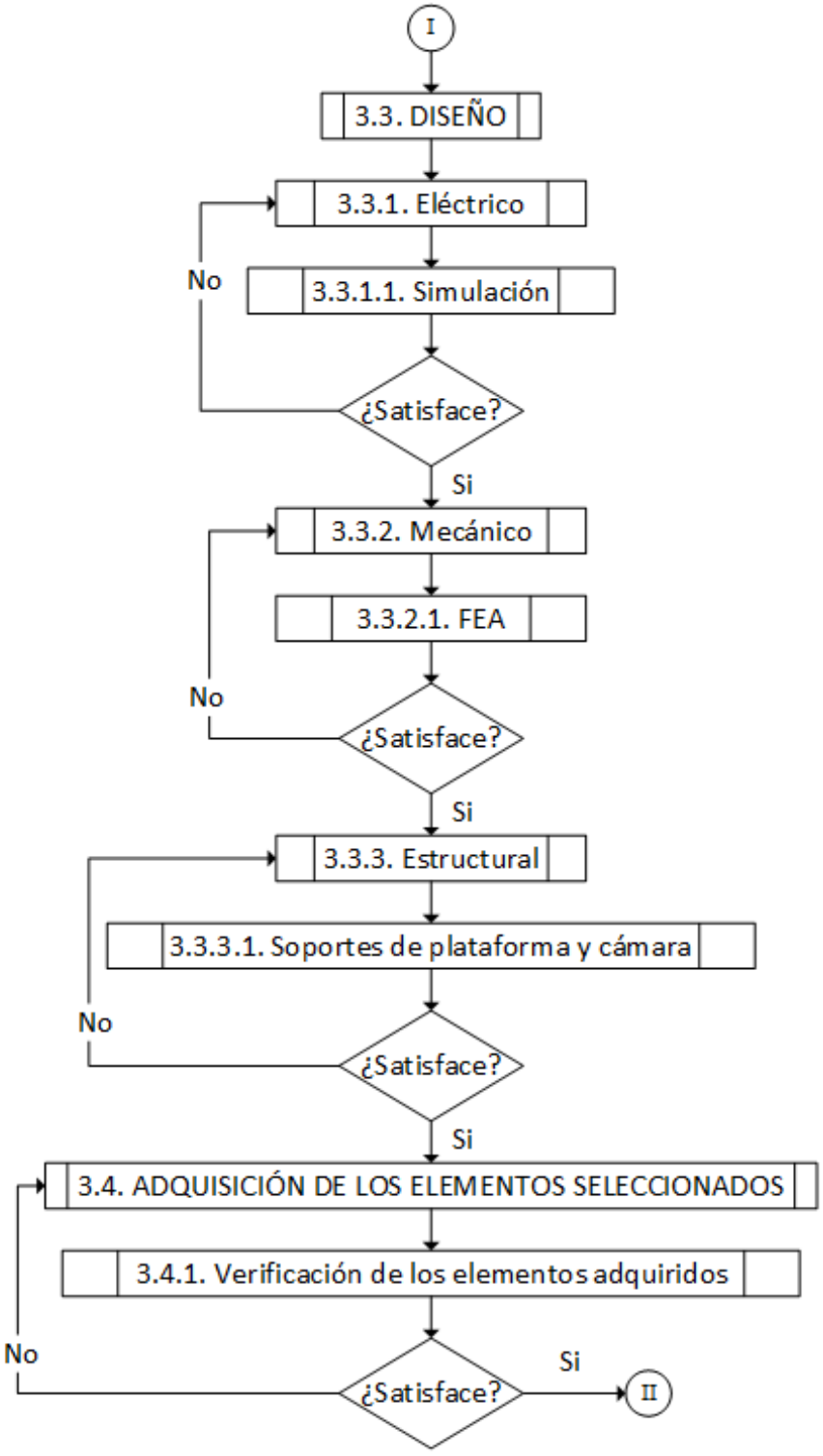
En la Figura 5 se puede observar el diagrama de flujo del diseño metodológico desarrollado en este proyecto de tesis.

Figura 5. Diagrama de flujo del diseño metodológico



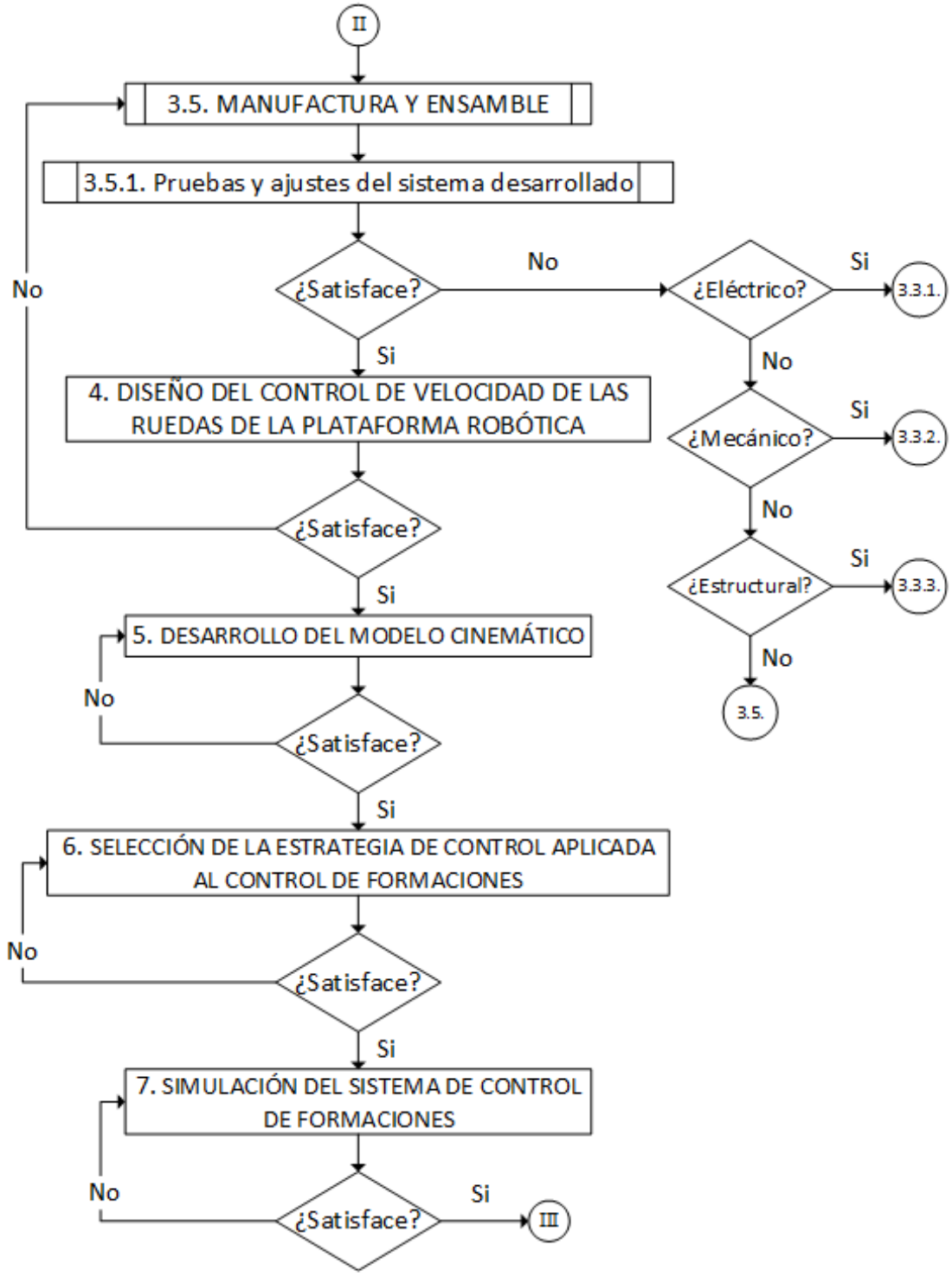
Fuente: Autor

Figura 5. (Continuación)



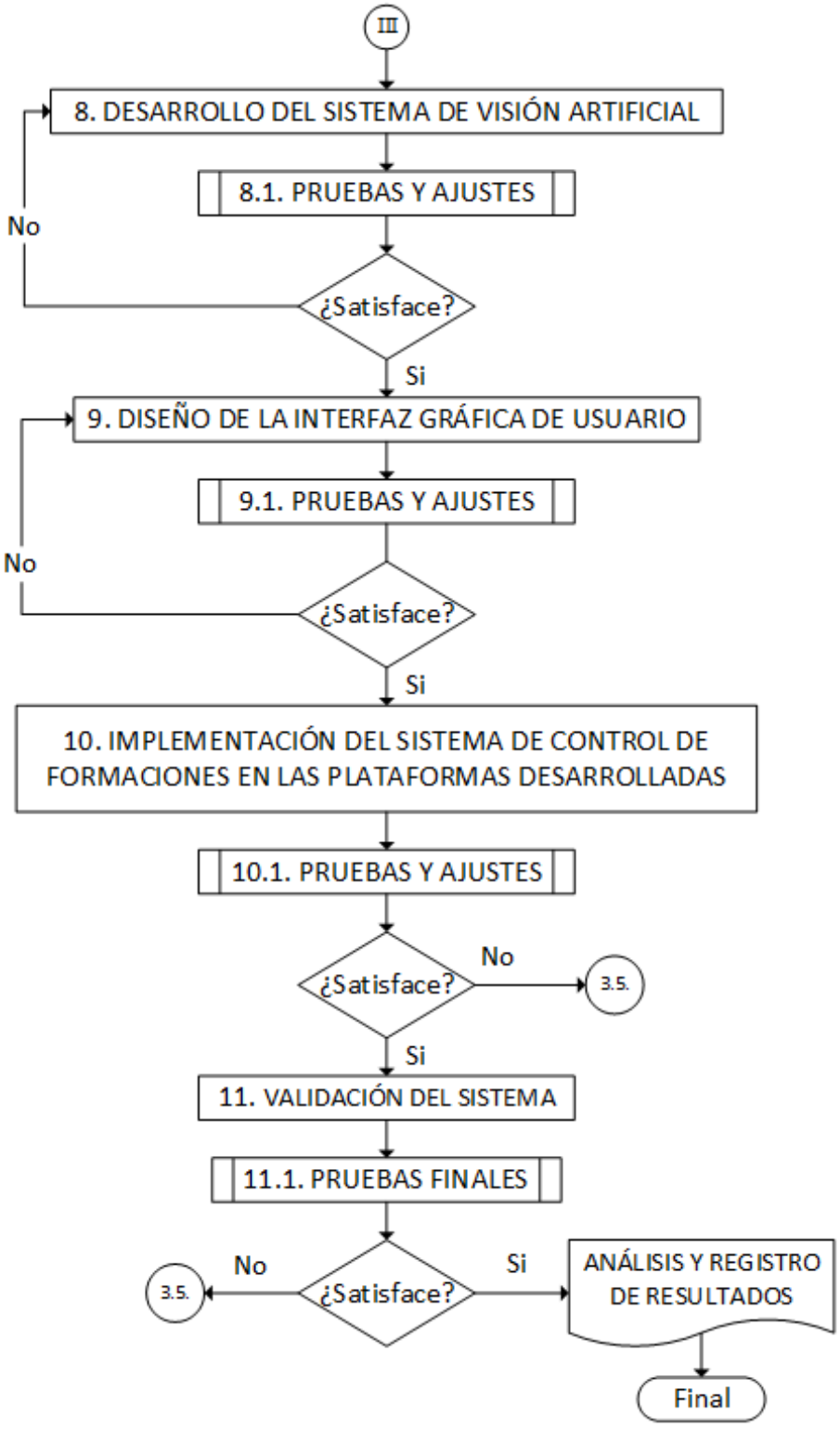
Fuente: Autor

Figura 5. (Continuación)



Fuente: Autor

Figura 5. (Continuación)



Fuente: Autor

2. MARCO TEÓRICO

En el presente capítulo se describen, desde una perspectiva general, los conceptos y fundamentos teóricos requeridos para la realización de los objetivos propuestos.

2.1 CLASIFICACIÓN DE LOS ROBOTS

Antes de clasificar a los robots en grupos y ramificaciones, se hace menester resolver los interrogantes ¿qué representa el término “robot”? y ¿cuál es el alcance de su definición? Preguntas que, resueltas, nos darán una idea global de los elementos que componen ese concepto.

El término robot se deriva de la palabra checa “robota”, que hace referencia a las labores que se hacen de manera forzada. Se da a conocer en 1920 a través de la obra teatral “*Rossumovi Univerzální Roboti (R.U.R.)*” del dramaturgo checo, Karel Čapek[16], en la que desarrolló la idea de fabricar máquinas con forma humana que reemplazaran al hombre en una línea de trabajo industrial. A finales de los 70’s, y después de varias políticas y estandarizaciones que popularizaron el uso de robots en la industria manufacturera, el Instituto Nacional Estadounidense de Estándares (*ANSI*, pos sus siglas en inglés), adoptó los estándares de la Asociación de las Industrias Robóticas (*RIA*, por sus siglas en inglés), definiendo el término como: “Un manipulador reprogramable y multifuncional diseñado para trasladar materiales, piezas, herramientas o aparatos específicos a través de una serie de movimientos programados para la realización de diversas tareas”[17], definición acorde a la tecnología y las necesidades de la época. Sin embargo, hoy día, los alcances de la robótica se han extendido fuera del ámbito industrial (académico, médico, militar, deportivo, recreativo, etc.), desdibujando los límites de esa definición, lo que deriva en una dificultad para lograr un entendimiento común sobre qué objeto puede ser llamado un robot[18]. Por lo tanto, se requiere de una taxonomía que, aunque no

defina el término en sí, nos permita señalar las características que hacen que los robots sean únicos con respecto a otros "objetos", en palabras de Joseph Engelberger, pionero de la robótica industrial, "No puedo definir un robot, pero conozco uno cuando lo veo"[19]. Ahora bien, en semejanza a la definición, existen varias clasificaciones que dependen de diferentes criterios según el autor, ejemplo de ello, son las que se basan en los grados de libertad, la cinemática, el espacio de trabajo, el nivel de autonomía, la aplicación, entre otras tantas pautas de selección, por lo que, en concordancia a lo expresado previamente (subtítulo 1.6 ESTADO DEL ARTE), se procede a describir únicamente la tipología pertinente al objetivo de investigación del presente proyecto, esto es, la clasificación de los robots móviles.

2.1.1 Robots móviles

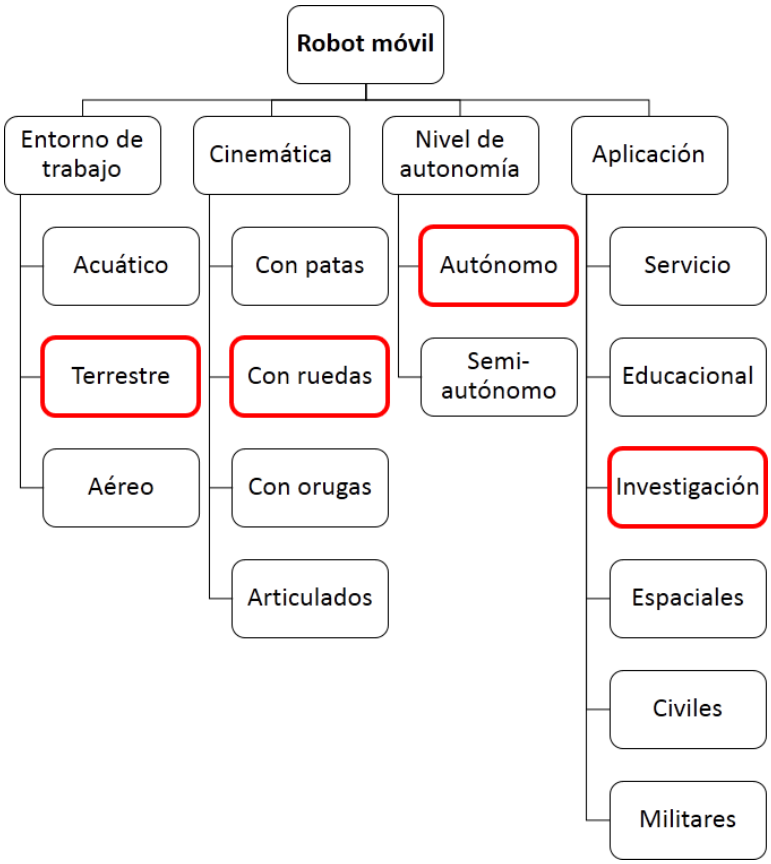
Los robots móviles son aquellos que, gracias a su sistema de locomoción, pueden desplazarse libremente por un área de trabajo sin definir para lograr su objetivo, esto, sin la intervención de un operador humano[20]. Se pueden clasificar mediante los criterios y grupos expuestos en la Figura 6, siendo los bloques rojos nuestros grupos de interés.

2.1.1.1 Robots móviles con ruedas

Esta clasificación está dada por el tipo de configuración en la locomoción del robot (Ver Figura 7), siendo las más importantes[22] la dirección Ackerman, omnidireccional, síncrona, de tipo triciclo y diferencial, esta última, la implementada en este proyecto.

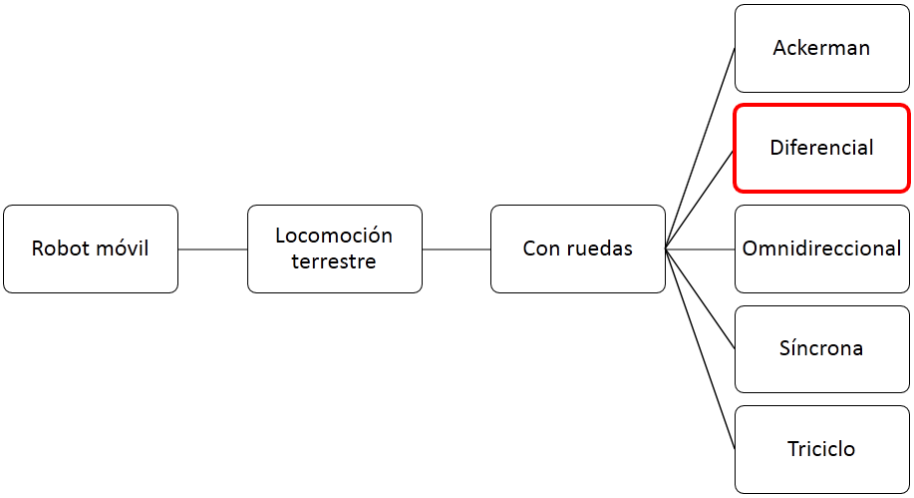
- Dirección diferencial. Esta configuración consta de dos ruedas fijas motrices ubicadas, paralelamente entre sí, a lado y lado de la plataforma. Puede constar de una o dos ruedas libres que le concedan estabilidad (Ver Figura 8). La locomoción de la plataforma se logra controlando independientemente el sentido de giro y la velocidad de cada una de las ruedas (Ver Tabla 1).

Figura 6. Clasificación de los robots móviles



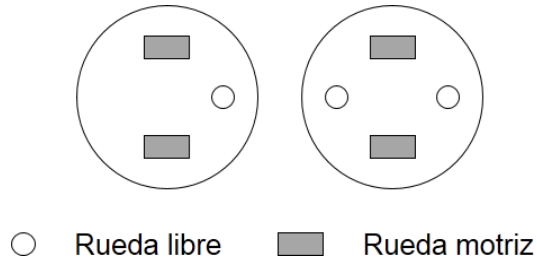
Fuente: ZHANG, Houxiang[21]

Figura 7. Direcciones más empleadas en los robots móviles con ruedas



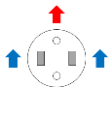
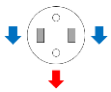
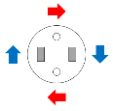
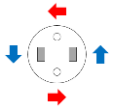
Fuente: TZAFESTAS, Spyros[22]

Figura 8. Dirección diferencial



Fuente: Autor

Tabla 1. Desplazamientos elementales de un robot móvil diferencial

Velocidad angular [$uds.$]	ω_i	ω_d	ω_i	ω_d	ω_i	ω_d	ω_i	ω_d	ω_i	ω_d	ω_i	ω_d	ω_i	ω_d
	0	0	0	1	1	0	1	1	-1	-1	1	-1	-1	1
Desplazamiento														
	Inmóvil		Giro anti-horario		Giro horario		Adelante		Atrás		Giro horario en el propio eje		Giro anti-horario en el propio eje	

Fuente: Autor

2.1.1.2 Cinemática de un robot móvil

Al analizar el movimiento del robot, sin considerar las masas ni las fuerzas que lo producen, se puede establecer el vínculo entre la configuración del robot en su espacio de trabajo, las relaciones entre los parámetros de su geometría y las restricciones impuestas en su trayectoria[23], obteniendo las ecuaciones cinemáticas que nos pueden ayudar con el control del robot.

- Cinemática directa e inversa. Mediante la cinemática directa de un robot móvil podemos obtener una función que nos permita relacionar los grados de libertad

del robot con las coordenadas cartesianas del plano en el que trabaja, es decir, mediante las velocidades angulares de cada rueda de un robot móvil diferencial, podemos determinar su posición y orientación en el plano cartesiano.

Con el modelo cinemático inverso ocurre exactamente lo contrario al directo, esto es, obtenemos las velocidades angulares del robot móvil mediante las coordenadas de su posición y la orientación correspondiente, algo de gran utilidad al momento de asignarle un punto objetivo de desplazamiento al robot.

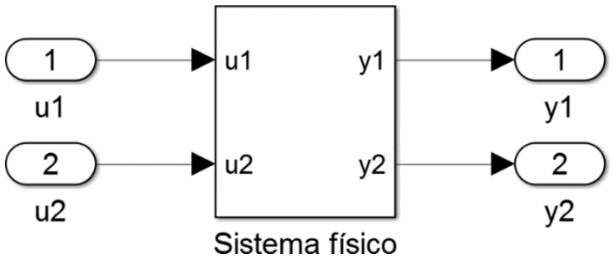
2.2 SISTEMAS DE CONTROL

Para este proyecto se seleccionaron dos sistemas de control, uno de velocidad y otro de formación. Al unirlos, se espera que las velocidades angulares calculadas lleven al robot móvil diferencial al punto específico deseado.

2.2.1 Control de velocidad

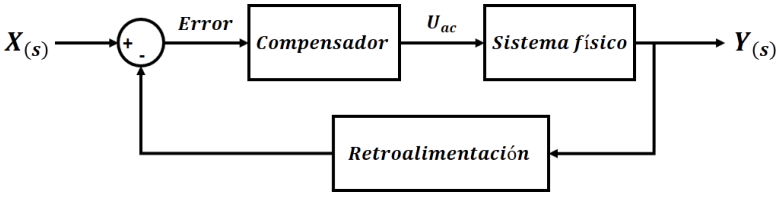
Para lograr el control de la velocidad de las ruedas de un robot móvil es necesario, en primera instancia, determinar el modelo matemático que represente ese sistema o planta, con ello, podemos caracterizar las relaciones entre las variables de entrada (u) y salida (y) del mismo (Ver Figura 9). Con la función de transferencia establecida, se procede entonces a seleccionar y diseñar la estrategia de control para cerrar el lazo, esto es, al retroalimentar la señal controlada ($Y_{(s)}$) y compararla con la entrada de referencia ($X_{(s)}$), se genera una diferencia que el compensador interpreta para enviar una acción de control (U_{ac}) al sistema para corregir ese error[24] (Ver Figura 10), de esa manera, se controla la variable del proceso, en este caso, la velocidad de las ruedas del robot móvil.

Figura 9. Representación de un sistema físico modelado



Fuente: Autor

Figura 10. Sistema de control de lazo cerrado



Fuente: Autor

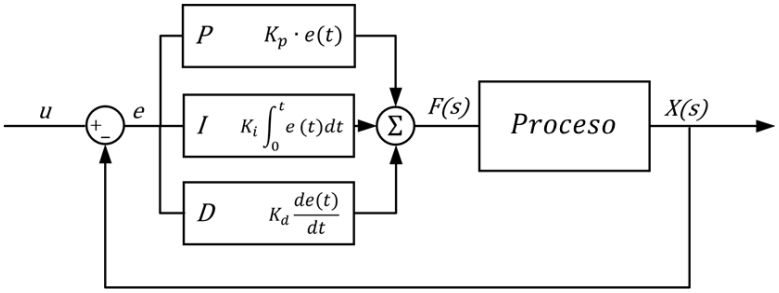
2.2.1.1 Controlador Proporcional Integrativo Derivativo – PID

Dada la simplicidad de la estructura del controlador PID (Ver Figura 11) y la robustez que ha demostrado al ser aplicado en diversas aplicaciones, esta es una de las estrategias de control más aplicadas y difundidas en la industria. Este funciona ajustando la variable de control según el error presente (P – acción de control proporcional), el error acumulado en el pasado (I – acción de control integral) y el error futuro predicho (D – acción de control derivativa). La ecuación de un controlador con esta acción combinada se obtiene mediante[26]:

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt} \quad (1)$$

Donde $u(t)$ es la señal de control, K_p es una ganancia proporcional ajustable, $e(t)$ es la señal de error, T_i el tiempo integral y T_d el tiempo derivativo.

Figura 11. Estructura del controlador PID



Fuente: DEWESoft[25]

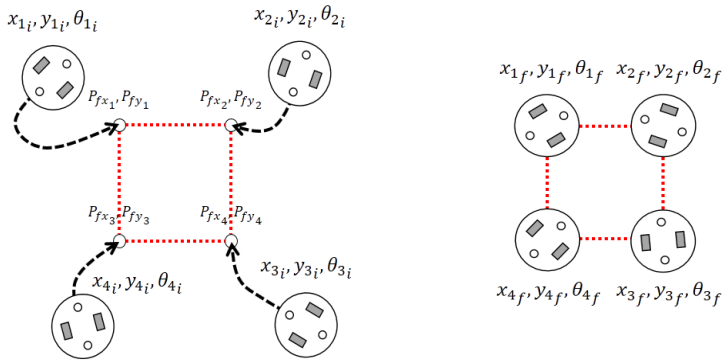
2.2.2 Control de formación

Entiéndase el control de formación como la posibilidad de abordar el problema del desplazamiento de un enjambre de robots móviles partiendo de un enfoque individual a uno simultáneo, esto es, como desde el estudio del desplazamiento individual de las plataformas robóticas, se logran cumplir los objetivos propuestos de un sistema multiagente. Para lograr ese propósito se debe, en primera instancia, caracterizar las plataformas robóticas para determinar si el grupo es homogéneo o heterogéneo y, en base a ello, establecer las leyes de desplazamiento según sean sus características. Se procede a asignar un rol a cada plataforma respecto a las demás, esto le permitirá ubicarse en el grupo según sea su tarea a desempeñar. Posterior a ello, se implementa la estrategia a analizar, siendo las estructuras virtuales, los comportamientos grupales y la configuración líder-seguidor, las principales propuestas para los sistemas multiagente[26].

Para el presente trabajo, se optó por un control de formación visual enfocado a la representación de formas geométricas bajo un esquema de control centralizado. Su funcionamiento se puede observar en la Figura 12. Consiste en seleccionar la figura a realizar, dividirla en puntos coordenados conforme a la cantidad de elementos en el espacio de trabajo, según sean los datos de orientación y posición de cada plataforma, adquiridos por una única cámara web cuyo plano de imagen es paralelo al plano de movimiento de los robots, el nodo central planea una única trayectoria

para cada plataforma hacia cada uno de los puntos y verifica que cada robot siga su respectiva trayectoria de manera correcta.

Figura 12. Representación de un cuadrado mediante el control de formación visual seleccionado. (a) Posición aleatoria inicial. (b) Posición final



(a) (b)

Fuente: Autor

2.3 VISIÓN ARTIFICIAL

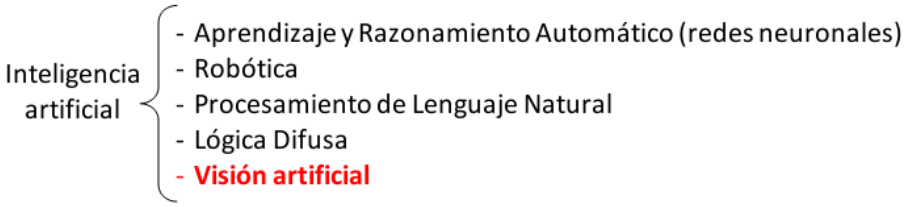
Mediante la implementación de una de las ramas de la inteligencia artificial (Ver Figura 13), la visión artificial, se emulará en la computadora la percepción visual de los seres vivos, captando imágenes de las cuales se extraerá la información pertinente para el posicionamiento y la orientación de los robots como resultado de la aplicación de algoritmos de segmentación de color, operaciones morfológicas y filtrado.

2.3.1 Espacios o modelos de color

Un espacio o modelo de color, es la especificación de un sistema de coordenadas tridimensional y de un subespacio de este sistema en el que cada color queda representado por un único punto. Los espacios de color más empleados en el

procesamiento de imágenes son: YIQ, CMY, HSI, YCbCr, RGB y HSV[29], siendo los dos últimos, los modelos implementados en este proyecto.

Figura 13. Ramas de la inteligencia artificial



Fuente: SABIA[28]

2.3.1.1 Modelo de color RGB

Este modelo se fundamenta en el sistema de coordenadas rectangulares, siendo el hexaedro mostrado en la Figura 14 el subespacio de color de interés. Los colores de este espacio son puntos dentro del hexaedro definidos por los vectores desde el origen, siendo la escala de grises la diagonal entre el color negro (0,0,0) y el blanco (255,255,255). Las imágenes en este modelo consisten en tres planos de imagen independientes, uno por cada color primario: R (Red/Rojo), G (Green/Verde) y B (Blue/Azul), cuando llegan a un monitor RGB, estas tres imágenes se combinan en la pantalla fosforescente para producir una imagen en color compuesta[30]. Para este proyecto, esas imágenes compuestas son adquiridas en este modelo de color mediante una cámara web (Ver Figura 29a, HD Pro Webcam C920), esto, debido a la configuración de fábrica de dicha cámara.

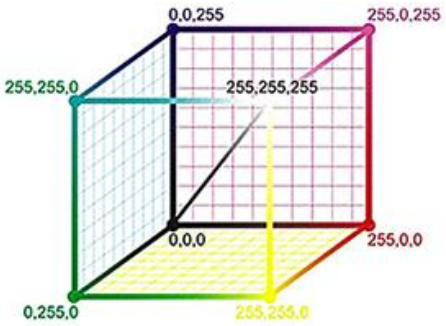
2.3.1.2 Modelo de color HSV

Los parámetros del modelo de color HSV son la tonalidad o matiz (Hue/H), la saturación (Saturation/S) y el valor o brillo (Value/V). Se representa mediante un cono de color (Ver Figura 15), donde el eje vertical representa al valor, la distancia horizontal tomando como referencia el eje V corresponde a la saturación, mientras que el ángulo que se establece tomando como punto de rotación el eje V define la

tonalidad. El punto que corresponde al negro se ubica en el pico del cono, mientras que el punto que corresponde al blanco se encuentra localizado en la parte central de la base. Los tres colores básicos rojo, verde y azul, y sus respectivas combinaciones, amarillo, cian y magenta se encuentran distribuidos en la base de la pirámide[32].

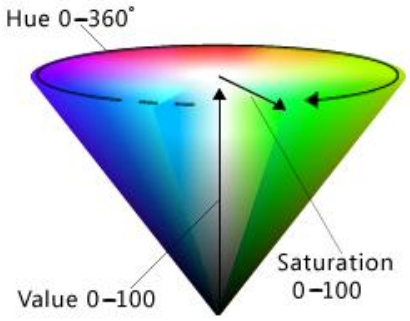
Dado que la discriminación de objetos por colores en el modelo RGB se dificulta debido a la intensidad de luz que incide sobre ellos, en este proyecto se opta por convertir las componentes R, G y B de la imagen digital capturada por la cámara web, a los parámetros (matiz, saturación y brillo) del modelo HSV, con los que la descripción de la imagen adquirida se hace más relevante.

Figura 14. Hexaedro representativo del modelo de color RGB



Fuente: GÓMEZ, Aure[31]

Figura 15. Cono representativo del modelo de color HSV



Fuente: TraumaBot[33]

2.3.2 Filtros espaciales

Responden a la siguiente ecuación:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{2}$$

Donde $f(x+s, y+t)$ es el valor de los píxeles del bloque seleccionado y $w(s,t)$ los coeficientes que se aplicarán al bloque (máscara).

Los filtros espaciales modifican la contribución de determinados rangos de frecuencias de una imagen a través de la convolución. El término espacial se refiere a que el filtro se aplica directamente a la imagen y no a una transformada de la misma. Pueden clasificarse basándose en su linealidad (lineales y no lineales) o según las frecuencias que dejen pasar (paso bajo, alto y banda)[34].

En este proyecto se hace uso de los filtros espaciales para resaltar o suprimir, de forma selectiva, información contenida en las imágenes adquiridas por la cámara web, con el fin de destacar algunos elementos en esas imágenes digitales.

2.3.2.1 Filtro gaussiano

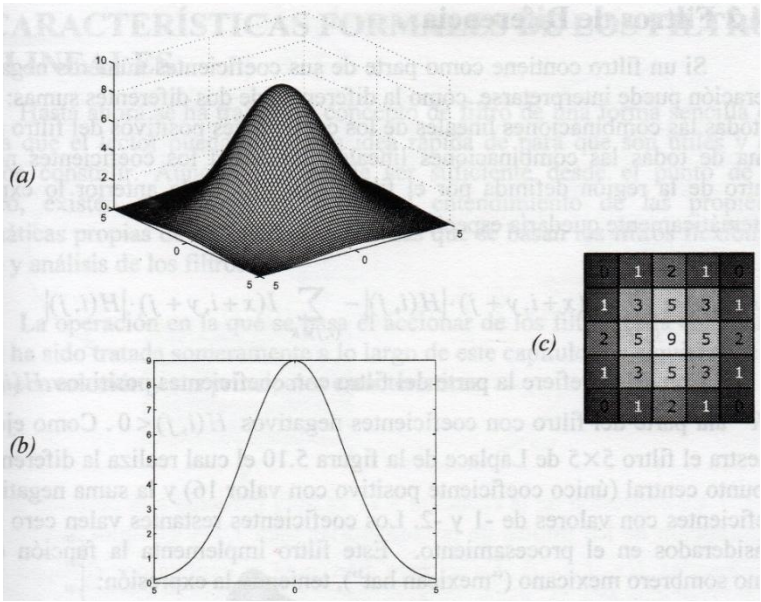
El filtro Gaussiano corresponde a una función de Gauss bidimensional y discreta tal como:

$$G_\sigma(r) = \mathrm{e}^{-\frac{r^2}{2\sigma^2}} \qquad \text{ó} \qquad G_\sigma(x,y) = \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3}$$

Donde la desviación estándar σ representa el radio de cobertura de la función de Gauss tal y como se muestra en la Figura 16. El elemento central del filtro representa

el peso máximo que participa en la combinación lineal de la operación, mientras que los valores de los demás coeficientes tienen menor influencia conforme estos se alejan del centro del filtro. Este filtro es isotrópico (invariante en la rotación) y permite suavizar las regiones en donde los valores de intensidad son homogéneos sin diluir los bordes de la imagen[35]. Se aplica en este trabajo con el fin de reducir el ruido digital que pueda ser producido por la cámara web al momento de adquirir las imágenes.

Figura 16. El filtro Gauss para suavizado. (a) Función tridimensional, (b) bidimensional y (c) mascarilla que implementa el filtro



Fuente: CUEVAS, Erik; ZALDÍVAR, Daniel; PÉREZ, Marco[35]

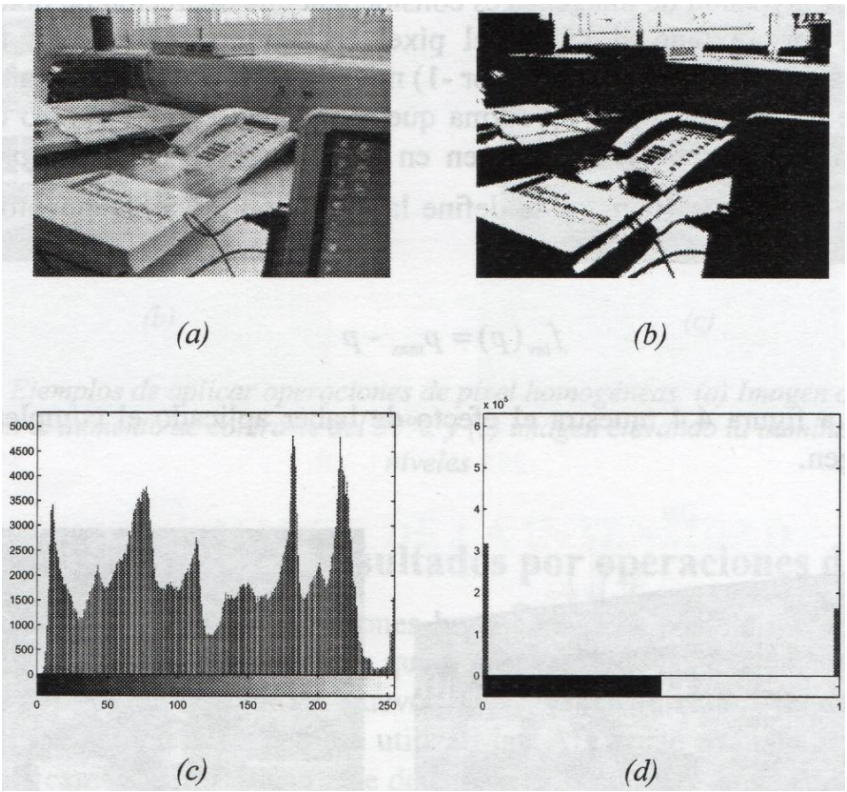
2.3.3 Segmentación por umbral (*threshold*)

La segmentación por la utilización de un umbral o umbralización, puede ser considerada como una forma especial de cuantificación en la cual los pixeles de la imagen son divididos en dos clases, dependiendo de un umbral predefinido P_{th}. Todos los pixeles de la imagen asumen dos diferentes valores p_0 o p_1 dependiendo de la relación que guarden con el umbral, definido formalmente como:

$$f_{th}(p) = \begin{cases} p_0 & si\ p < p_{th} \\ p_1 & si\ p \geq p_{th} \end{cases} \quad (4)$$

Donde $0 < p_{th} < p_{max}$. Una aplicación frecuente de esta operación es la binarización de una imagen a escala de grises considerando a $p_0 = 0$ y $p_1 = 1$ (Ver Figura 17)[36].

Figura 17. Segmentación por umbral de una imagen a escala de grises. (a) Imagen original. (b) Imagen binaria. (c) Histograma de la imagen original. (d) Histograma de la imagen binaria



Fuente: CUEVAS, Erik; ZALDÍVAR, Daniel; PÉREZ, Marco[36]

Dado que las plataformas robóticas de este proyecto se identifican por medio de colores, es a través de la umbralización que el sistema de visión artificial puede diferenciar unas de otras.

2.3.4 Transformaciones morfológicas

Las transformaciones morfológicas son algunas operaciones sencillas basadas en la forma de la imagen. Normalmente se realiza en imágenes binarias. Estas transformaciones requieren de dos entradas, la imagen original y el elemento estructurador o kernel, quién decide la naturaleza de la operación[36]. Estas transformaciones nos permitirán reducir algunas imperfecciones que se puedan encontrar después de umbralizar la imagen adquirida por la cámara web.

2.3.4.1 Erosión

Con esta operación todos los píxeles cercanos al borde del objeto en la imagen se descartarán dependiendo del tamaño del kernel, esto es, el grosor o el tamaño del objeto (región blanca) en el primer plano disminuirá según sea la matriz aplicada. Es útil para eliminar pequeños ruidos, separar dos objetos conectados, entre otras aplicaciones[38]. En la Figura 18 se observa el efecto de aplicar un kernel completo de unos de 5x5 (matriz cuadrada de unos) a una imagen binaria.

Figura 18. Ejemplo de erosión. (a) Imagen binaria original. (b) Resultado

(a) (b)

Fuente: Autor

2.3.4.2 Dilatación

Esta operación es justo lo opuesto a la erosión, por ende, aumenta la región blanca en la imagen. También depende del tamaño del kernel aplicado. La dilatación es útil

para unir partes abiertas de un objeto[39]. En la Figura 19 se observa el efecto de la dilatación usando el mismo kernel de la Figura 18.

Figura 19. Ejemplo de dilatación. (a) Imagen binaria original. (b) Resultado

(a) (b)

Fuente: Autor

2.3.4.3 Apertura

La operación de apertura es el resultado de aplicar la erosión seguida de la dilatación con el propósito de eliminar los ruidos blancos. Un ejemplo de su uso se observa en la Figura 20, en donde se aplica un kernel completo de unos de 3x3 a una imagen binaria con ruido blanco.

Figura 20. Ejemplo de apertura. (a) Imagen binaria con ruido blanco. (b) Resultado

(a) (b)

Fuente: Autor

2.3.4.4 Cierre

La operación de cierre es la inversa de la apertura, esto es, aplicar primero la dilatación seguidamente de la erosión. Es útil para cerrar pequeños agujeros dentro de los objetos de la imagen como se puede observar en la Figura 21, en donde se aplica un kernel completo de unos de 3x3.

Figura 21. Ejemplo de cierre. (a) Imagen binaria con orificios en el objeto. (b) Resultado

(a) (b)

Fuente: Autor

2.3.5 Características geométricas

Un objeto O de una imagen binaria puede ser interpretado como una distribución de puntos de valor uno $\boldsymbol{x_i} = (x, y)$ en una rejilla bidimensional, esto es:

$$O = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \dots, \boldsymbol{x_N}\} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_N, y_N)\} \quad (5)$$

Para el cálculo de la mayoría de las características geométricas, un objeto es considerado como un conjunto de unos que se encuentran agrupados bajo un criterio de vecindad[40]. Es decir, los cálculos se realizarán en base a la zona blanca

resultante de la umbralización de la imagen adquirida, siendo esta zona, la representación de los identificadores de cada plataforma robótica.

2.3.5.1 Área

El área de un objeto O puede ser calculada sencillamente mediante el número de pixeles que lo conforman. Esto es:

$$Área(O) = N = |O| \tag{6}$$

Cuando el objeto al cual se le desea determinar su área no se encuentra descrito por un conjunto agrupado de pixeles, sino por el contorno que lo rodea es posible aproximar el área del contorno exterior cerrado (sólo si esta no contiene contornos internos), mediante la aplicación de la ecuación:

$$Área(O) = \frac{1}{2} \cdot \left| \sum_{i=1}^{M} \left(x_i \cdot y_{[(i+1) mod M]} - x_{[(i+1) mod M]} \cdot y_i \right) \right| \tag{7}$$

Dónde x_i y y_i son las coordenadas de los puntos $\boldsymbol{x_1, \ldots, x_M}$ que forman parte del contorno cerrado que conforma al objeto. El contorno descrito por estos puntos es descrito mediante el código de cadena $C_C = (c_1, c_2, \ldots, c_M)$[41].

En vista que para la plataforma robótica se hace uso de dos circunferencias del mismo color para establecer la posición (diámetro mayor) y la orientación (diámetro menor) de la misma en el espacio de trabajo, se emplea esta característica geométrica para esclarecer, según sea el área calculada, a qué parámetro corresponde cada circunferencia.

2.3.6 Características estáticas de forma

Estas características son el centroide y los momentos de los objetos dentro de una imagen binaria. Para su cálculo se considera al objeto como una distribución de coordenadas de puntos de dos dimensiones, no obstante, también se pueden calcular en nubes de puntos no conectados[42]. Mediante su uso, se obtienen las coordenadas en píxeles de los objetos en la imagen umbralizada, esto es, las coordenadas cartesianas de las circunferencias de la plataforma robótica, estableciendo con ello, la posición y orientación del robot en el espacio de trabajo.

2.3.6.1 Centroide

El centroide o centro de masas de un objeto es el punto en donde por su geometría se encuentra concentrada la masa del objeto. El centroide $\bar{\boldsymbol{x}} = (\bar{x}, \bar{y})$ de un objeto binario (no necesariamente agrupados) se calcula como el punto medio aritmético de las coordenadas en la dirección x y y, tal que[43]:

$$\bar{x} = \frac{1}{\acute{A}rea(O)} \sum_{(x,y)\in O} x \tag{8}$$

$$\bar{y} = \frac{1}{\acute{A}rea(O)} \sum_{(x,y)\in O} y \tag{9}$$

2.3.6.2 Momentos

El centroide o punto central de un objeto definido en las ecuaciones (8) y (9), puede ser considerado como un caso especial de un concepto general de la estadística, los llamados momentos. Estos pueden ser expresados a través de:

$$m_{pq} = \sum_{(x,y)\in O} I(x,y) \cdot x^p \cdot y^q \tag{10}$$

Que representa el momento de orden p, q para una función discreta $I(x, y) \in \{0,1\}$, que puede representar a una imagen a escala de grises. Para el caso especial de imágenes binarias $I(x, y) \in \{0,1\}$, donde los pixeles pertenecientes al objeto tienen el valor de 1, los momentos pueden se calculados utilizando:

$$m_{pq} = \sum_{(x,y)\in O} x^p \cdot y^q \tag{11}$$

De esta manera el área de un objeto en una imagen binaria puede ser considerada como el momento de orden cero, tal que:

$$m_{00} = \sum_{(x,y)\in O} x^0 \cdot y^0 = \sum_{(x,y)\in O} 1 \cdot 1 = \acute{A}rea(O) \tag{12}$$

Por lo tanto el centroide de un objeto $\bar{\boldsymbol{x}} = (\bar{x}, \bar{y})$ puede ser considerado como:

$$\bar{x} = \frac{1}{\acute{A}rea(O)} \sum_{(x,y)\in O} x^1 \cdot y^0 = \frac{m_{10}(O)}{m_{00}(O)} \tag{13}$$

$$\bar{y} = \frac{1}{\acute{A}rea(O)} \sum_{(x,y)\in O} x^0 \cdot y^1 = \frac{m_{01}(O)}{m_{00}(O)} \tag{14}$$

Estos momentos representan características físicas concretas de objetos[43].

3. DESARROLLO DE LA PLATAFORMA ROBÓTICA

En el presente capítulo se desarrolla el diseño de la plataforma robótica junto a los diseños de los elementos estructurales (plataforma de trabajo y soporte de la cámara web) necesarios para su operación, tomando como directrices las siguientes hipótesis simplificadoras:

- El trabajo se realiza en un entorno controlado.
- La plataforma robótica se mueve sobre una superficie plana.
- La plataforma robótica, en conjunto, se comporta como un cuerpo rígido.
- Las ruedas se mueven con rodadura pura.
- Como máximo hay un eje de dirección por rueda.
- Los ejes de la dirección son perpendiculares al suelo.
- El plano de imagen es paralelo al plano de movimiento de la plataforma robótica.

3.1 SELECCIÓN DE LA CONFIGURACIÓN DEL ROBOT MÓVIL

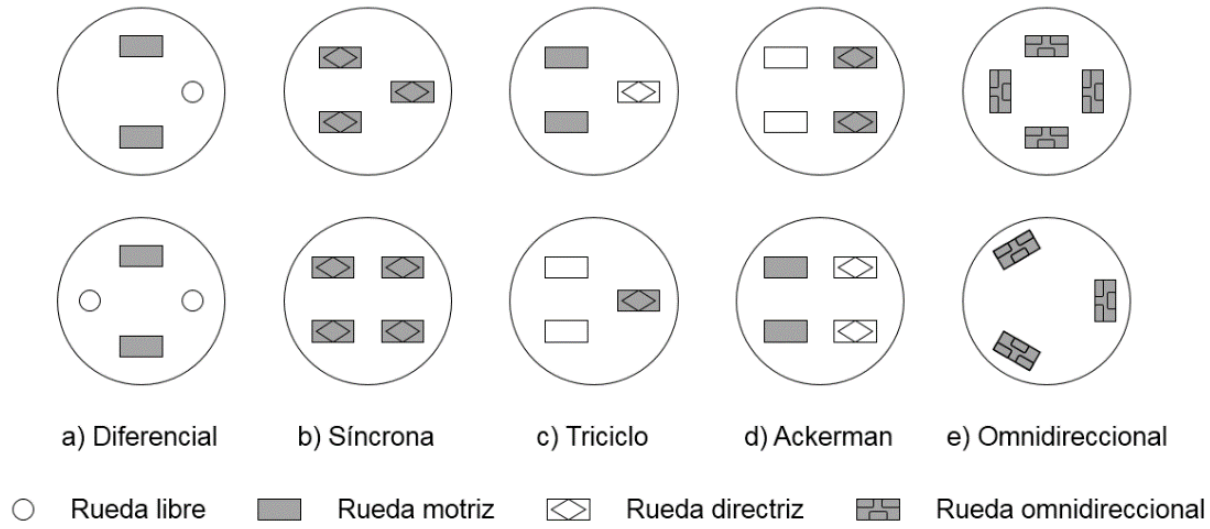
Si bien, los sistemas de locomoción de los robots móviles abarcan una amplia gama de la robótica, este proyecto se centrará en la locomoción con ruedas. Esto dada la facilidad de implementación de este sistema en comparación a otros, tales como los articulados, en los que, al aumentar la cantidad de actuadores, se incrementa el peso de la estructura, el procesamiento, la demanda energética entre otras consideraciones.

Como se mencionó anteriormente en el marco teórico (subtítulo 2.1.1.1 Robots móviles con ruedas), dado que en relación a la locomoción con ruedas existen distintas configuraciones típicamente empleadas en robótica móvil (diferencial, triciclo, Ackerman, sincronizada y omnidireccional), se hace menester identificar

cuál de ellas es la indicada para este proyecto. Con ese fin, se presenta el ANEXO 1 (COMPARATIVA DE LOS SISTEMA DE LOCOMOCIÓN) con el que se establece que el sistema de locomoción que mejor se adapta a las necesidades del proyecto es el DIFERENCIAL.

Para facilitar la interpretación de la información del anexo, se proporciona la Figura 22 en la cual se muestran de forma gráfica las configuraciones de locomoción antes mencionadas. De esta gráfica se puede deducir que la configuración diferencial se presenta como la más sencilla de todas. Si bien esta se parece a la configuración triciclo, esta varía en la implementación de la rueda de soporte. Mientras que en la configuración triciclo la rueda de soporte es la rueda directriz, la cual permite la dirección deseada del robot, en la configuración diferencial la rueda libre o “rueda loca”, solo cumple la función de soporte, obteniendo el cambio de dirección de la diferencia de velocidades angulares de sus dos ruedas motrices.

Figura 22. Sistemas de locomoción por ruedas



Fuente: Autor

3.2 INSTRUMENTACIÓN

Establecido el sistema de locomoción, se procede entonces con la selección de los elementos mecánicos y electrónicos que, en conjunto, conformarán la plataforma robótica.

3.2.1 Selección de la rueda

Para seleccionar la rueda que mejor se adaptara al sistema de locomoción diferencial, se empleó el ANEXO 2 (COMPARATIVA DE LAS RUEDAS), que establece la rueda Pololu de 32×7 [mm] como mejor opción (Ver Figura 23).

Figura 23. Rueda Pololu de 32×7 [mm]

Fuente: TDROBÓTICA[44]

3.2.2 Selección del motor

Para este paso del diseño de la plataforma robótica se requieren resolver los siguientes interrogantes:

- ¿Cuál es la masa total de la plataforma?
- ¿Cuál es el torque mínimo necesario para mover esa masa?

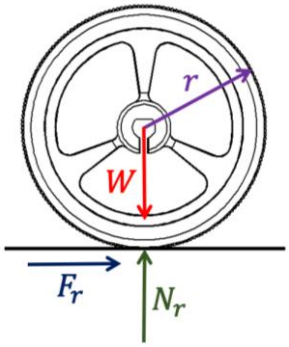
Para el primer interrogante se plantea una solución arbitraria de 1000 [g], esto es:

$$m = \ 1000\ [g] = 1\ [kg] \tag{15}$$

Para el segundo interrogante, se emplean las fuerzas de la Figura 24, de modo que:

$$T_m = F_r \cdot \mathrm{r} \tag{16}$$

Figura 24. Diagrama de fuerzas presentes en la rueda



Fuente: Autor

Donde T_m es el torque a vencer, F_r es la fuerza de fricción y r es el radio de la rueda. Las otras fuerzas en la gráfica son el peso total de la plataforma robótica (W) y la normal a ese peso (N_r). Del ANEXO 2 se sabe que $\mathrm{r} = 0.016\ [m]$. Para la fricción se tiene que:

$$F_r = \mu_s \cdot N_r \tag{17}$$

Donde μ_s es el coeficiente de fricción estática entre la superficie y la rueda; para nuestro caso, madera MDF (siglas en inglés de *Medium Density Fibreboard*) y silicona, respectivamente. En vista que no se encuentra un valor teórico para este coeficiente de fricción, se procede a hallarlo con ayuda del montaje experimental de la Figura 25, de donde:

$$\sum F_x = 0$$

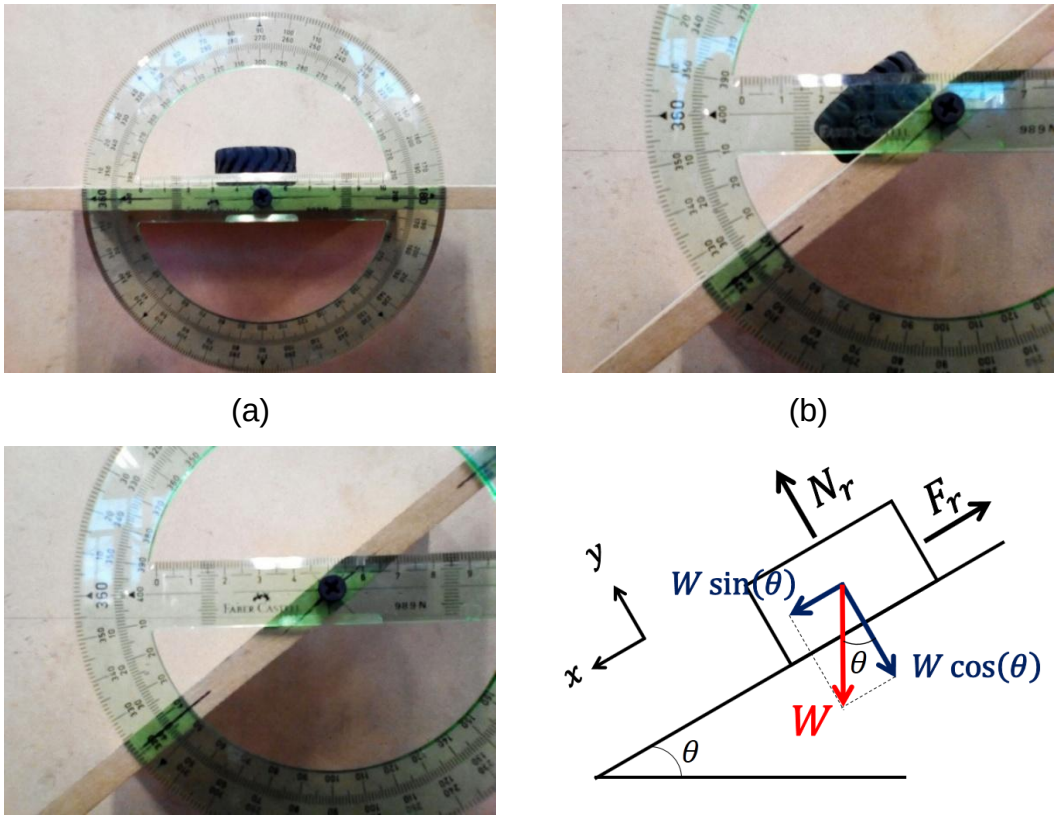
$$W \sin(\theta) - F_r = 0 \tag{18}$$

$$\sum F_y = 0$$

$$N_r - W \cos(\theta) = 0$$

$$W = \frac{N_r}{\cos(\theta)} \tag{19}$$

Figura 25. Montaje experimental para hallar el coeficiente de fricción entre la rueda y la superficie de trabajo de la plataforma robótica. (a) Posición inicial horizontal. (b) Posición de deslizamiento inminente ($\theta = 43.5°$)*. (c) Posición después del deslizamiento ($\theta = 44°$). (d) Diagrama de fuerzas



* Ángulos alternos internos

Fuente: Autor

Reemplazando (22) en (21):

$$\frac{N_r \cdot \sin(\theta)}{\cos(\theta)} - F_r = 0$$
$$F_r = N_r \cdot \tan(\theta)$$
$$\mu_s \cdot N_r = N_r \cdot \tan(\theta)$$
$$\mu_s = \tan(\theta)$$
$$\mu_s = \tan(43.5°)$$
$$\mu_s = 0.948965 \quad (20)$$

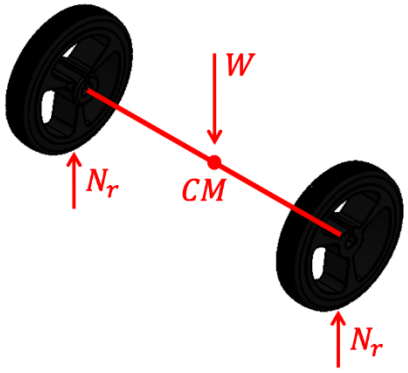
Para el peso y la normal, según la Figura 26, se tendrá que:

$$W = m \cdot g$$
$$W = 1\ [kg] \cdot 9.81\ [m/s^2]$$
$$W = 9.81\ [N] \quad (21)$$
$$\sum F_y = 0$$
$$2N_r - W = 0$$
$$N_r = \frac{W}{2} = \frac{9.81\ [N]}{2}$$
$$N_r = 4.905\ [N] \quad (22)$$

Reemplazando (17) en (16) y los valores de μ_s y N_r en (17):

$$T_m = \mu_s \cdot N_r \cdot \mathrm{r} = (0.948965) \cdot (4.905\ [N]) \cdot (0.016\ [m])$$
$$T_m = 0.0744752\ [N \cdot m] = 0.75943553\ [kg \cdot cm] \quad (23)$$

Figura 26. Diagrama de fuerzas del peso de la plataforma



Fuente: Autor

Asumiendo un factor de seguridad arbitrario de 3 ($n = 3$), se tendrá que el torque mínimo necesario para mover la masa de la plataforma robótica es de:

$$T_m = (0.75943553\ [kg \cdot cm]) \cdot (3)$$
$$T_m = 2.27830659\ [kg \cdot cm] \quad (24)$$

Con el valor de T_m establecido, se emplea el ANEXO 3 (COMPARATIVA DE LOS DISPOSITIVOS DE LOCOMOCIÓN) para establecer que el motor a incorporar en la estructura de la plataforma robótica es el micromotor Pololu con eje extendido de la Figura 27.

Figura 27. Micromotor Pololu con eje extendido

Fuente: TDROBÓTICA[45]

3.2.3 Selección de la instrumentación restante

Para la selección de los elementos de la Figura 28 se emplean los anexos enlistados en la Tabla 2. El método de selección aplicado en todas las comparativas se explica en el ANEXO 10 (CÁLCULOS EMPLEADOS EN LAS COMPARATIVAS).

Tabla 2. Selección de la instrumentación restante

Elemento	Anexo
Encoder	ANEXO 4. COMPARATIVA DE LOS ENCODERS
Sensor	ANEXO 5. COMPARATIVA DE LOS SENSORES
Sistema embebido	ANEXO 6. COMPARATIVA DE LOS SISTEMAS EMBEBIDOS
Sistema de comunicación	ANEXO 7. COMPARATIVA DE LOS SISTEMAS DE COMUNICACIÓN
Puente H	ANEXO 8. COMPARATIVA DE LOS PUENTE H
Batería	ANEXO 9. COMPARATIVA DE LAS BATERÍAS

Fuente: Autor

El criterio de selección de los elementos de la Figura 29 fueron los buenos resultados de su implementación en proyectos previamente realizados con esos mismos componentes.

Figura 28. Selección de la instrumentación restante. (a) Encoder magnético Pololu. (b) Sensor ultrasónico HC-SR04. (c) Arduino Mega 2560 R3. (d) Módulo L298N. (e) Módulo de radio frecuencia NRF24L01. (f) Batería LiPo 2200 mAh 11.1V

(a) (b) (c)

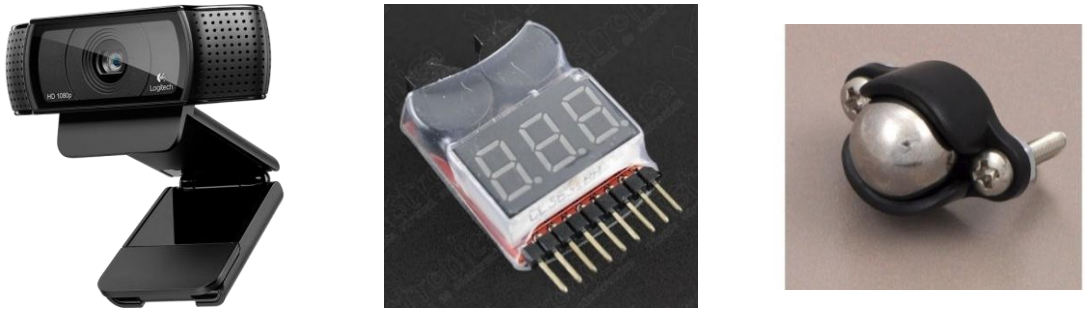
Fuente: TDROBÓTICA[46], [47], [48]

Figura 28. (Continuación)

(d) (e) (f)

Fuente: TDROBÓTICA[49], [50], [51]

Figura 29. Elementos seleccionados previamente. (a) HD Pro Webcam C920. (b) Indicador de tensión de Baterías LiPo. (c) “Rueda loca” Pololu con rueda metálica de 3/8”



(a) (b) (c)

Fuente: LOGITECH[52], VISTRÓNICA[53], POLOLU[54]

3.3 DISEÑO DE LA PLATAFORMA ROBÓTICA

Seleccionada la instrumentación requerida para el funcionamiento de la plataforma robótica, se procede entonces con los diseños de la estructura de soporte que ha de contener dichos elementos, del circuito electrónico que los conecta, del espacio de trabajo y del soporte de la cámara web.

3.3.1 Diseño eléctrico

Previo a la incorporación estructural de la instrumentación a la plataforma robótica, es necesario conocer cómo interactúan esos elementos eléctricamente entre sí, con el fin de 1) acotar los límites de consumo eléctrico, 2) verificar su funcionamiento, 3) obtener una distribución preliminar de la instrumentación y 4) establecer que componentes electrónicos se requieren para complementar el circuito eléctrico.

Como primer paso para el diseño eléctrico de la plataforma robótica, se optó por establecer cuál es la máxima intensidad de corriente que la instrumentación puede extraer de la batería. Para ese fin, se hizo uso de la Tabla 3, en donde se especifica el consumo máximo de cada elemento y la carga total que representa la suma de los mismos.

Tabla 3. Máximo consumo eléctrico de la instrumentación

Elemento	**Cantidad**	**Voltaje [V]**	**Consumo máximo unitario [mA]**	**Consumo máximo por elementos [mA]**
Micromotor con eje extendido (rotor bloqueado)	2	6	1600	3200
Encoder magnético	2	2.7 - 18	20	40
Sensor ultrasónico (HC-SR04)	4	5	15	60
Arduino Mega 2560 R3	1	7 - 12	200	200
Módulo RF NRF24L01	1	3.3	115	115
Módulo L298N	1	7 - 50	36	36
	Consumo máximo total [mA]			3651

Fuente: ANEXOS COMPARATIVOS (ANEXO 3 al ANEXO 8)

Establecidos los topes de consumo de corriente, se procedió a clasificar la instrumentación según su propósito (Ver Tabla 4), esto es:

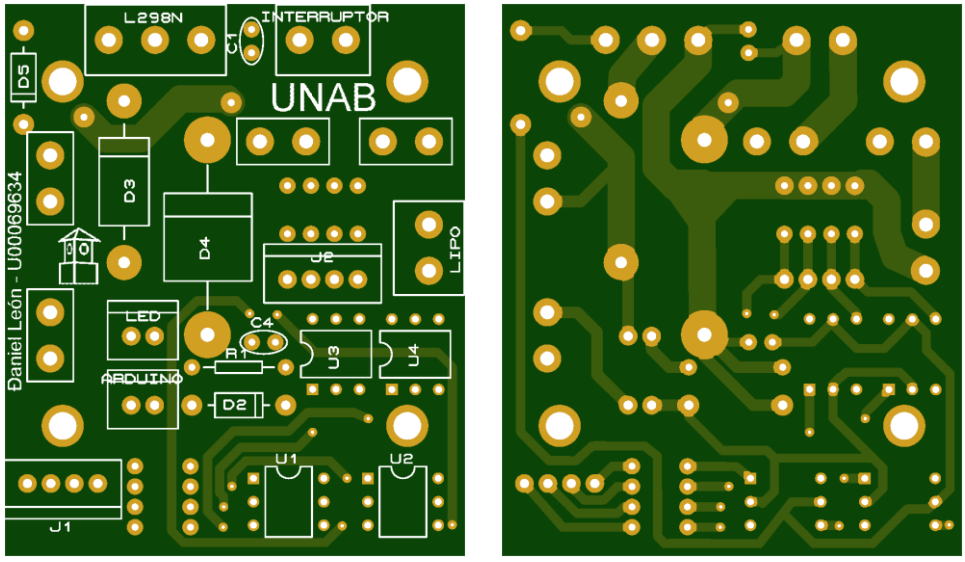
Tabla 4. Clasificación de la instrumentación

	Elemento
Potencia	Micromotor con eje extendido
	Módulo L298N
Control	Sensor ultrasónico (HC-SR04)
	Arduino Mega 2560 R3
	Módulo RF NRF24L01
	Encoder magnético

Fuente: Autor

En base a las tablas Tabla 3 y Tabla 4, se diseñó el circuito eléctrico del ANEXO 11. Este circuito integra eléctricamente la instrumentación junto a algunos componentes electrónicos pasivos y activos. Se verificó su funcionamiento al ser implementado en una placa de pruebas (protoboard). Una vez aprobado, se continuó con el diseño y manufactura de la placa de circuito impreso (PCB) de la Figura 30, cuyas propiedades se enlistan en la Tabla 5.

Figura 30. Placa de circuito impreso doble cara. (a) Cara superior. (b) Cara inferior



(a) (b)

Fuente: Autor

Tabla 5. Propiedades de la PCB doble cara

Propiedad	Valor
Capas	2
Dimensiones [mm]	50 x 60
Espesor [mm]	1.6
Color	Verde
Acabado superficial	Nivelación de soldadura (con plomo) de aire caliente
Peso del cobre [oz/ft^2]	1
Detalles del material	FR4-Estandar con Tg = 140ºC

Fuente: Autor

El siguiente paso fue el de soldar los componentes electrónicos (pasivos y activos) a la PCB, conectando posteriormente la instrumentación. Se realizó una prueba de funcionamiento al circuito, registrando los valores de alimentación y consumo de los componentes en la Tabla 6, donde se observa que el consumo de corriente de la instrumentación cuando no se tiene carga, es aproximadamente 12 veces menor que su consumo máximo.

Tabla 6. Consumo eléctrico sin carga de la instrumentación

Elemento	Cantidad	Voltaje [V]	Consumo unitario [mA]	Consumo por elementos [mA]
Micromotor con eje extendido (100% PWM)	2	6.01	61.36	122.72
Encoder magnético	2	11.55	8.94	17.88
Sensor ultrasónico (HC-SR04)	4	4.98	3.34	13.36
Arduino Mega 2560 R3	1	12.16	129.14	129.14
Módulo RF NRF24L01	1	3.29	9.62	9.62
Módulo L298N	1	12.38	13.56	13.56
	Consumo total [mA]			306.28

Fuente: Autor

Sin embargo, el consumo de corriente hallado no considera el movimiento de la plataforma robótica, por ende, no es un valor confiable a la hora de calcular la independencia del robot. Para ello, se requiere cuantificar el valor de corriente de operación (con carga), esto es, la intensidad de corriente que fluye de la batería a la plataforma mientras esta se encuentra en movimiento. Esta medición se realiza al prototipo mientras se le envía una señal de 100% de PWM a sus motores, resultando en:

$$i_{op} = 913.2\ [mA] \tag{25}$$

Por el ANEXO 9 se sabe que la capacidad de almacenamiento de carga de la batería empleada en este proyecto es de 2200 [mAh], por lo que se tendrá:

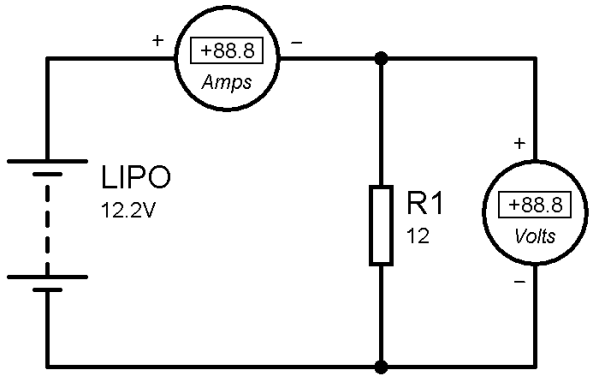
$$t_{op} = \frac{i_{bat}}{i_{op}} = \frac{2200\ [mAh]}{913.2\ [mA]} = 2.41[h] \tag{26}$$

Donde t_{op} es el tiempo de operación a máxima capacidad, i_{bat} la capacidad de almacenamiento de carga de la batería e i_{op} la corriente de operación de la plataforma robótica.

El tiempo de operación hallado (2 horas, 24 minutos y 36 segundos), sería el lapso teórico que le toma a la plataforma robótica descargar completamente la batería mientras funciona a plena capacidad. Sin embargo, el vendedor advierte sobre las consecuencias negativas de llevar a la batería a ese extremo (completa descarga)[59]. Sugiere descargar la batería hasta que esta llegue a un voltaje específico. No obstante, los fabricantes, vendedores y usuarios, no han llegado a un consenso sobre el valor exacto de ese voltaje, pero, todos concuerdan en que la descarga no se debe llevar más allá de la zona de deriva o precipitación.

Por lo anterior, se diseñó el montaje de la Figura 31 para establecer experimentalmente el valor del voltaje límite de descarga de una batería LiPo con las propiedades del ANEXO 9.

Figura 31. Montaje experimental de descarga de la batería LiPo



Fuente: Autor

Se reemplazó la corriente operativa (i_{op}) con una carga de prueba. Se aproximó su valor conforme a los componentes electrónicos disponibles para ese entonces, esto es:

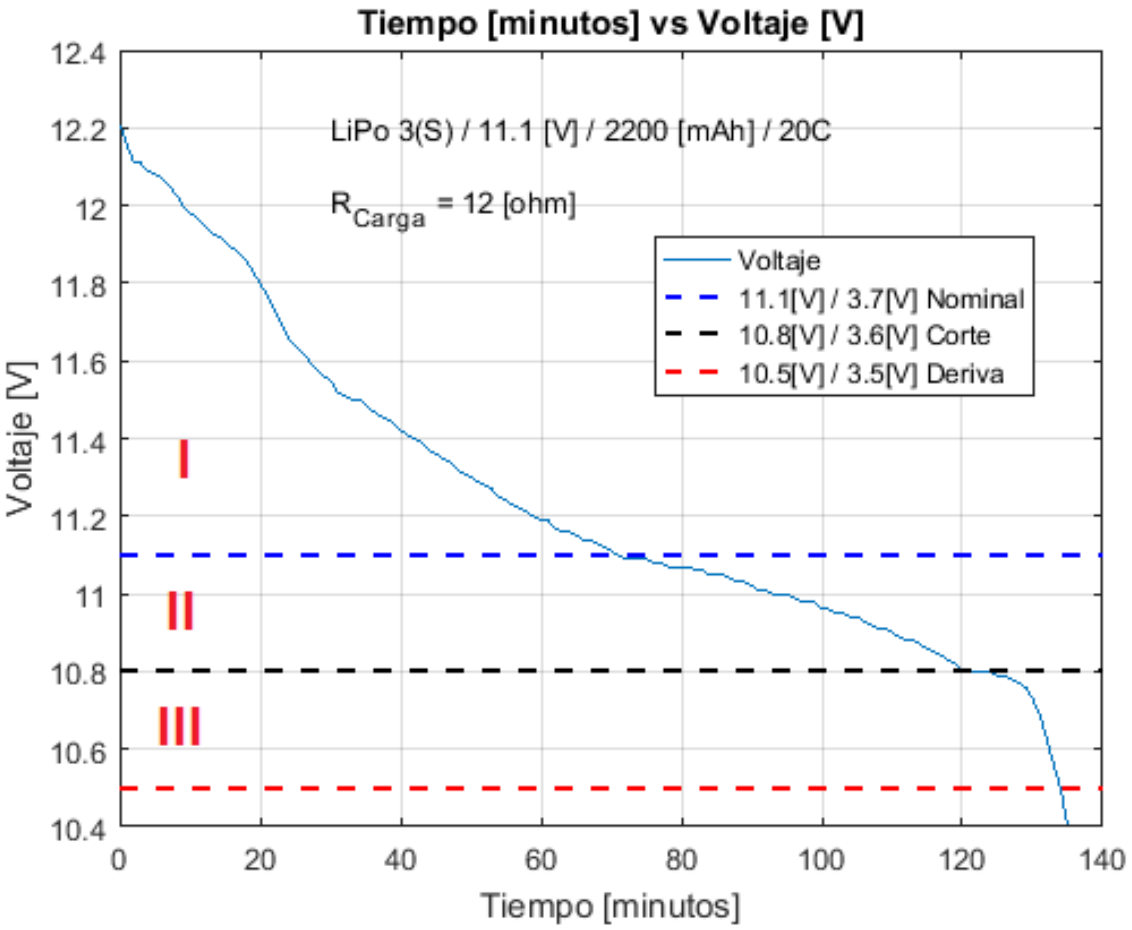
$$i_{op_{apx}} = \frac{V}{R} = \frac{12.2\ [V]}{12\ [\Omega]} = 1.02\ [A] \cong 0.9132\ [A] \tag{27}$$

Los resultados de las mediciones se grafican en la Figura 32, donde se observa la curva de descarga de voltaje de la batería. Esta curva se divide en tres regiones de descarga, cada una especificada por su respectivo valor de voltaje:

- Zona nominal (I). Es la región comprendida entre el voltaje máximo de carga de la batería LiPo (12.6 [V] totales con 4.2 [V] por celda) y el voltaje nominal (11.1 [V]/3.7 [V]). En este caso, la región entre 12.2 [V] y 11.1 [V]. Al optar por este

voltaje de desconexión, se tendría un tiempo de operación aproximado de 70 minutos a plena capacidad.

Figura 32. Curva de descarga de la batería LiPo



Fuente: Autor

- Zona de corte o inminente (II). Es la franja acotada entre el voltaje nominal (11.1 [V]/3.7 [V]) y el voltaje previo a la zona de deriva o precipitación (para esta batería, 10.8 [V]/3.6 [V]). Esta región aún continúa siendo segura para la batería conforme la curva de descarga. Si se usa este voltaje como criterio de desconexión, operando a toda capacidad, se tendrá una independencia aproximada de 120 minutos, 50 más que con el nominal.

- Zona en deriva o precipitación (II). Es la zona en la que el voltaje de la batería inicia una caída libre hasta sus valores mínimos. Si bien, esta se ubica por debajo del valor del voltaje de corte, es hasta un voltaje inferior a este (<10.75 [V]) en que se da inicio a la descarga acelerada. Descargar continuamente la batería hasta esta región, puede considerase como un peligro latente a la integridad de la batería, pese a que haya usuarios que recomienden descargar hasta un voltaje de 10.5 [V] (3.5 [V] por cada celda). Además, el tiempo extra que ofrece, aproximadamente 10 a 15 minutos más, no compensa el riesgo que acarrea llegar hasta este nivel de descarga.

Conforme a lo anterior, se optó por usar el voltaje de corte (10.8 [V]/3.6 [V]) como criterio de desconexión, significando 1) una independencia a plena carga de aproximadamente 120 minutos y 2) un ajuste de la alarma del indicador de voltaje con ese valor.

3.3.2 Diseño mecánico

En esta etapa del desarrollo de la plataforma robótica se diseñó la estructura de soporte para la instrumentación y su respectivo circuito eléctrico, considerando los siguientes requerimientos enlistados de mayor a menor relevancia:

- Reducir al máximo el área de la plataforma robótica sobre el espacio de trabajo.
- El peso máximo de la plataforma será de 1 [kg].
- La estructura de la plataforma debe ser modular.
- Los elementos electrónicos deben ser de acople no permanente.
- Se debe garantizar un fácil acceso a la batería.
- El sistema de locomoción consta de máximo dos motores con sus respectivas ruedas.
- La plataforma robótica debe poseer mínimo 2 ruedas de soporte.
- Las señales de encendido y voltaje deben ser visibles en todo momento.

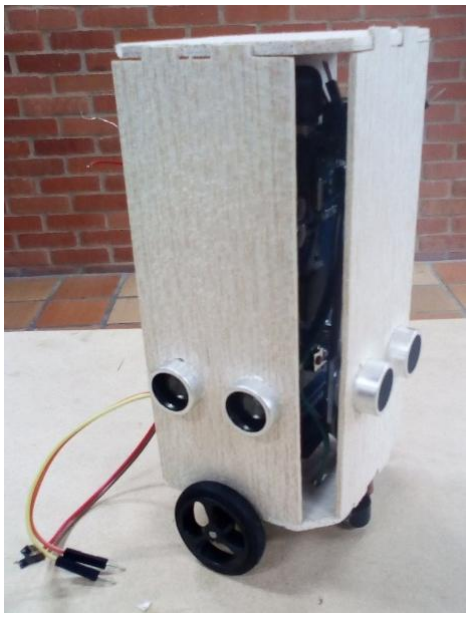
- De existir obstáculos en el espacio de trabajo, estos tendrán una altura no menor a 7 cm [cm].

En primera instancia, y basados en los anteriores requerimientos, se realizaron diversos bosquejos a mano de la plataforma robótica. Posterior a ello, algunos de esos diseños fueron digitalizados mediante la herramienta de diseño SolidWorks para validarlos por medio de análisis de elementos finitos, llegando a materializar tres de esos modelos tridimensionales. El primero de ellos, fue el diseño a base de madera de balsa (Ver Figura 33). Aunque fue el más compacto y liviano de todos (97.8 [g]), su estructura era frágil y propensa a agrietarse, esto, debido a que los requerimientos de peso y tamaño reducido desembocaron en cortes delgados del material. El segundo, fue el diseño impreso en PLA (Ver Figura 34) usando la técnica de modelado por deposición fundida (MDF). La estructura resultante, de densidad más elevada (relleno de impresión al 70%) y un peso de 132.2 [g], ofrecía una rigidez mayor que la del balso, sin embargo, algunas de las capas impresas no se adhirieron adecuadamente, ocasionando que partes de la estructura se quebraran al momento de manipularla. Por último, está el diseño en acrílico blanco de 3 [mm] de espesor (Ver Figura 35), con el que finalmente se satisficieron los requerimientos anteriormente enlistados de la siguiente forma:

- Dadas las dimensiones de los elementos que componen la instrumentación, se buscó la mejor distribución de los mismos en la estructura de la plataforma para que el área ocupada en el espacio de trabajo fuese la menor posible, esto es, 94.53 [cm^2] de 15600 [cm^2]. En la Figura 36 se pueden relacionar dichas áreas.
- El peso de la estructura de soporte es de 183.2 [g], mientras que el peso total de la plataforma robótica (incluida la estructura de soporte, la instrumentación y los elementos de sujeción) es de 726.55 ± 7.27 [g] (Ver Tabla 7).
- Mediante la implementación de uniones atornilladas, se logró prototipar una estructura modular (Ver ANEXO 12, ANEXO 13 y ANEXO 14) de ensamble

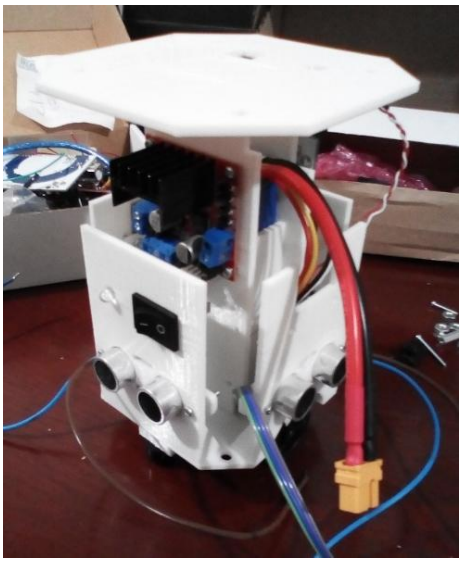
simple y de acceso sencillo a sus componentes, facilitando el mantenimiento o reemplazo de los mismos.

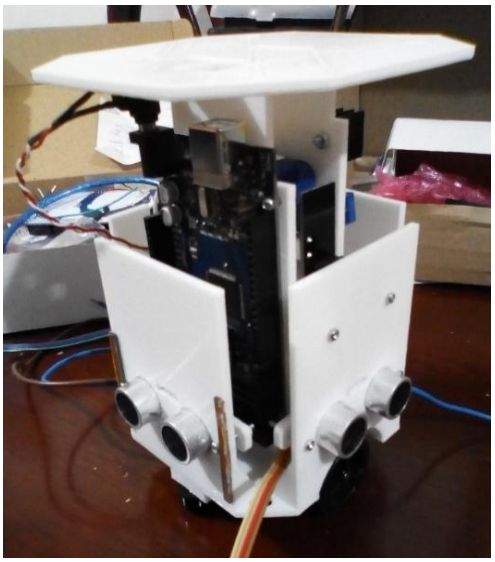
Figura 33. Diseño de la plataforma robótica en madera de balsa



Fuente: Autor

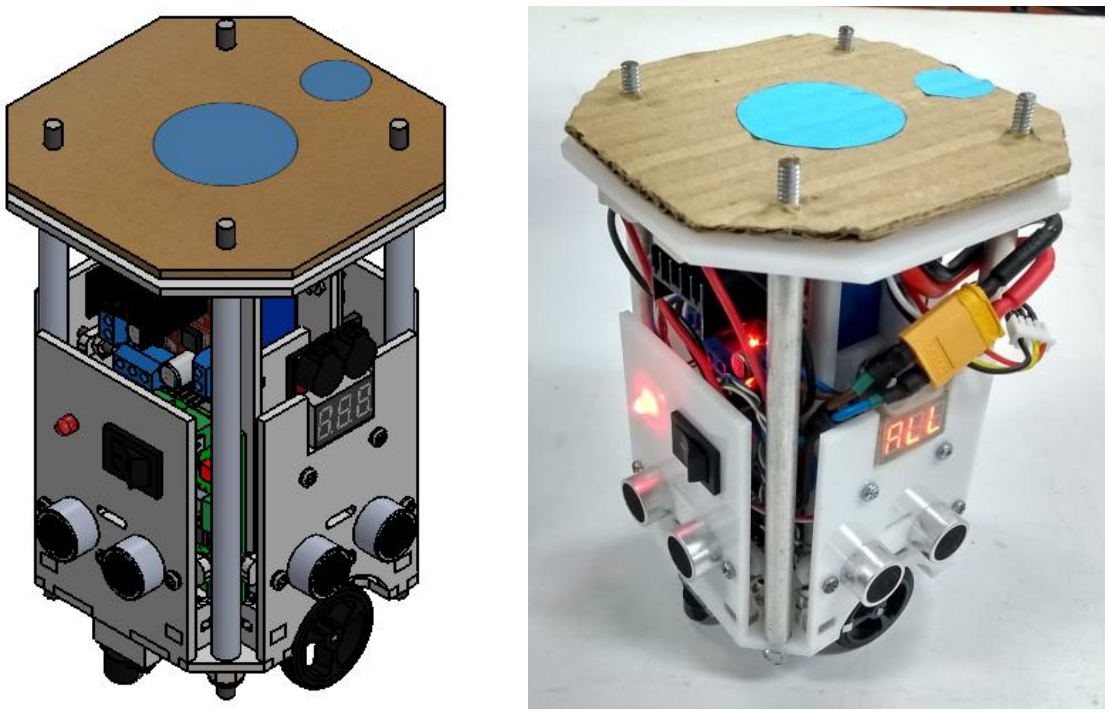
Figura 34. Diseño de la plataforma robótica impreso en PLA





Fuente: Autor

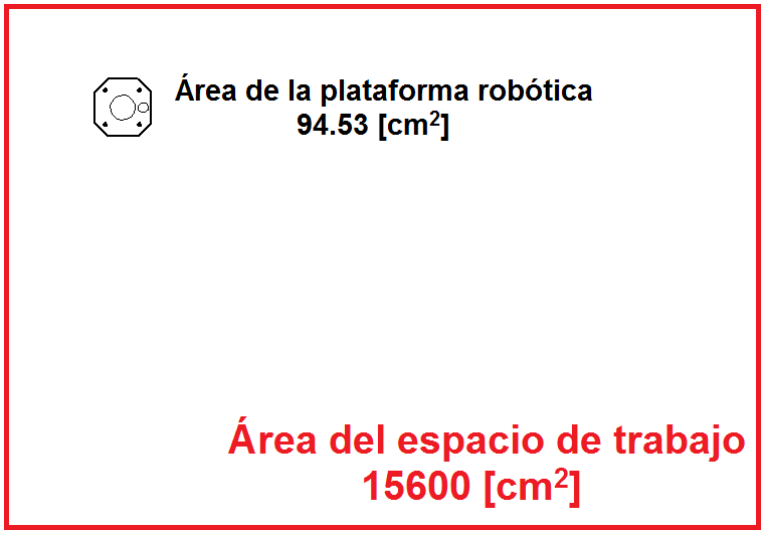
Figura 35. Diseño de la plataforma robótica en acrílico blanco de 3 [mm] de espesor. (a) Diseño CAD. (b) Prototipo



(a) (b)

Fuente: Auto

Figura 36. Área ocupada por la plataforma robótica en el espacio de trabajo.



Fuente: Autor

Tabla 7. Distribución del peso de la plataforma robótica

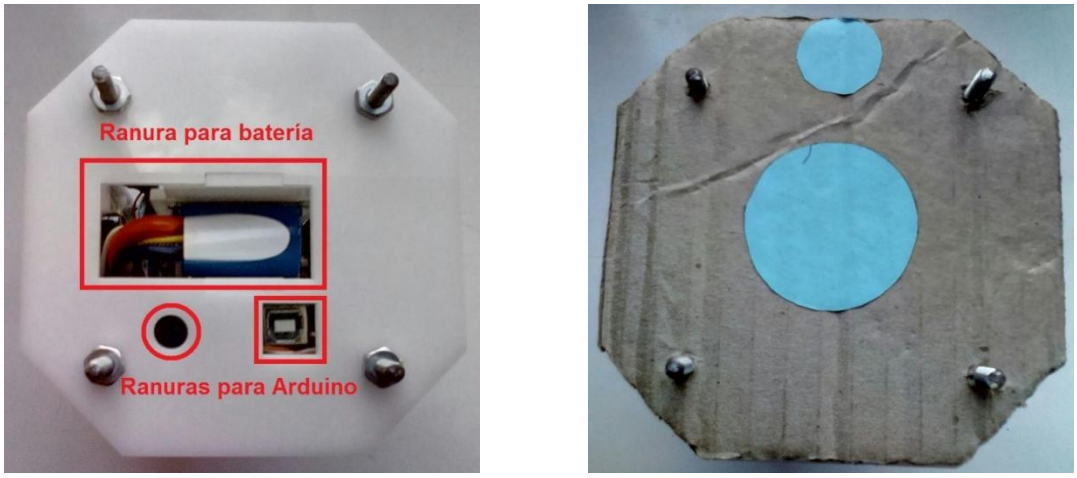
	Elemento	Cantidad	Peso [g]	
			Unitario	Total
Acrílico	Estructura	1	173.9	173.9
	Soporte rueda loca	2	1.3	2.6
	Soporte NRF24L01	1	0.7	0.7
	Soporte HC-SR04	4	1.5	6
Instrumentación	Rueda	2	3.1	6.2
	Micromotor	2	9.5	19
	Encoder	2	1	2
	Sensor HC-SR04	4	15	60
	Arduino Mega 2560	1	37	37
	Módulo L298N	1	33	33
	Módulo NRF24L01	1	6	6
	Batería LiPo	1	188	188
Elementos de sujeción		92	-	192.15
			Total	726.55

Fuente: Autor

- Se dispuso de una amplia ranura en la tapa de la plataforma robótica, mediante la cual se ingresa, ajusta y ubica la batería en el centro de la estructura. Esta ranura, junto a la de comunicación serial y alimentación del Arduino Mega 2560, se ocultan mediante la segunda cubierta con los identificadores del robot (Ver Figura 37).
- En la Figura 38 se observan dos micromotores, cada uno con su respectivo soporte y encorder magnético, junto a dos ruedas locas, componentes que en conjunto, representan el sistema de locomoción y soporte de la plataforma robótica.
- En la Figura 39 se observan las señales de encendido y voltaje de la plataforma robótica, ubicadas en el lado exterior de los módulos posterior y lateral derecho

respectivamente, garantizando su visibilidad durante todo el tiempo de trabajo de la plataforma.

Figura 37. Ranura de la plataforma robótica para el ingreso de la batería. (a) Expuesta. (b) Oculta



(a) (b)

Fuente: Autor

Figura 38. Vista inferior de la plataforma robótica

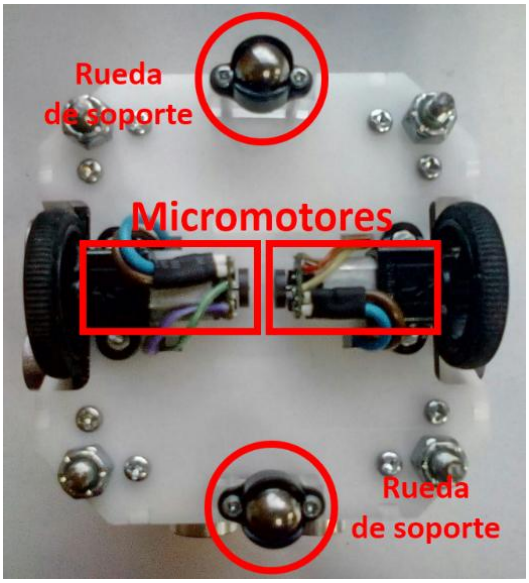
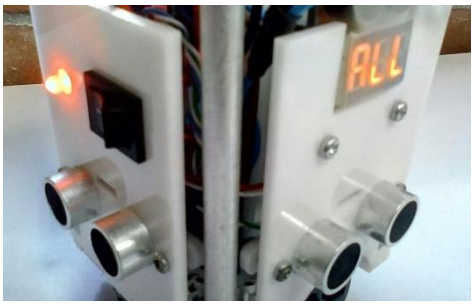


Figura: Autor

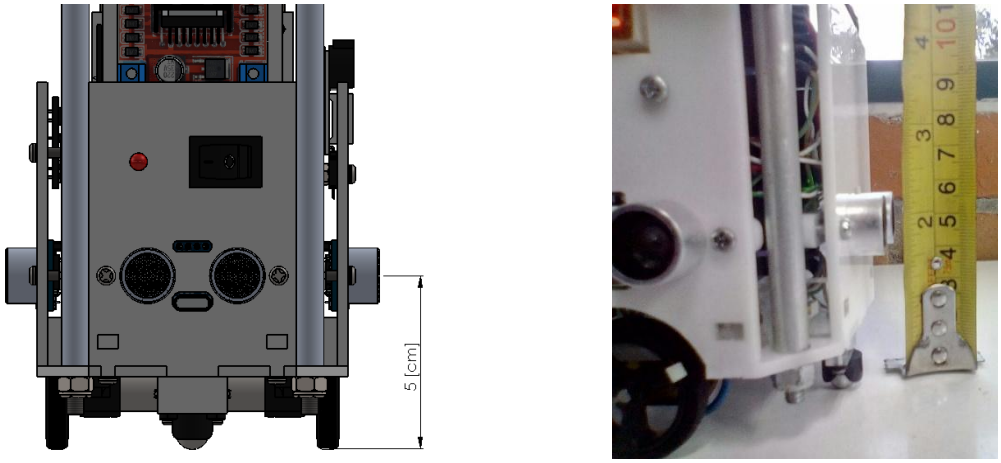
Figura 39. Señales de encendido y voltaje de la plataforma robótica



Fuente: Autor

- Cada una de las cuatro caras de la estructura del robot cuenta con un sensor ultrasónico HC-SR04 como apoyo al sistema de visión artificial. Estos sensores se ubican a una altura de 5 [cm] del suelo con respecto a su punto medio (Ver Figura 40), esto es, 2 [cm] por debajo de la altura mínima de los posibles obstáculos dentro del espacio de trabajo de la plataforma robótica.

Figura 40. Ubicación de los sensores ultrasónicos HC-SR04



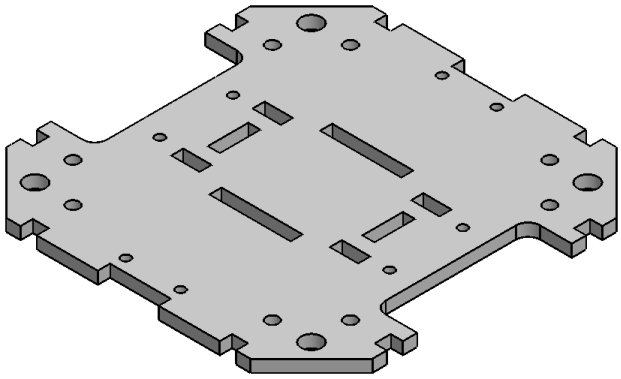
Fuente: Autor

3.3.2.1 Validación por análisis de elementos finitos

Para comprobar si la estructura diseñada podía soportar la carga arbitraria de 1 [kg], se realizó un análisis de elementos finitos en SolidWorks al modelo CAD de la

plataforma robótica de la Figura 35a. Dicho análisis, se enfocó en el eslabón principal de la estructura, la base (Ver Figura 41), siendo esta, la pieza a la que se acoplan los motores y el resto de los módulos con su respectiva instrumentación.

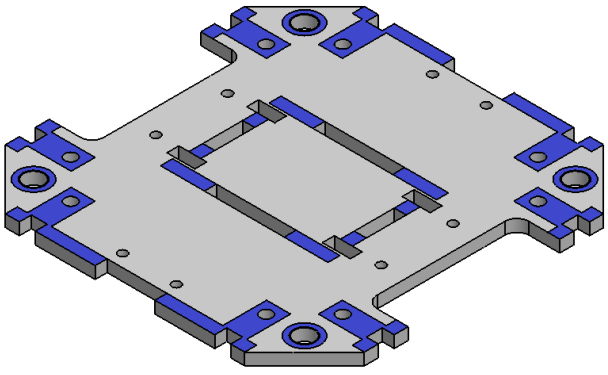
Figura 41. Módulo principal de soporte de la estructura



Fuente: Autor

Para facilitar el análisis por elementos finitos, se reemplazaron las piezas del modelo por la física que las representa, en este caso, su peso. Dicha fuerza se distribuye sobre la superficie de la base mediante áreas de aplicación según sean las zonas de contacto entre los elementos, esto es, concentrar el peso que incide sobre la estructura en las zonas azules de la Figura 42.

Figura 42. Áreas de concentración del peso de la plataforma robótica



Fuente: Autor

Identificadas las áreas de aplicación, se procedió con el análisis realizando los siguientes pasos:

I. Importar a SolidWorks la geometría de la Figura 42.
II. Activar el complemento de simulación (*Simulation*) en el "Administrador de comandos" del software.
III. Seleccionar el tipo de análisis, para este caso, análisis estático: *Simulation>Nuevo estudio>Análisis estático*.
IV. Aplicar al sólido el material correspondiente, para este estudio, acrílico. Las propiedades de ese material se enlistan en la Tabla 8.

Tabla 8. Propiedades del acrílico

Propiedad	**Valor**
Módulo elástico	3.00e+9 [N/m^2]
Coeficiente de Poisson	0.35
Módulo cortante	8.90e+8 [N/m^2]
Densidad de masa	1200 [kg/m^3]
Límite de tracción	7.30e+7 [N/m^2]
Límite elástico	4.50e+7 [N/m^2]
Coeficiente de expansión térmica	5.2e-5 [1/K]
Conductividad térmica	0.21 [W/(m·K)]
Calor específico	1500 [J/(kg·K)]

Fuente: SOLIDWORKS[55]

V. Seleccionar las áreas del modelo en donde se soporta la pieza para asignarlas como sujeciones de geometría fija. Estas áreas se representan en la Figura 43 mediante el color rojo y equivalen a las zonas de contacto entre la superficie inferior de la base, los soportes del micromotor y las tuercas de seguridad de la varilla roscada de 3/16".

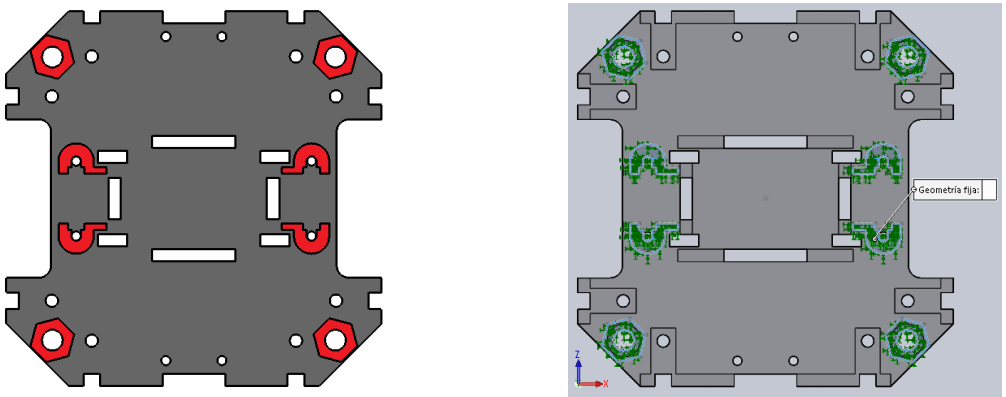
VI. Aplicar el peso como carga externa sobre las áreas de concentración de la Figura 42 (Ver Figura 44).

VII. Configurar y aplicar el mallado. Los parámetros empleados y su resultado se observan en la Figura 45.

VIII. Iniciar el estudio estático.

IX. Finalizar el estudio con la Interpretación de los resultados obtenidos, esto, con el fin de establecer una respuesta al interrogante que originó el análisis por elementos finitos.

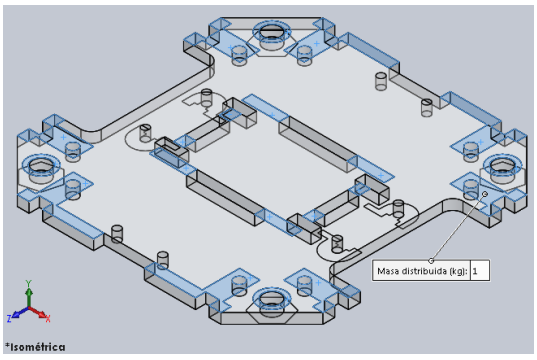
Figura 43. Vista inferior de la base. (a) Áreas de soporte de la base. (b) Asignación de la sujeción de geometría fija en SolidWorks



(a) (b)

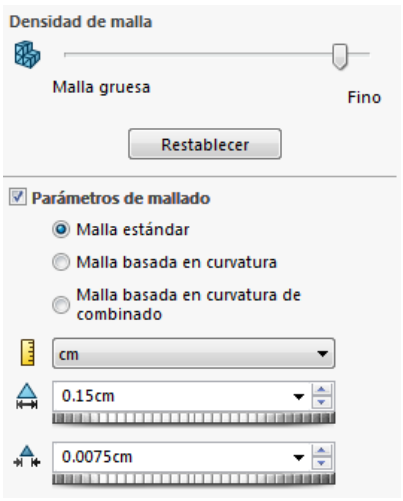
Fuente: Autor

Figura 44. Asignación del peso como carga externa en SolidWorks

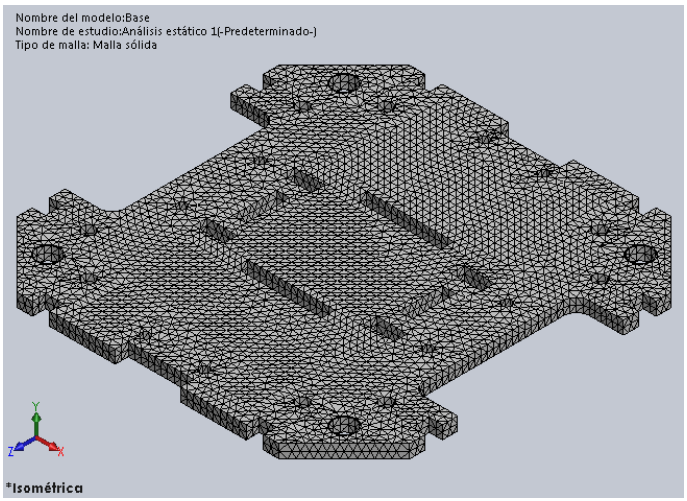


Fuente: Autor

Figura 45. Mallado del modelo CAD. (a) Parámetros de mallado. (b) Resultado



(a)



(b)

Fuente: Autor

Del análisis realizado se obtuvieron 3 tipos de resultados: tensiones de von Mises (Ver Figura 46), desplazamientos (Ver Figura 47) y factores de seguridad (Ver Figura 48), mediante los cuales se pudo esclarecer si la pieza analizada soportaba o no la carga arbitraria. Dichos resultados se describen a continuación:

- Tensiones de von Mises. En la leyenda de la Figura 46 se observa el rango de valores de las tensiones de von Mises desarrolladas por el modelo ante la carga arbitraria de 1 [kg], siendo el valor máximo y mínimo:

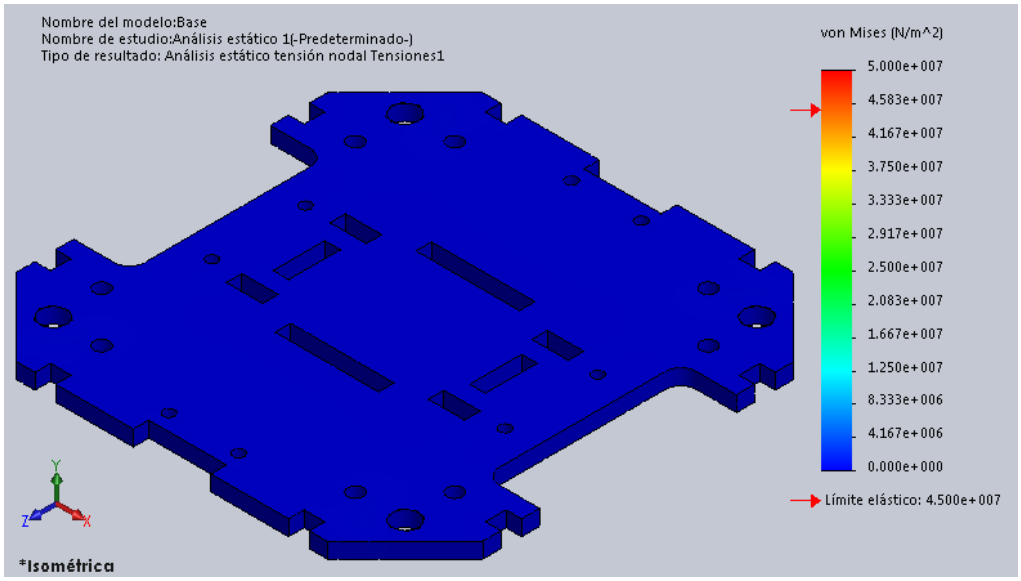
$$\sigma_{máx} = 5.964x10^{5}\ [N/m^2] \quad (28)$$

$$\sigma_{mín} = 3.245x10^{2}\ [N/m^2] \quad (29)$$

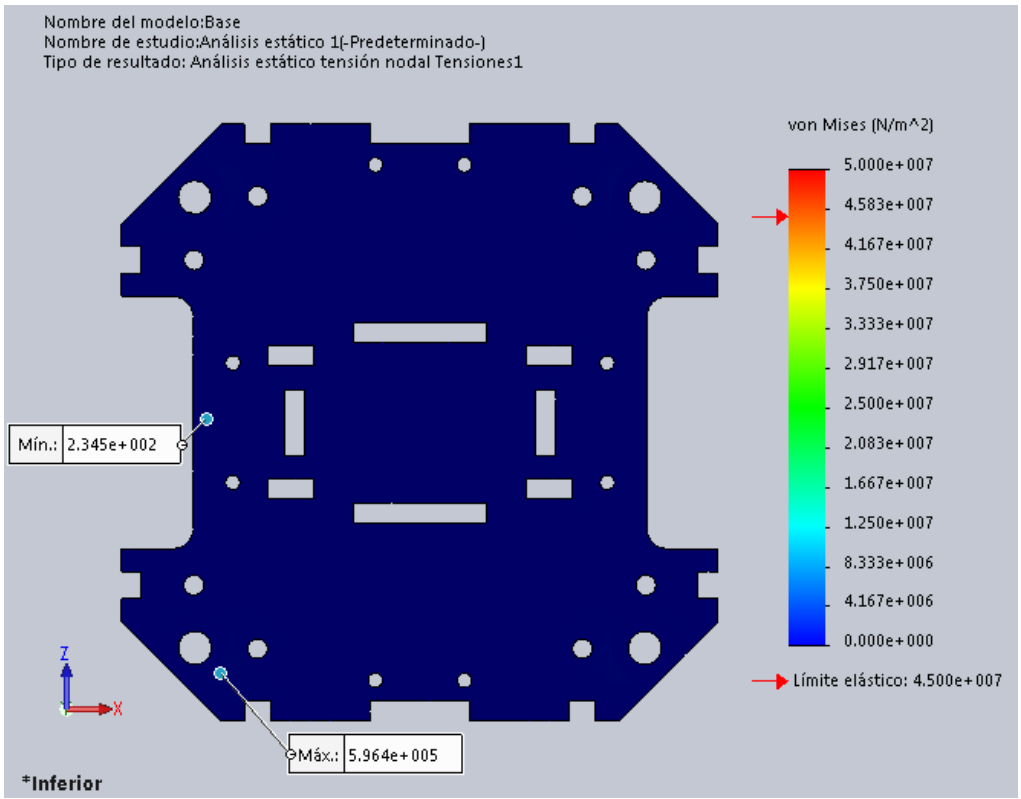
De las propiedades del acrílico empleado se saber que el límite elástico es de:

$$\sigma = 4.500x10^{7}\ [N/m^2] \quad (30)$$

Figura 46. Tensiones de von Mises. (a) Vista isométrica del modelo CAD. (b) Vista inferior del modelo CAD. (c) Variación en la escala de la leyenda con fines visuales



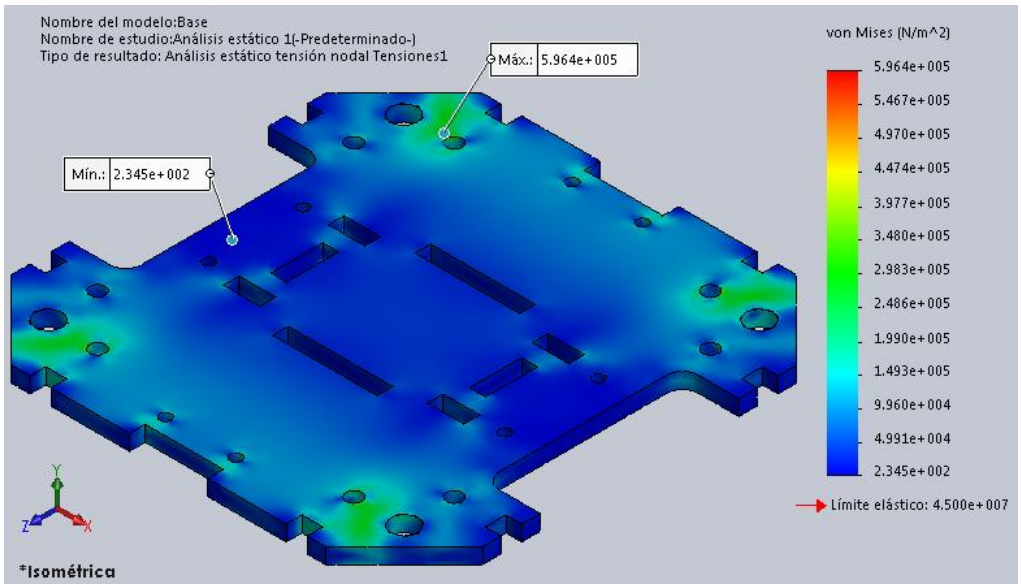
(a)



(b)

Fuente: Autor

Figura 46. (Continuación)



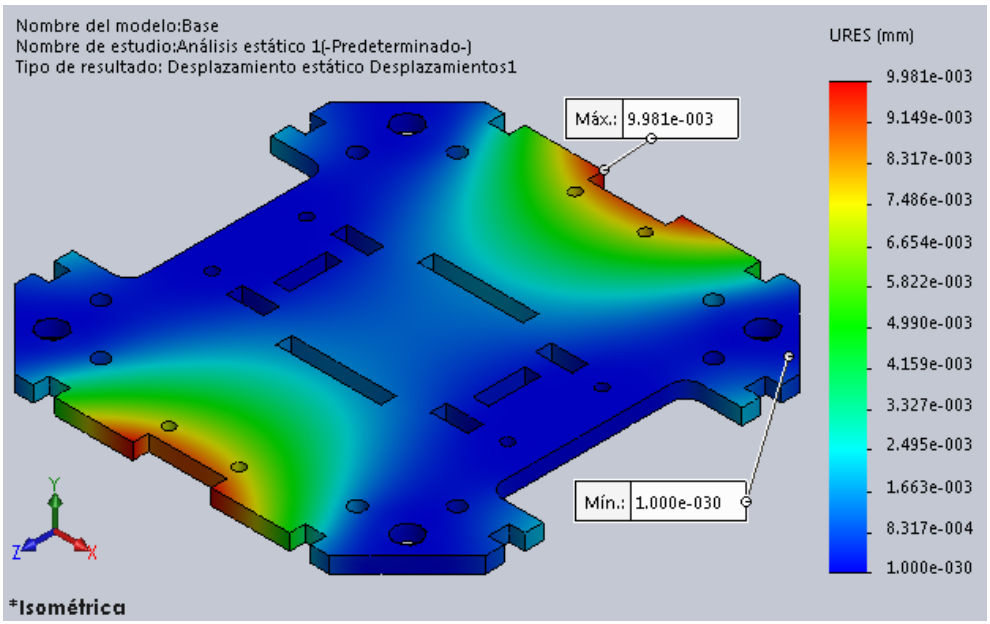
(c)

Fuente: Autor

Al comparar (28) y (30) se tiene que $\sigma \gg \sigma_{máx}$, por lo que la carga a la que se somete el material no producirá una deformación considerable a la pieza que le impida recuperar su forma original. Adicionalmente, dada la ubicación de la tensión máxima y mínima (Ver Figura 46b), se determina que las áreas en donde el peso tiene mayor incidencia sobre la pieza, son la zona frontal y trasera de la estructura del robot, siendo la zona media la menos afectada. Además, sugiere que el par aplicado a la tuerca de seguridad que une la base y la tapa del robot mediante la varilla roscada de 3/16", debe ajustar sólo lo necesario ya que este puede incrementar los esfuerzos en la zona detallada como de máxima tensión.

- Desplazamientos. Según la leyenda de la Figura 47, el desplazamiento máximo generado al aplicar la carga es de 9.981×10^{-3} [mm]. Como se mencionó previamente, este desplazamiento no implica un sobrepaso al límite elástico del material.

Figura 47. Desplazamientos



Fuente: Autor

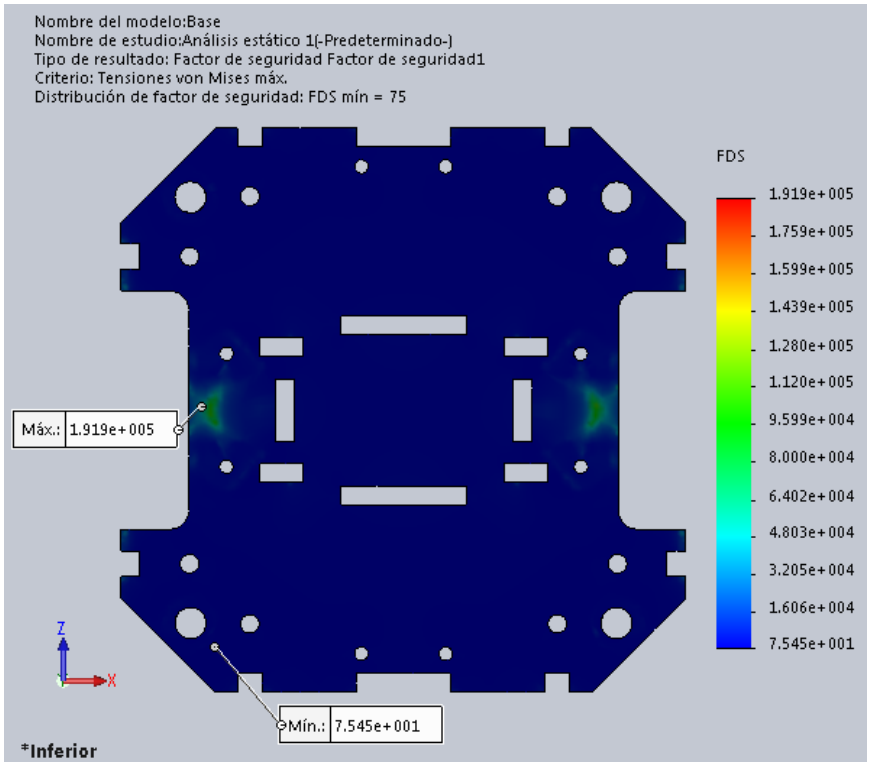
- Factores de seguridad (FDS). De la Figura 48 se deduce que, como resultado de aplicar la carga, el modelo CAD presenta un FDS máximo de $1.919x10^5$ y un mínimo de $7.545x10^1$, indicando que 1) el diseño no presenta zonas en las que el material pueda fallar ya que los FDS son mayores que uno ($FDS > 1$)[56] y, 2) el espesor del material se puede reducir, siempre y cuando, las iteraciones que se hagan con ese fin no arrojen resultados de $FDS \leq 1$ ya que, de ser así, el diseño podría presentar zonas en las que el material se encuentra en falla o próximo a fallar[57].

  Para hallar el valor del FDS manualmente, se relacionan (28) y (30) entre sí[58], esto es:

$$FDS = \frac{\sigma}{\sigma_{máx}} = \frac{4.500x10^7\ [N/m^2]}{5.964x10^5\ [N/m^2]} = 75.453 \qquad (31)$$

Corroborando con ello que el valor teórico corresponde al valor computado por el software (Ver Figura 48), dando validez a los resultados obtenidos y al procedimiento aplicado.

Figura 48. Factores de seguridad



Fuente: Autor

3.3.3 Diseño estructural

Entiéndase “diseño estructural” como el medio para obtener la estructura de soporte de la cámara web (Ver ANEXO 15) y la estructura del espacio de trabajo de la plataforma robótica (Ver ANEXO 16). En la Figura 49 se aprecia la delimitación del espacio de trabajo del proyecto en el laboratorio de oleoneumática de la Universidad. Esta delimitación consta del área de trabajo (Ver Figura 49a), el soporte de la cámara web (Ver Figura 49b), el espacio de trabajo de la plataforma robótica (Ver Figura 49c) y la estación de trabajo.

Figura 49. Delimitación del espacio de trabajo. (a) Área de trabajo. (b) Soporte de la cámara web. (c) Espacio de trabajo de la plataforma robótica

(a)

(b)

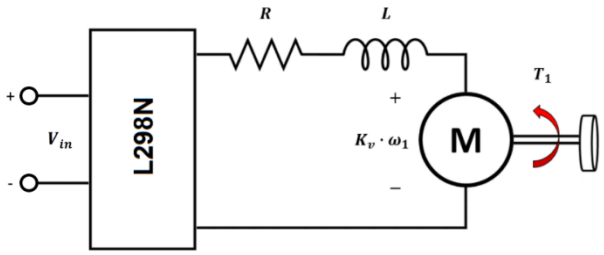
(c)

Fuente: Autor

3.3.4 Diseño del control de velocidad de las ruedas de la plataforma robótica

Se optó por diseñar el control de velocidad por medio de un modelo matemático de caja blanca. Su diseño se inicia con la interpretación del modelo equivalente del sistema de locomoción de la Figura 50.

Figura 50. Modelo equivalente del sistema de locomoción



Fuente: Autor

Dado que el módulo L298N (puente H) se comporta como una ganancia al sistema, se tendrá que:

$$V_{in} = K \cdot V_{cc} \quad ; \quad 0 < K < 1 \tag{32}$$

Donde, V_{in} es el voltaje de entrada, K la ganancia del driver y V_{cc} el voltaje de alimentación. Analizando el lazo del sistema, se tendrán las siguientes ecuaciones diferenciales:

$$V_{in} = Ri + L\frac{di}{dt} + K_v\omega_1 \tag{33}$$

$$T_m - B_m\omega_1 - T_1 = J_m\frac{d\omega_1}{dt} \quad ; \quad T_m = K_i i$$

$$K_i i - B_m\omega_1 - T_1 = J_m\frac{d\omega_1}{dt} \tag{34}$$

Aplicando la Transformada de Laplace a (33) y (34), se tendrá:

$$V_{in}(s) = Ri(s) + Lsi(s) + K_v\omega_1(s) = i(s)(R + Ls) + K_v\omega_1(s) \tag{35}$$

$$K_i i(s) - B_m\omega_1(s) - T_1(s) = J_m s\omega_1(s) \tag{36}$$

Aplicando el principio de superposición al sistema, se obtiene:

$$G_a(s) = \left.\frac{\omega_1(s)}{V_{in}(s)}\right|_{T_1(s)=0} \tag{37}$$

$$G_b(s) = \left.\frac{\omega_1(s)}{T_1(s)}\right|_{V_{in}(s)=0} \tag{38}$$

Por lo tanto, cuando $T_1(s) = 0$, se despeja $i(s)$ de (36):

$$K_i i(s) - B_m\omega_1(s) = J_m s\omega_1(s)$$

$$i(s) = \frac{J_m s\omega_1(s) + B_m\omega_1(s)}{K_i} = \frac{\omega_1(s)(J_m s + B_m)}{K_i} \tag{39}$$

Reemplazando (39) en (35):

$$V_{in}(s) = \left[\frac{\omega_1(s)(J_m s + B_m)}{K_i}\right](R + Ls) + K_v\omega_1(s)$$

$$K_i V_{in}(s) = (R + Ls)\omega_1(s)(J_m s + B_m) + K_i K_v\omega_1(s)$$

$$K_i V_{in}(s) = [(R + Ls)(J_m s + B_m) + K_i K_v]\omega_1(s)$$

$$G_a(s) = \left.\frac{\omega_1(s)}{V_{in}(s)}\right|_{T_1(s)=0} = \frac{K_i}{(R + Ls)(J_m s + B_m) + K_i K_v} \tag{40}$$

En vista que se desea relacionar la ecuación de transferencia $G_a(s)$ con el porcentaje de PWM, se tendrá que:

$$V_{in} = K \cdot PWM(s) \tag{41}$$

Obteniendo:

$$G_a(s) = \frac{\omega_1(s)}{PWM(s)}\bigg|_{T_1(s)=0} = \frac{KK_i}{(R+Ls)(J_m s + B_m) + K_i K_v} \tag{42}$$

Dado que los motores son alimentados con 6 [V], se tiene que:

$$K = \frac{6}{255} = 0.0235 \tag{43}$$

Resultando en la ecuación de transferencia del sistema de locomoción de la plataforma robótica. Su nomenclatura se enlista en la Tabla 9.

$$G_a(s) = \frac{\omega_1(s)}{PWM(s)}\bigg|_{T_1(s)=0} = \frac{0.0235K_i}{(R+Ls)(J_m s + B_m) + K_i K_v} \tag{44}$$

Tabla 9. Nomenclatura de la ecuación de transferencia $G_a(s)$

Símbolo	**Significado**
$\omega_1(s)$	Velocidad angular de la rueda
$PWM(s)$	Porcentaje de PWM
K_i	Constante mecánica del motor
K_v	Constante contraelectromotríz del motor
R	Resistencia de armadura del motor
L	Inductancia de armadura del motor
J_m	Inercia del motor
B_m	Coeficiente de fricción viscosa del motor

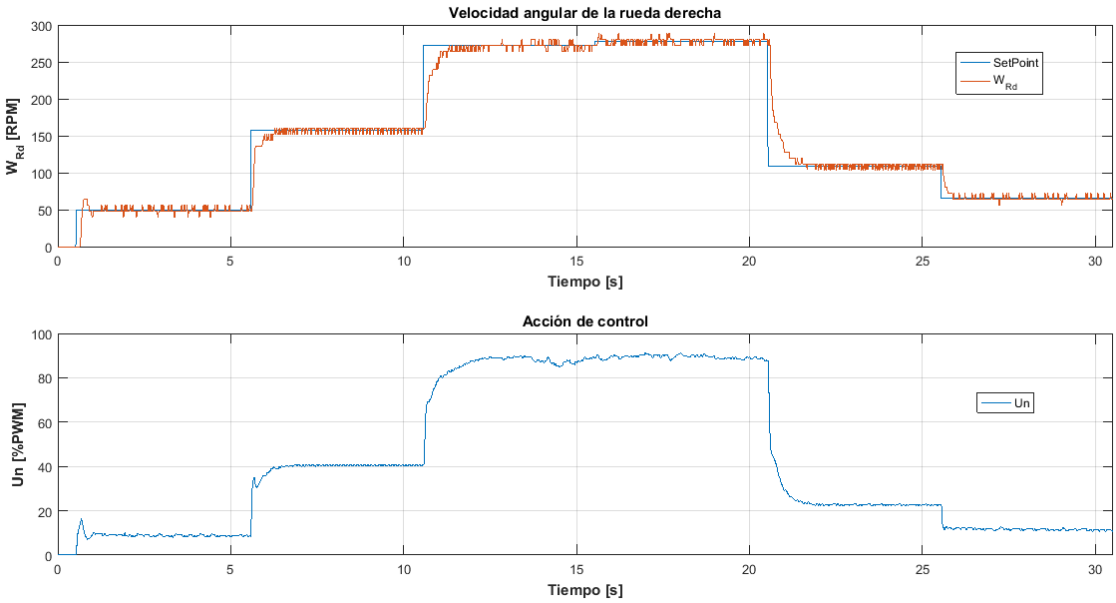
Fuente: Autor

Se discretizó la función de transferencia con un periodo de muestreo de 1 [ms], obteniendo el siguiente compensador PID:

$$G_a(z) = \frac{0.09z^2 + 0.0189z - 0.0711}{z^2 - z} \tag{45}$$

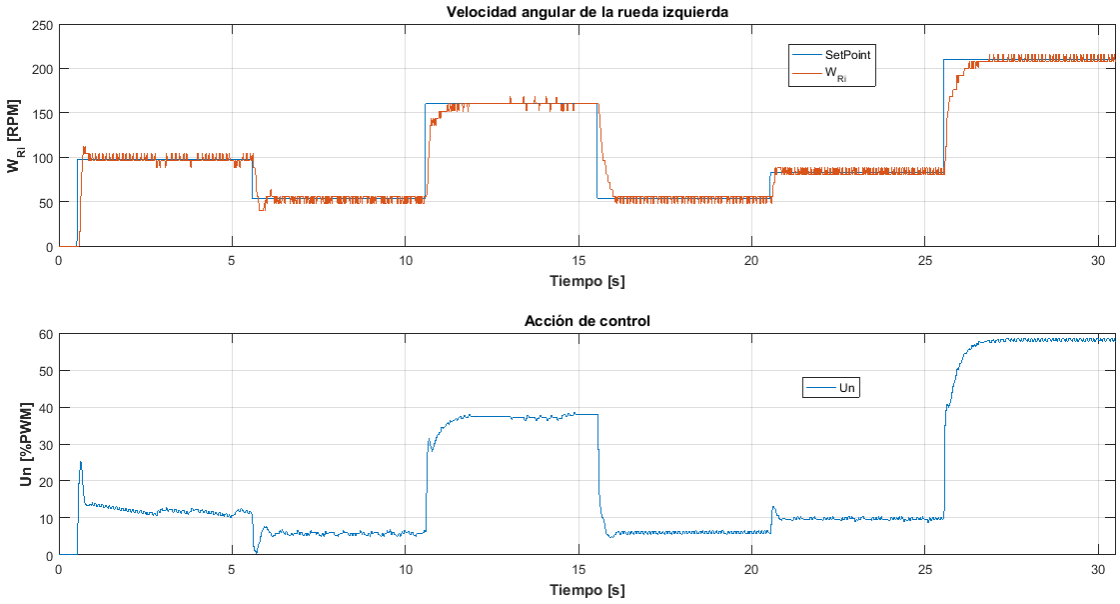
Los resultados de implementar el mismo compensador en ambos motores de la plataforma robótica se muestran en las Figura 51 y Figura 52. En ellas se observa la respuesta del sistema ante la variación del punto de consigna.

Figura 51. Velocidad angular de la rueda derecha



Fuente: Autor

Figura 52. Velocidad angular de la rueda izquierda



Fuente: Autor

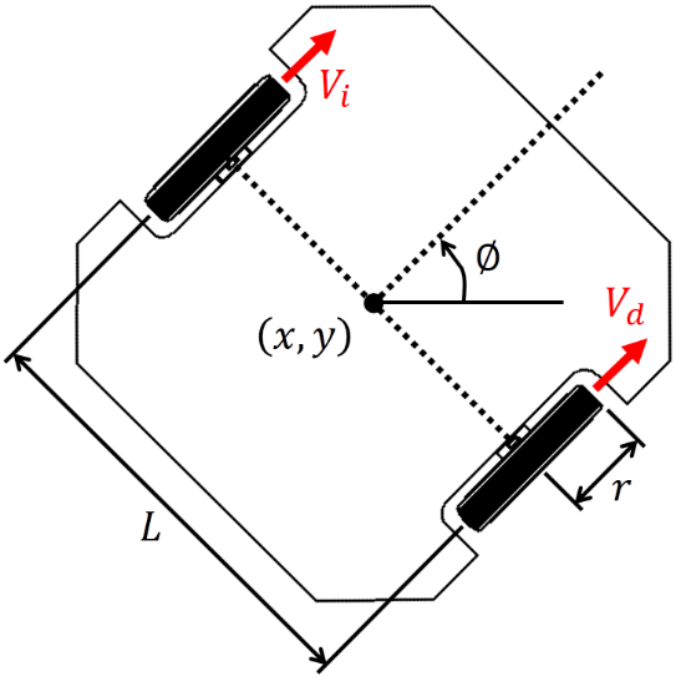
3.3.5 Desarrollo del modelo cinemático

El modelo cinemático de este proyecto se basa en el modelo propuesto por el Dr. Magnus Egerstedt[60], director ejecutivo del Instituto de Robótica y Máquinas Inteligentes del Instituto de Tecnología de Georgia (EEUU), quien parte del modelo cinemático del monociclo:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\emptyset} \end{bmatrix} = \begin{bmatrix} \cos(\emptyset) & 0 \\ \sin(\emptyset) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\emptyset) \\ v \cdot \sin(\emptyset) \\ \omega \end{bmatrix} \quad (46)$$

para llegar al modelo cinemático diferencial, obteniendo las siguientes ecuaciones en base a la Figura 53:

Figura 53. Cinemática del robot diferencial



Fuente: Autor

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\emptyset} \end{bmatrix} = \begin{bmatrix} \frac{(\omega_d + \omega_i) \cdot r \cdot \cos(\emptyset)}{2} \\ \frac{(\omega_d + \omega_i) \cdot r \cdot \sin(\emptyset)}{2} \\ \frac{(\omega_d - \omega_i) \cdot r}{L} \end{bmatrix} \quad (47)$$

Donde $\dot{x}$ es la componente de la velocidad del robot en el eje x, $\dot{y}$ es la componente de la velocidad del robot en el eje y, $\dot{\emptyset}$ es la velocidad angular del robot, ω_d la velocidad angular de la rueda derecha, ω_i la velocidad angular de la rueda izquierda, r el radio de la rueda, L la distancia entre las ruedas y $\emptyset$ la orientación del robot con respecto al eje x. Las ecuaciones en (47) representan la cinemática directa del robot móvil diferencial, mediante las cuales, integrando $\dot{x}$, $\dot{y}$ y $\dot{\emptyset}$, se obtienen la posición (x, y) y orientación ($\emptyset$) de la plataforma robótica en el espacio de trabajo. Las entradas de este modelo son las velocidades angulares de las ruedas del robot.

3.3.6 Selección de la estrategia de control aplicada al control de formaciones

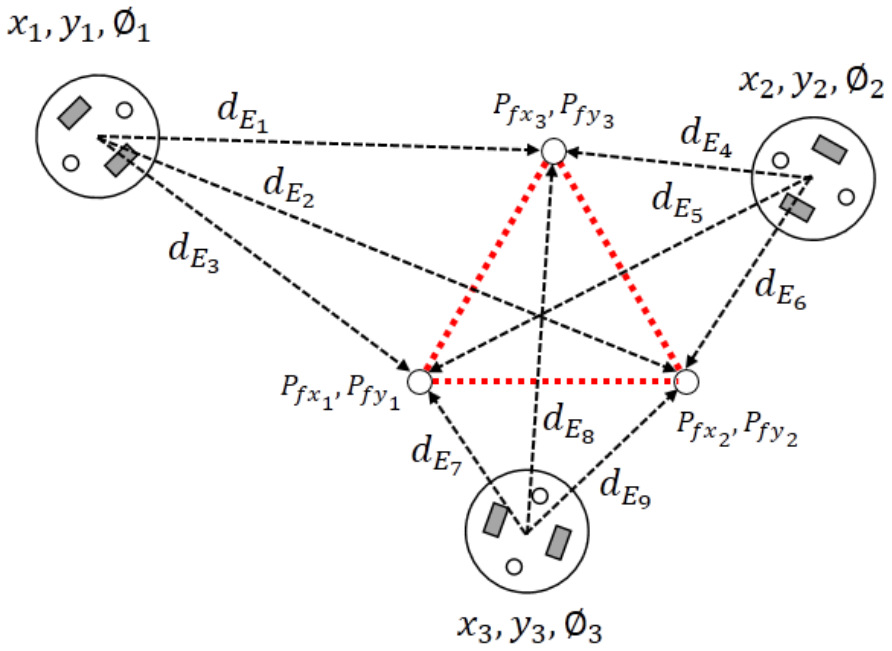
Complementario a lo que se ha dicho al respecto previamente (subtítulo 2.2.2 Control de formación), la estrategia de control de formación visual seleccionada se divide en dos etapas, la asignación y el seguimiento, procesos secuenciales que se describen a continuación.

3.3.6.1 Asignación de puntos objetivos

Es la primera etapa a desarrollar y consiste en asignar los puntos objetivos de la figura geométrica a realizar a cada una de las plataformas robóticas operativas en el espacio de trabajo. Esto, con el fin de establecer el punto final de la trayectoria a efectuar por cada robot. Esta asignación será invariante en el tiempo, siempre y cuando, el número de robots o el número de puntos en el espacio de trabajo sean constantes, es decir, de haber un cambio en la cantidad de robots, incremento o decremento, al igual que con los puntos, la asignación se realizará nuevamente.

Para realizar la distribución de los puntos es necesario ejecutar los siguientes pasos: 1) obtener la matriz con las distancias euclídeas entre los robots y los puntos de la figura geométrica a desarrollar (Ver Figura 54) y 2) aplicar el criterio de asignación seleccionado a esa matriz de distancias, siendo este último, el resultado de la comparación de siete (7) criterios de asignación (Ver Tabla 10) con los que se buscaba encontrar la mínima distancia recorrida por todo el sistema. Para ello, se plantearon dos escenarios en los que participaban igual cantidad de puntos objetivos como de plataformas robóticas:

Figura 54. Distancias euclídeas de las plataformas robóticas a los puntos objetivos de la figura geométrica



Fuente: Autor

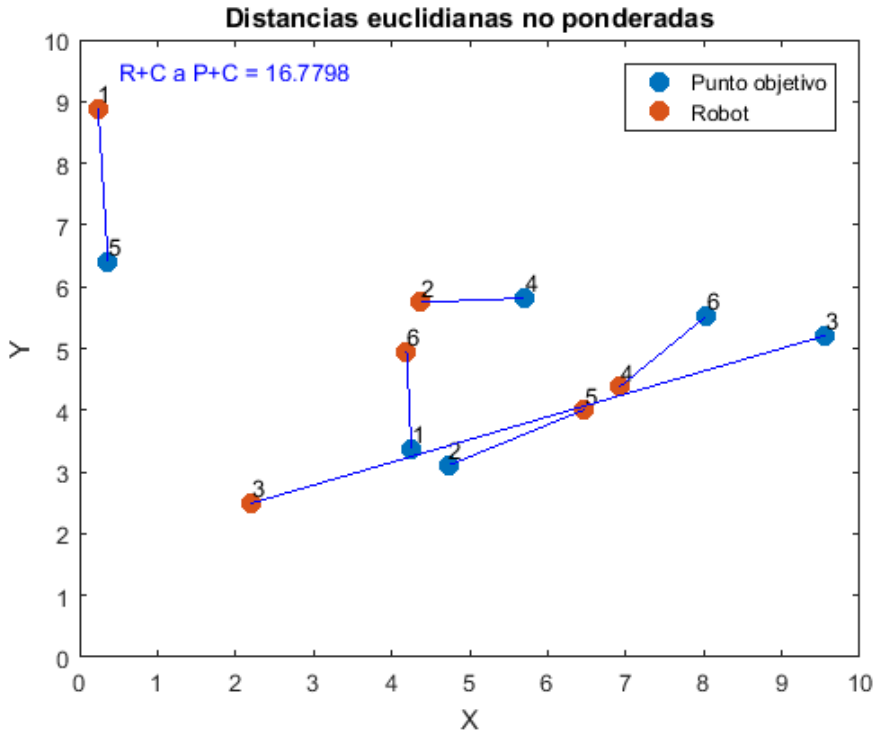
Tabla 10. Criterios de asignación

Criterio	Siglas
Robot más cercano al punto más cercano	R+C a P+C
Robot más alejado al punto más cercano	R+A a P+C
Robot más cercano al punto más alejado	R+C a P+A
Robot con la distancia más cercana a la media de las distancias mínimas de los robots al punto más cercano	R+C(DM) a P+C
Robot con la distancia más cercana a la media de las distancias mínimas de los robots al punto más alejado	R+C(DM) a P+A
Robot con la menor distancia recorrida al punto más cercano	RmDR a P+C
Robot con la menor distancia recorrida al punto más alejado	RmDR a P+A

Fuente: Autor

- Puntos aleatorios – Robots aleatorios (PA-RA). En este escenario la distribución de los robots y los puntos objetivos en el espacio de trabajo es aleatoria (Ver Figura 55). Se realizaron seis (6) ejecuciones de código (*run*), cada una con 15 iteraciones (Ver ANEXO 17), obtenido los resultados de la Tabla 11. La distancia recorrida en cada run equivale a la sumatoria de las distancias recorridas por el sistema en las 15 iteraciones realizadas (Ver Figura 56), significando que la distancia total recorrida (DTR) es igual a la sumatoria de las 90 iteraciones ejecutadas. Para este escenario, el criterio con menor DTR corresponde al R+C a P+C, seguido muy de cerca por el R+A a P+C con una diferencia de tan solo el 1,06%.

Figura 55. Distribución de los elementos en el espacio de trabajo bajo el escenario PA-RA (criterio R+C a P+C, run n.º 6, iteración n.º 15)



Fuente: Autor

Tabla 11. Comparativa de los criterios de selección (PF-RA)

	Puntos aleatorios (6) / Robots aleatorios (6)					
	Runs					
	1		**2**		**3**	
Criterio	**Distancia Recorrida [$uds.$]**	**Posición**	**Distancia Recorrida [$uds.$]**	**Posición**	**Distancia Recorrida [$uds.$]**	**Posición**
R+C a P+C	279,5409	1	303,1445	1	296,1090	2
R+A a P+C	284,3203	2	309,4759	2	290,9269	1
R+C a P+A	608,3887	7	588,6837	7	596,9186	7
R+C(DM) a P+C	329,6832	3	327,4925	4	302,0718	3
R+C(DM) a P+A	593,9927	6	576,4737	6	581,1920	5
RmDR a P+C	340,1649	4	324,4314	3	311,6934	4
RmDR a P+A	580,5023	5	567,1990	5	588,8303	6

Fuente: Autor

Tabla 11. (Continuación)

	Puntos aleatorios (6) / Robots aleatorios (6)					
	Runs					
	4		**5**		**6**	
Criterio	**Distancia Recorrida [$uds.$]**	**Posición**	**Distancia Recorrida [$uds.$]**	**Posición**	**Distancia Recorrida [$uds.$]**	**Posición**
R+C a P+C	302,5889	1	291,9731	1	283,4632	1
R+A a P+C	307,7939	2	295,8552	2	287,3263	2
R+C a P+A	570,8277	6	563,7750	7	587,4481	7
R+C(DM) a P+C	324,1648	3	308,5915	4	309,6142	3
R+C(DM) a P+A	573,9070	7	558,1013	6	573,5215	6
RmDR a P+C	335,6474	4	303,2437	3	318,4571	4
RmDR a P+A	559,9593	5	553,8094	5	569,4932	5

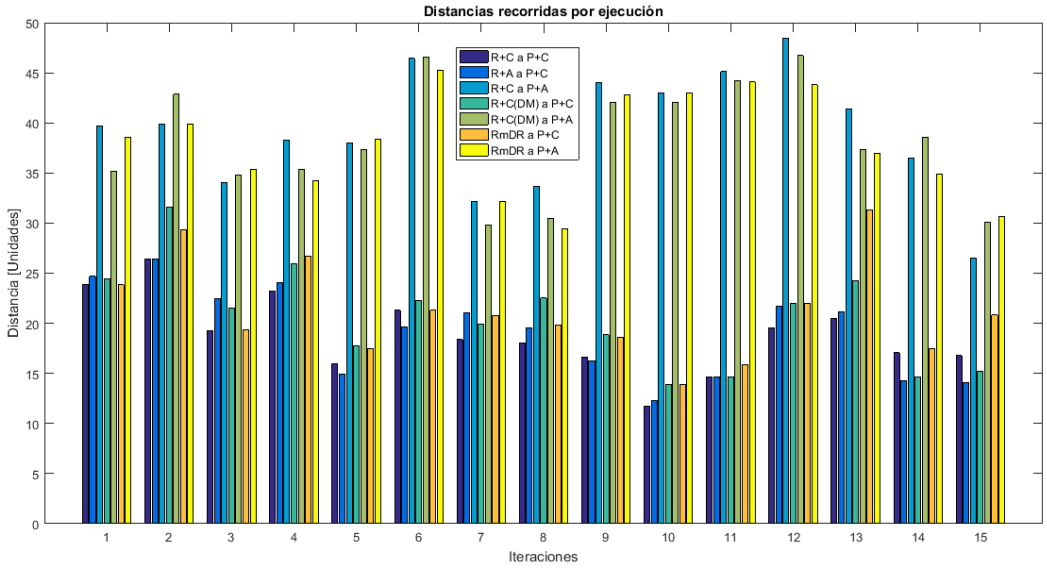
Fuente: Autor

Tabla 11. (Continuación)

Criterio	Total	
	Distancia Total Recorrida [$uds.$]	**Posición**
R+C a P+C	1.756,8196	1
R+A a P+C	1.775,6985	2
R+C a P+A	3.516,0418	7
R+C(DM) a P+C	1.901,6180	3
R+C(DM) a P+A	3.457,1882	6
RmDR a P+C	1.933,6379	4
RmDR a P+A	3.419,7935	5

Fuente: Autor

Figura 56. Distancias recorridas por el sistema bajo el escenario PA-RA (run n.º 6)



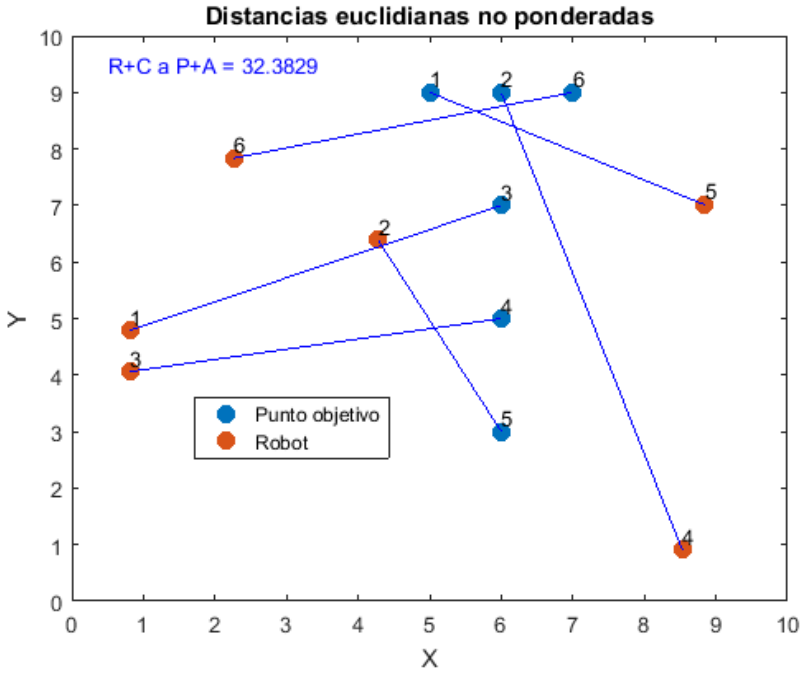
Fuente: Autor

- Puntos fijos – Robots aleatorios (PF-RA). En este escenario la distribución de las plataformas robóticas es aleatoria, mientras que la de los puntos es fija (Ver

Figura 57), lo que permite realizar formaciones de figuras predeterminadas, en este caso, tres letras (L, T y j) y tres figuras geométricas (triángulo, rectángulo y hexágono), cada una con 15 iteraciones (Ver ANEXO 18), obtenido los datos de la Tabla 12. Las distancias enlistadas en esa tabla presentan la misma metodología que las de la Tabla 11, con la diferencia que, para este escenario, el criterio con menor DTR fue el de R+A a P+C, seguido por el R+C a P+C con una diferencia de 1,16%.

Al comprar ambos escenarios, se puede observar que los criterios con menor distancia recorrida son aquellos en los que se asignan los puntos más cercanos, por ende, y dada la poca diferencia (0.1%) entre los criterios con menor distancia recorrida (R+C a P+C y R+A a P+C), se escoge el R+C a P+C como criterio para esta etapa, resultando en el proceso de asignación descrito en el ANEXO 19.

Figura 57. Distribución de los elementos en el espacio de trabajo bajo el escenario PF-RA (criterio R+C a P+A, letra T, iteración n.º 15)



Fuente: Autor

Tabla 12. Comparativa de los criterios de selección (PF-RA)

	Puntos fijos (6) / Robots aleatorios (6)					
	Letras					
	L		T		j	
Criterio	**Distancia Recorrida [$uds.$]**	**Posición**	**Distancia Recorrida [$uds.$]**	**Posición**	**Distancia Recorrida [$uds.$]**	**Posición**
R+C a P+C	311,3271	2	341,4402	1	311,3271	2
R+A a P+C	306,3243	1	342,0881	2	306,3243	1
R+C a P+A	506,2276	7	506,4866	6	506,2276	7
R+C(DM) a P+C	317,4042	4	356,2321	4	317,4042	4
R+C(DM) a P+A	499,7255	5	516,2469	7	499,7255	5
RmDR a P+C	314,3274	3	352,8533	3	314,3274	3
RmDR a P+A	502,4199	6	500,9154	5	502,4199	6

Fuente: Autor

Tabla 12. (Continuación)

	Puntos fijos (6) / Robots aleatorios (6)					
	Figuras geométricas					
	Hexágono		Triángulo		Rectángulo	
Criterio	**Distancia Recorrida [$uds.$]**	**Posición**	**Distancia Recorrida [$uds.$]**	**Posición**	**Distancia Recorrida [$uds.$]**	**Posición**
R+C a P+C	256,2543	1	266,2181	2	263,7913	2
R+A a P+C	256,5891	2	262,8829	1	252,3733	1
R+C a P+A	488,6044	7	562,2186	5	543,1164	6
R+C(DM) a P+C	264,0659	4	277,8092	3	278,0711	3
R+C(DM) a P+A	487,6271	6	565,2332	7	543,7628	7
RmDR a P+C	262,9379	3	287,8192	4	280,3816	4
RmDR a P+A	485,9952	5	562,3383	6	540,0328	5

Fuente: Autor

Tabla 12. (Continuación)

Criterio	Total	
	Distancia Total Recorrida [$uds.$]	**Posición**
R+C a P+C	1.715,6263	2
R+A a P+C	1.695,7502	1
R+C a P+A	3.057,3973	7
R+C(DM) a P+C	1.778,5627	3
R+C(DM) a P+A	3.055,0873	6
RmDR a P+C	1.779,3258	4
RmDR a P+A	3.036,1751	5

Fuente: Autor

3.3.6.2 Seguimiento de trayectoria

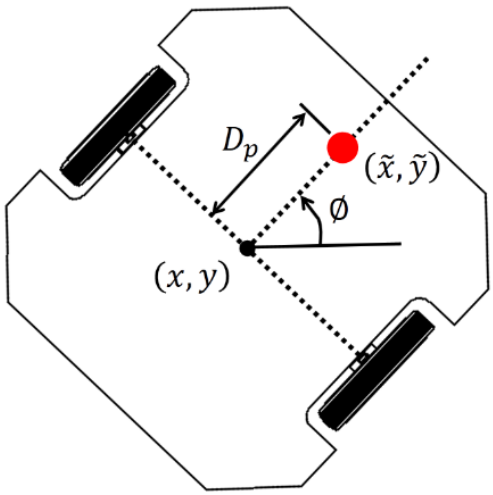
Es la segunda etapa a desarrollar y su propósito es el de garantizar que la plataforma robótica se desplace desde una posición inicial hasta una final, para este proyecto, la posición actual del robot hasta el punto objetivo respectivamente. Para realizar ese proceso, el Dr. Egerstedt propone una transformación al modelo del monociclo (Ver (46)) conocida como el "truco inteligente"[61]. Dicha propuesta consiste en prescindir de la orientación de la plataforma robótica añadiendo un punto imaginario al sistema a una distancia D_p del punto céntrico de la plataforma (Ver Figura 58). Este nuevo punto se describe geométricamente mediante las siguientes ecuaciones:

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} x + D_p \cdot \cos(\emptyset) \\ y + D_p \cdot \sin(\emptyset) \end{bmatrix} \tag{48}$$

Al derivarlas, se tiene que:

$$\begin{bmatrix}\dot{\tilde{x}}\\ \dot{\tilde{y}}\end{bmatrix} = \begin{bmatrix}\dot{x} - D_p \cdot \dot{\emptyset} \cdot \sin(\emptyset)\\ \dot{y} + D_p \cdot \dot{\emptyset} \cdot \cos(\emptyset)\end{bmatrix} \tag{49}$$

Figura 58. Truco inteligente



Fuente: Autor

Reemplazando (46) en (49):

$$\begin{bmatrix}\dot{\tilde{x}}\\ \dot{\tilde{y}}\end{bmatrix} = \begin{bmatrix}v \cdot \cos(\emptyset) - D_p \cdot \omega \cdot \sin(\emptyset)\\ v \cdot \sin(\emptyset) + D_p \cdot \omega \cdot \cos(\emptyset)\end{bmatrix}$$

$$\begin{bmatrix}\dot{\tilde{x}}\\ \dot{\tilde{y}}\end{bmatrix} = \begin{bmatrix}\cos(\emptyset) & -\sin(\emptyset)\\ \sin(\emptyset) & \cos(\emptyset)\end{bmatrix}\begin{bmatrix}v\\ D_p\omega\end{bmatrix}$$

$$\begin{bmatrix}\dot{\tilde{x}}\\ \dot{\tilde{y}}\end{bmatrix} = \begin{bmatrix}\cos(\emptyset) & -\sin(\emptyset)\\ \sin(\emptyset) & \cos(\emptyset)\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & D_p\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix} \tag{50}$$

Ahora, si se asume que se puede controlar instantáneamente ese nuevo punto, se tendrá que:

$$\begin{bmatrix}\cos(\emptyset) & -\sin(\emptyset)\\ \sin(\emptyset) & \cos(\emptyset)\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & D_p\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix} = \begin{bmatrix}x_{Obj} - x_R\\ y_{Obj} - y_R\end{bmatrix} \tag{51}$$

Donde la primera matriz es una matriz de rotación, x_{Obj} es la abscisa del punto objetivo, x_R la abscisa actual del robot, y_{Obj} la ordenada del punto objetivo, y_R la ordenada actual del robot, v y ω la velocidad lineal y angular de la plataforma robótica respectivamente. Para hallar esta dos últimas variables, se invierte (51), de modo que:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{D_p} \end{bmatrix} \begin{bmatrix} \cos(-\emptyset) & -\sin(-\emptyset) \\ \sin(-\emptyset) & \cos(-\emptyset) \end{bmatrix} \begin{bmatrix} x_{Obj} - x_R \\ y_{Obj} - y_R \end{bmatrix} \quad (52)$$

Conociendo v y ω e igualando las $\dot{x}$ y las $\dot{\emptyset}$ de (46) y (47), se pueden hallar las velocidades angulares de la plataforma robótica:

$$\omega_d = \frac{2v + L\omega}{2r} \quad (53)$$

$$\omega_i = \frac{2v - L\omega}{2r} \quad (54)$$

Siendo ω_d y ω_i los datos a enviar a la plataforma robótica para que esta se desplace del punto inicial al punto final dentro del espacio de trabajo.

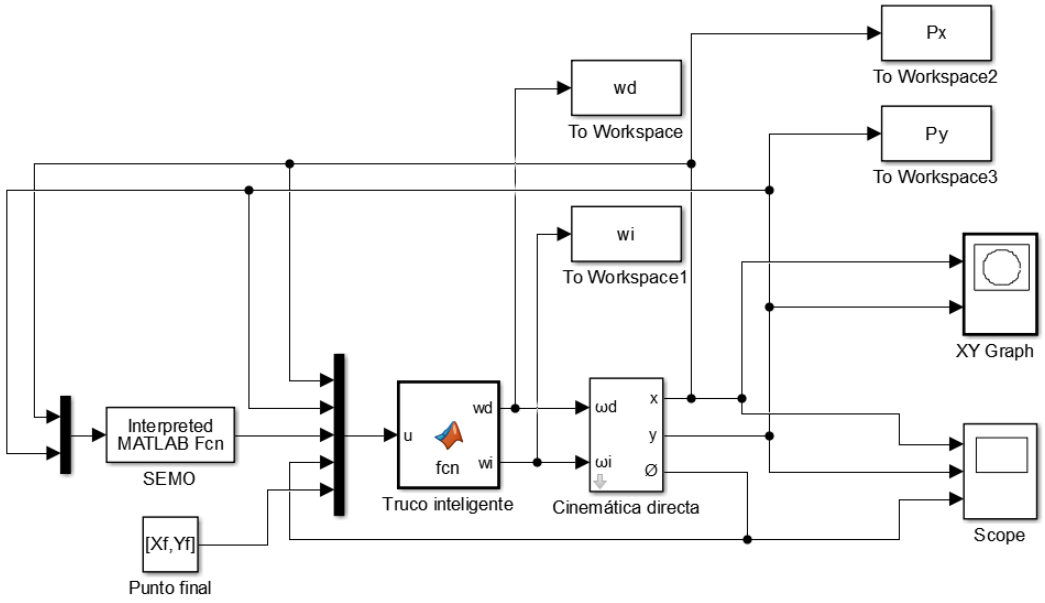
3.3.7 Simulación del sistema de control de formaciones

Para la simulación del sistema, se hizo uso del software MATLAB (R2016a) y su entorno de programación visual Simulink. La representación del modelo desarrollado se observa en la Figura 59. En él se incluyeron los siguientes elementos:

- Cinemática directa. Emula el comportamiento de la plataforma robótica ante las entradas de las velocidades angulares ω_d y ω_i (Ver Figura 60).

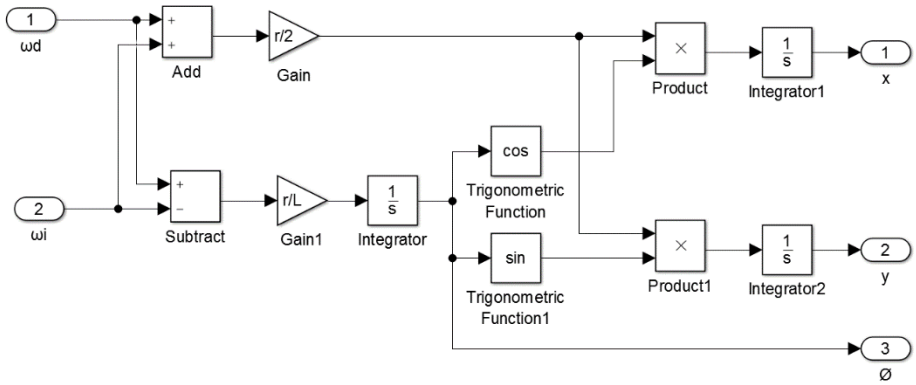
- Sistema de evasión de múltiples obstáculos (SEMO). Accionado en caso de presentarse algún obstáculo en la trayectoria suministrada (Ver ANEXO 20).
- El truco inteligente. Calcula las velocidades angulares requeridas para el desplazamiento de la plataforma robótica conforme al punto objetivo, el ángulo de evasión del SEMO y de las salidas de la cinemática directa (Ver ANEXO 21).

Figura 59. Modelo en Simulink del sistema de control de formaciones



Fuente: Autor

Figura 60. Bloque de la cinemática directa



Fuente: Autor

Para la simulación del sistema se planteó realizar dos trayectorias, una sin obstáculos y la otra en presencia de ellos. En ambos casos se emplearon los parámetros iniciales de la Tabla 13 en la cinemática directa.

Tabla 13. Parámetros iniciales de la cinemática directa

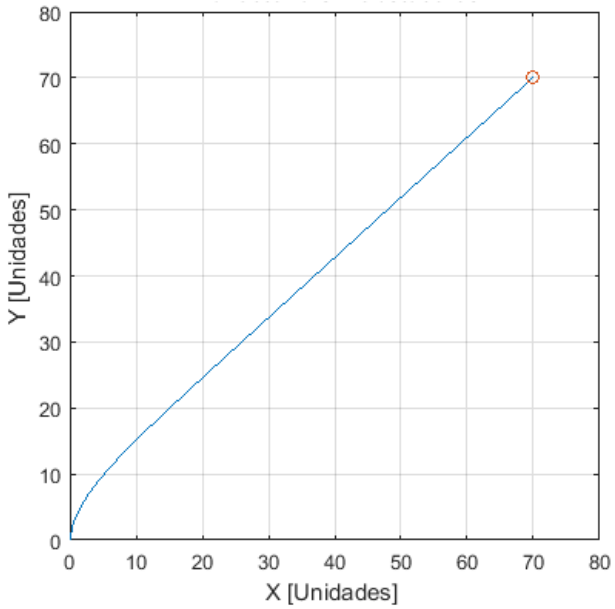
Parámetro	Valor
Componente en x	0
Componente en y	0
Orientación	90º

Fuente: Autor

3.3.7.1 Trayectoria sin obstáculos

En este caso la plataforma robótica se desplaza en el espacio de trabajo desde el punto inicial $(x = 0, y = 0)$ hasta el punto objetivo $(x = 70, y = 70)$ mediante la trayectoria descrita en la Figura 61.

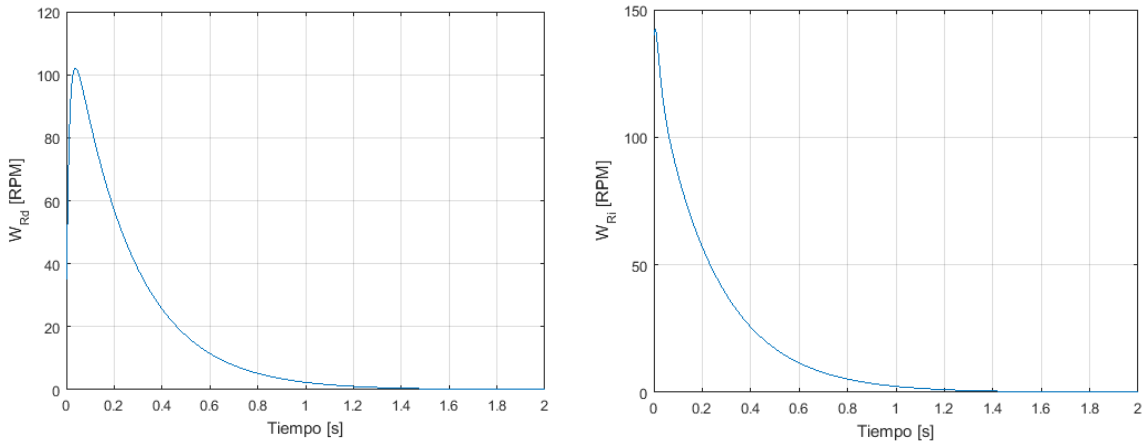
Figura 61. Trayectoria sin obstáculos



Fuente: Autor

Dado que la plataforma robótica se encuentra con una orientación inicial de 90º y el punto objetivo se ubica en un ángulo inferior a este, el robot debe girar en sentido horario para orientarse hacia su objetivo mientras avanza hacia él. Esta acción se verifica con la Figura 62, en donde se observa tanto la disminución de la velocidad angular de la rueda izquierda, como el incremento y la posterior reducción paulatina de la velocidad angular de la rueda derecha conforme la plataforma se orienta y acerca al punto objetivo.

Figura 62. Velocidades angulares de la plataforma robótica en una trayectoria sin obstáculos. (a) Rueda derecha. (b) Rueda izquierda



(a) (b)

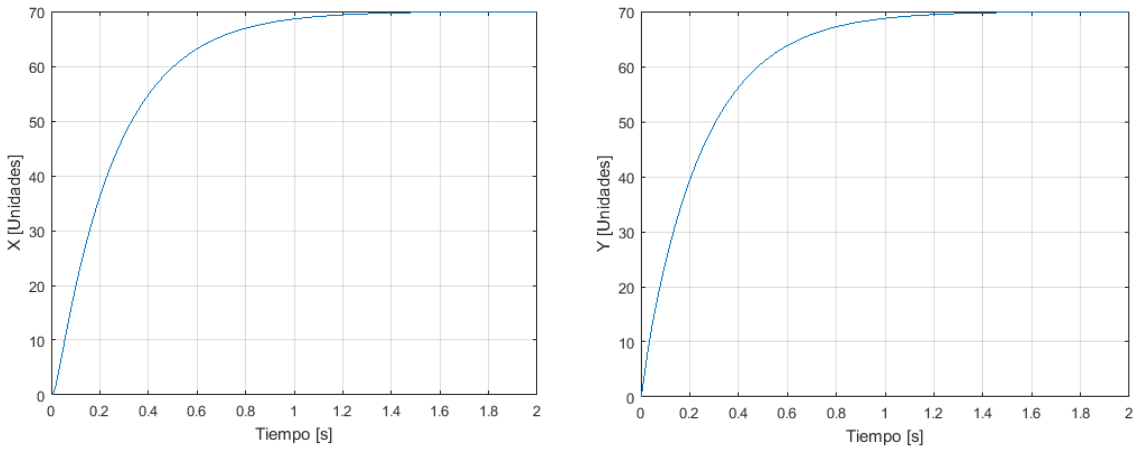
Fuente: Autor

En la Figura 63 se puede observar como varían las posiciones x e y del robot con respecto al tiempo desde su posición inicial ($x = 0, y = 0$) hasta el punto objetivo ($x = 70, y = 70$), comprobado la aproximación de la plataforma robótica al punto objetivo.

El cambio de la orientación de la plataforma con respecto al eje x se puede apreciar en la Figura 64. Se observa que mientras el robot se está desplazando, este está

girando para orientarse hacia el punto destino, una vez alineado con el objetivo, permanece constante en esa orientación.

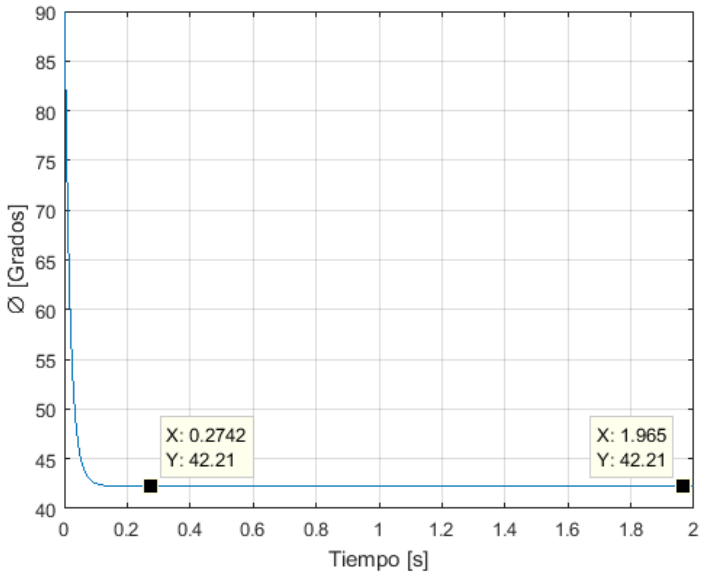
Figura 63. Posiciones x e y del robot con respecto al tiempo en una trayectoria sin obstáculos. (a) Posición en x. (b) Posición en y



(a) (b)

Fuente: Autor

Figura 64. Orientación de la plataforma robótica en una trayectoria sin obstáculos

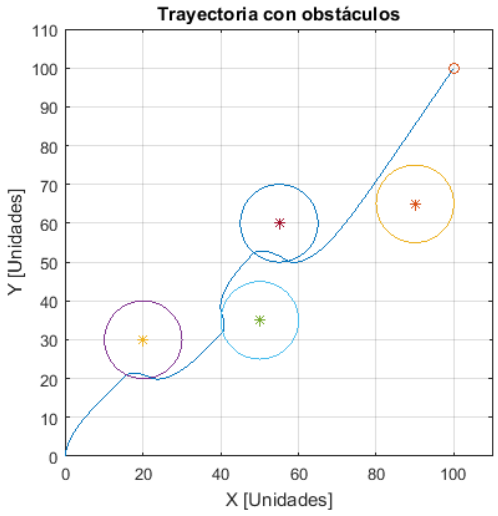


Fuente: Autor

3.3.7.2 Trayectoria con presencia de obstáculos

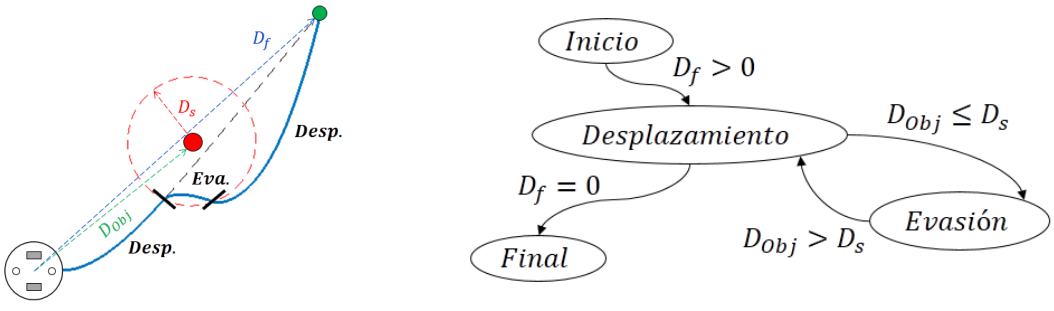
En este escenario el robot se desplaza desde el origen ($x = 0, y = 0$) hasta el punto objetivo ($x = 100, y = 100$) evadiendo aquellos obstáculos que se puedan presentar en su trayectoria. Este proceso se visualiza en la Figura 65, en donde los obstáculos son representados por asteriscos y su región de seguridad por las circunferencias que los rodean. La evasión se realiza por medio del SEMO, cuyo funcionamiento se describe en la Figura 66.

Figura 65. Trayectoria con presencia de obstáculos



Fuente: Autor

Figura 66. Principio de funcionamiento del SEMO. (a) Etapas. (b) Representación

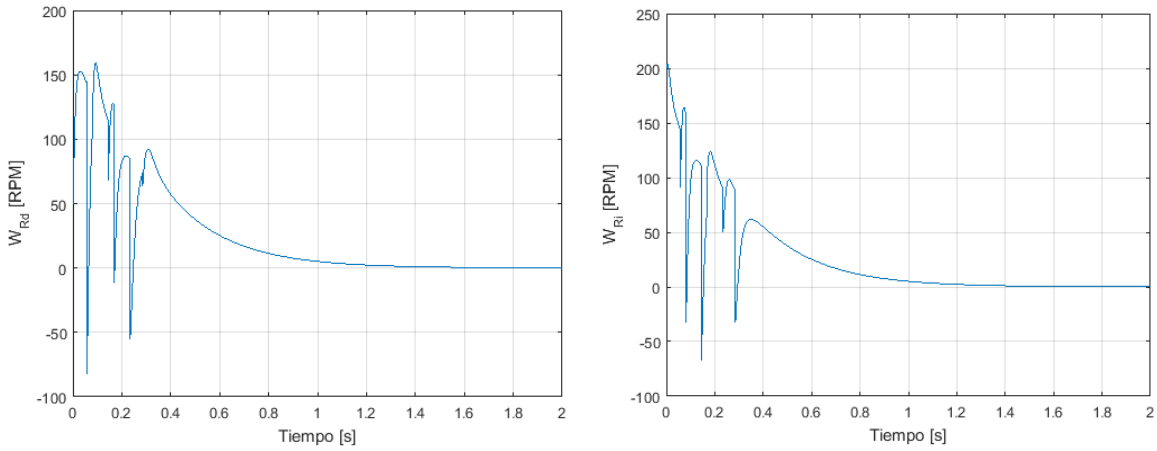


(a) (b)

Fuente: Autor

Las velocidades descritas por la plataforma robótica se muestran en la Figura 67. Los picos de velocidad que se observan son resultado de aplicar el SEMO y representan un cambio abrupto en la orientación del robot.

Figura 67. Velocidades angulares de la plataforma robótica en una trayectoria con presencia de obstáculos. (a) Rueda derecha. (b) Rueda izquierda



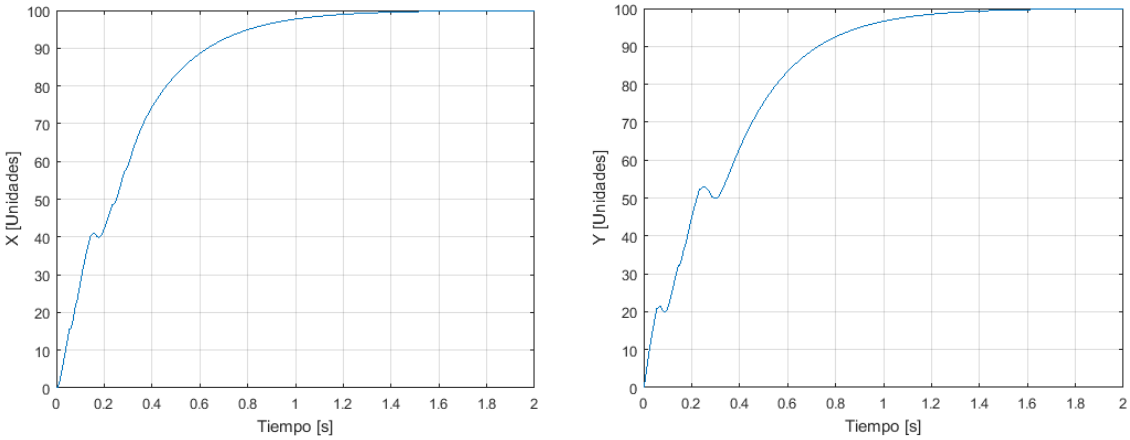
(a) (b)

Fuente: Autor

La variación de la posición del robot con respecto al tiempo se puede observar en la Figura 68. Las crestas presentes en ella representan un retroceso por parte de la plataforma en esa posición.

El comportamiento de la orientación de la plataforma se muestra en la Figura 69. En ella se aprecian los siete (7) cambios de orientación que realizó, cuatro (4) de desplazamiento y tres (3) de evasión, previos al alineamiento final con el punto objetivo.

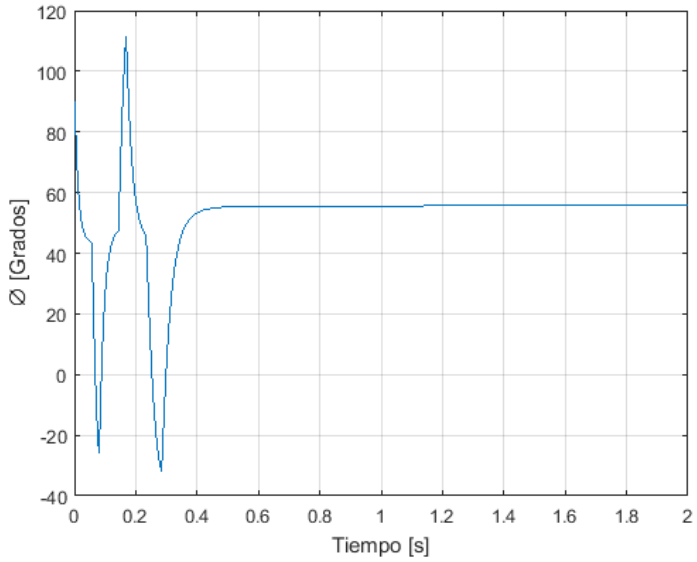
Figura 68. Posiciones x e y del robot con respecto al tiempo en una trayectoria con presencia de obstáculos. (a) Posición en x. (b) Posición en y



(a) (b)

Fuente: Autor

Figura 69. Orientación de la plataforma robótica en una trayectoria con obstáculos



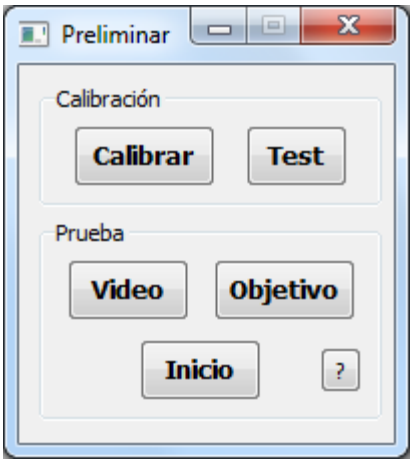
Fuente: Autor

4. DESARROLLO DEL SISTEMA DE VISIÓN ARTIFICIAL

En este capítulo se especificarán los cimientos del sistema de visión artificial finalmente implementado para este proyecto.

En la Figura 70 se observa la interfaz preliminar desarrollada en lenguaje python con la que se logró calibrar la cámara y realizar la umbralización. La interfaz consta de seis botones, el primero de ellos, “Calibrar”, se usa para establecer el valor del coeficiente de conversión entre la distancia en pixeles de la imagen y la distancia real en centímetros. El segundo, “Test”, permite verificar el coeficiente hallado mediante un patrón de medida fija, si es de igual medida, el coeficiente es fiable.

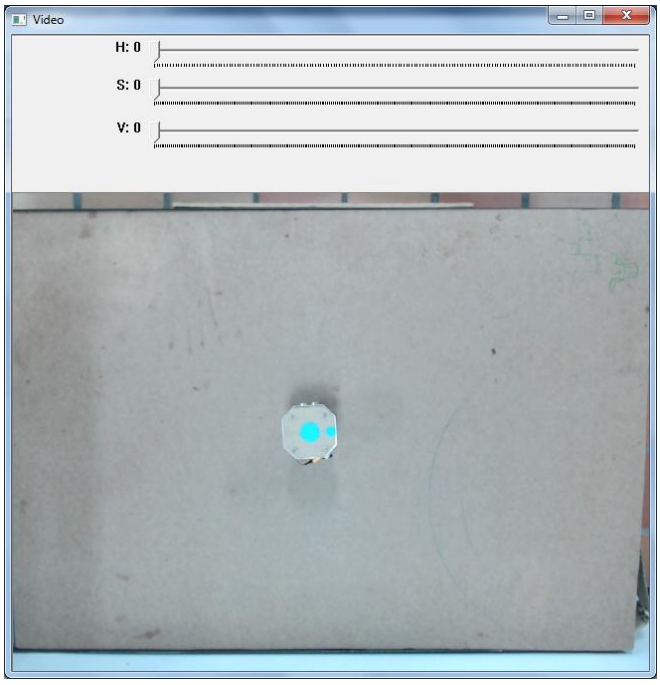
Figura 70. Interfaz gráfica preliminar



Fuente: Autor

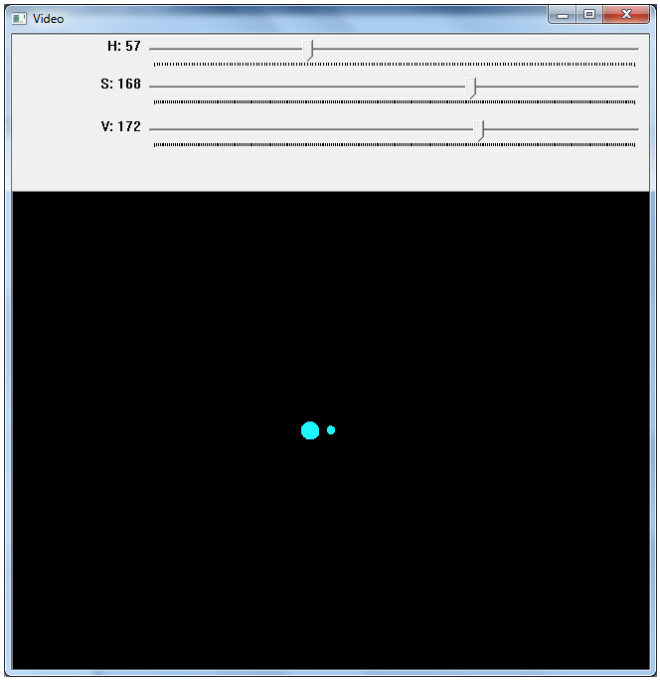
En la Figura 71 se aprecia el resultado de presionar el tercer botón, “Video”, con el que se inicia la captura de imagen de la cámara web en una ventana nueva con barras deslizantes. Estas barras permiten variar los parámetros del espacio de color HSV para, mediante de la umbralización, seleccionar el marcador de la plataforma robótica (Ver Figura 72).

Figura 71. Ventana de adquisición de imagen preliminar



Fuente: Autor

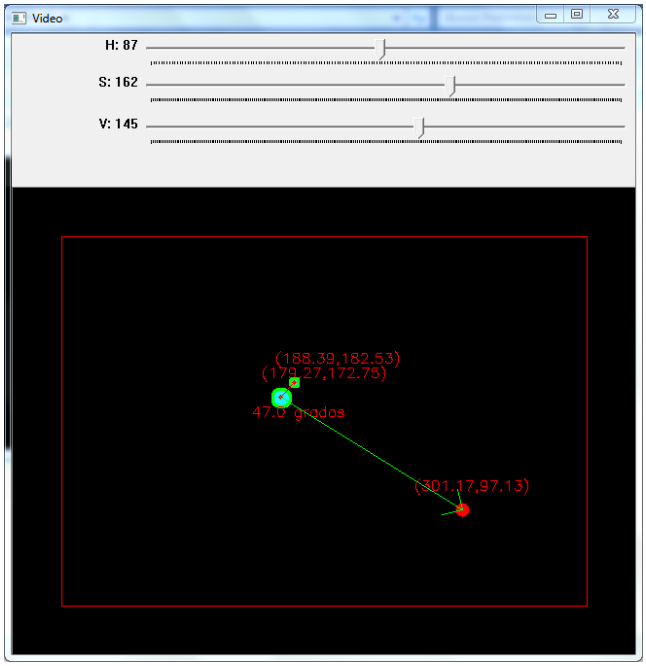
Figura 72. Ventana de adquisición preliminar con parámetros HSV variados



Fuente: Autor

El cuarto botón, “Objetivo”, agrega un punto virtual, de posición aleatoria, en el espacio de trabajo (Ver Figura 73). También activa la visualización de las coordenadas de los elementos visibles (punto objetivo, centroide de la plataforma y punto de orientación), la orientación del robot y el vector de asignación al objetivo.

Figura 73. Asignación del punto objetivo



Fuente: Autor

El quinto botón, “Inicio”, activa la comunicación serial con el Arduino maestro para enviar al robot las velocidades angulares calculadas por el software. El último botón, “?”, es un botón de ayuda que proporciona la lista con los comandos de teclado de la interfaz.

Una vez comprobado el funcionamiento de los códigos de calibración, umbralización, comunicación serial y de radiofrecuencia, se procede a realizar los ajustes pertinentes a la interfaz gráfica y a los códigos de Arduino (maestro y esclavos).

5. DISEÑO DE LA INTERFAZ GRÁFICA DE USUARIO

En este capítulo se desarrolla con más detalle la interfaz gráfica mencionada previamente, mejorando la experiencia de usuario y haciendo más robusta la adquisición de imágenes.

En la Figura 74 se observa la ventana principal de la interfaz gráfica de usuario. Consta de 5 secciones, las cuales se explican a continuación:

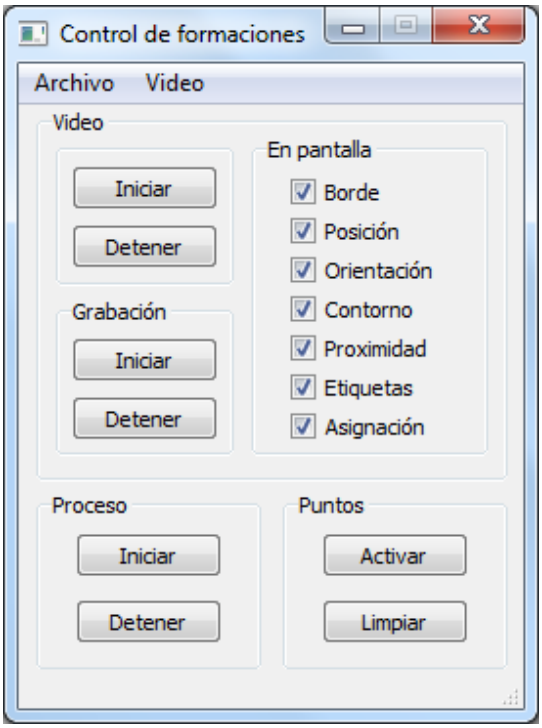
- Video. Consta de dos botones mediante los cuales se inicia o detiene la adquisición de imágenes en una ventana nueva.
- En pantalla. Ofrece una lista de chequeo por medio de la cual, el usuario puede escoger que parámetros visuales serán observables en pantalla.
- Grabación. Se compone de dos botones y permite realizar la grabación de video en formato AVI de la sesión en curso.
- Puntos. Activa la opción de agregar en el espacio de trabajo puntos virtuales mediante el uso del puntero del mouse. Si se desea borrar los puntos agregados, se puede hacer uso del botón “limpiar”.
- Proceso. Una vez asignados los puntos a los robots presentes en el espacio de trabajo, al presionar el botón “Iniciar”, se dará inicio a la comunicación entre el Arduino maestro y las plataformas robóticas. Al presionar “Detener”, dicho proceso se interrumpirá.

Con respecto a las dos opciones de menú, se tiene que:

- Archivo. Permite ingresar a la opción “Salir”, mediante la cual, se interrumpen todos los procesos activos, cerrando la interfaz gráfica.

- Video. Ofrece dos opciones de selección, “Segmentación” y “Calibración”, ventanas secundarias de la interfaz gráfica.

Figura 74. Venta principal de la interfaz gráfica de usuario



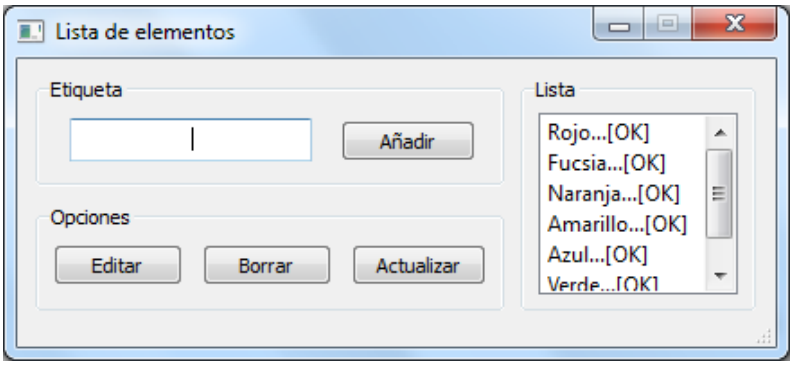
Fuente: Autor

La ventana secundaria “Segmentación” (Ver Figura 75) se emplea para agregar, modificar o eliminar, los parámetros del espacio de color HSV almacenados por la interfaz gráfica. Si se desea agregar los parámetros de un nuevo color, se debe ingresar un texto identificador del mismo, para luego agregar las variables en la ventana de tercer nivel “Editor de parámetros” (Ver Figura 76). De forma semejante a la Figura 72, se segmentará la imagen de acuerdo a los parámetros ingresados.

La ventana secundaría “Calibración” (Ver Figura 77) se emplea para calibrar la cámara web haciendo uso del patrón de calibración de la Figura 78, obteniendo el factor de conversión entre pixeles y centímetros mediante la siguiente expresión:

$$Px2cm = \frac{D_p}{D_{Px}} = \frac{13.6895873}{\sqrt{(Py_2 - Py_1)^2 + (Px_2 - Px_1)^2}} \quad (55)$$

Figura 75. Ventana secundaria de segmentación

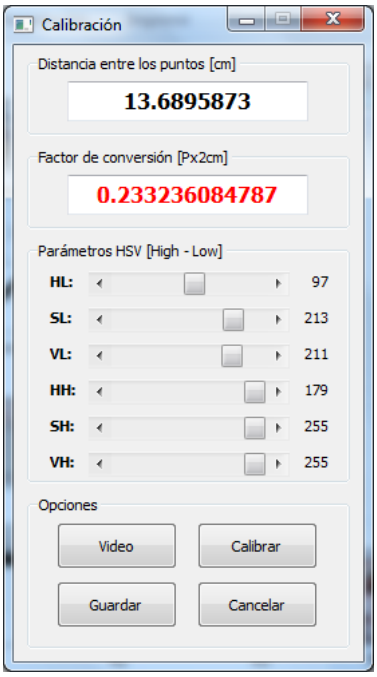


Fuente: Autor

Figura 76. Editor de parámetros

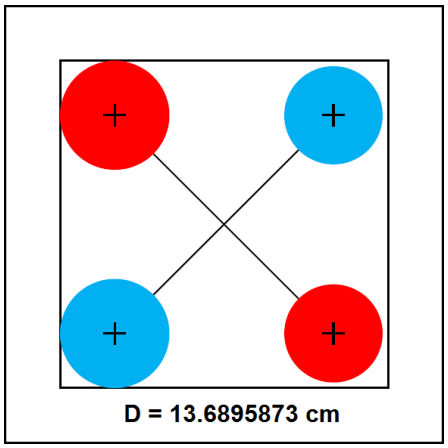
Fuente: Autor

Figura 77. Ventana secundaria de calibración de la cámara web



Fuente: Autor

Figura 78. Patrón de calibración de la cámara web



Fuente: Autor

En la Figura 79 se comprueba su uso al realizar la calibración de la cámara web por medio del color azul del patrón. Este proceso es manual y se debe repetir una vez las condiciones de ubicación de la cámara hayan variado.

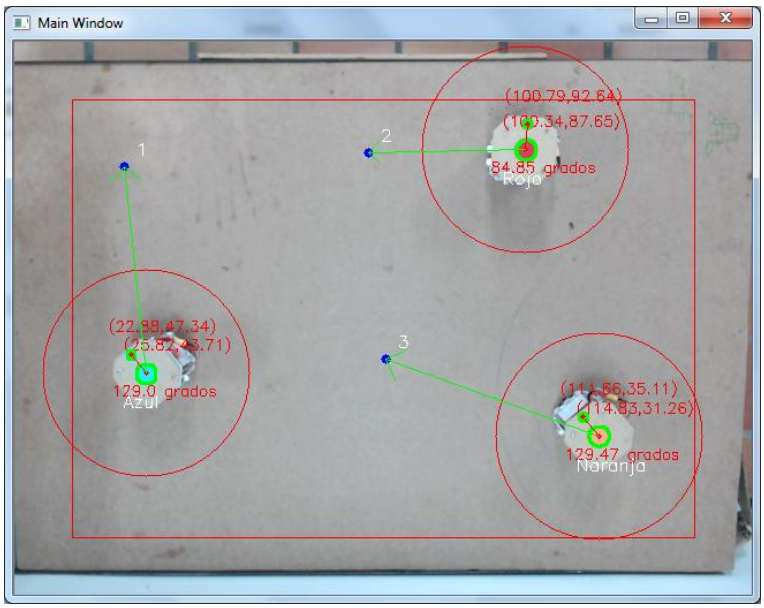
Figura 79. Calibración de la cámara web por color azul. (a) Ubicación del patrón de calibración. (b) Umbralización

(a)	(b)

Fuente: Autor

En la Figura 80 se puede observar la salida de video de la interfaz gráfica cuando todas las opciones de la de la lista de chequeo se encuentran activas.

Figura 80. Salida de video con todas las opciones activas



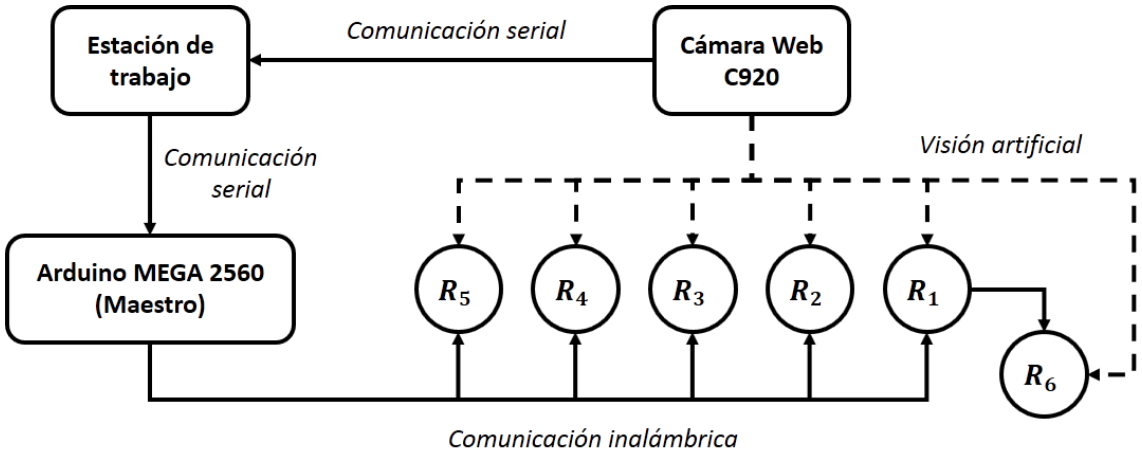
Fuente: Autor

6. IMPLEMENTACIÓN DEL SISTEMA DE CONTROL DE FORMACIONES EN LAS PLATAFORMAS DESARROLLADAS

En el presente capítulo se describe el proceso de enlace e implementación entre la plataforma robótica, el sistema de visión artificial, el de comunicación inalámbrica y el de comunicación serial.

En la Figura 81 se observa el lazo cerrado del sistema de control de formaciones, dónde el eslabón principal es la estación de trabajo (computador portátil). Es en ella donde se procesan las imágenes, para posteriormente enviar los datos calculados a las plataformas robóticas mediante el Arduino maestro.

Figura 81. Implementación del sistema de control de formaciones



Fuente: Autor

En la estación de trabajo se encuentra la interfaz gráfica desarrollada en lenguaje de programación python (Ver ANEXO 22). Esta interfaz le permite al usuario umbralizar las imágenes adquiridas por la cámara web según los parámetros HSV (alto y bajo) ingresados, permitiéndole diferenciar cada plataforma robótica según su color. Esos colores hacen de parámetros identificadores, además de

proporcionar la ubicación y orientación en el espacio de trabajo. Dichos datos son adquiridos por la cámara e introducidos en el código de trayectorias. Ese código calcula las velocidades angulares de cada robot según sea su punto objetivo, para luego enviarlas al Arduino maestro mediante comunicación serial. Así mismo, el Arduino envía esas velocidades a los robots mediante radiofrecuencia empleando una topología de árbol de tal forma que, el maestro es padre de las plataformas 1 al 5 y abuelo de la plataforma 6, quien es hijo de la plataforma 1.

El modelo de comportamiento del sistema de control de formaciones se describe en la Figura 82. En la Tabla 14 se detallan las salidas y entradas involucradas en el proceso:

$$\frac{Entradas}{Salidas} = \frac{R, P, O, C_1, C_2}{\omega_d, \omega_i} \tag{56}$$

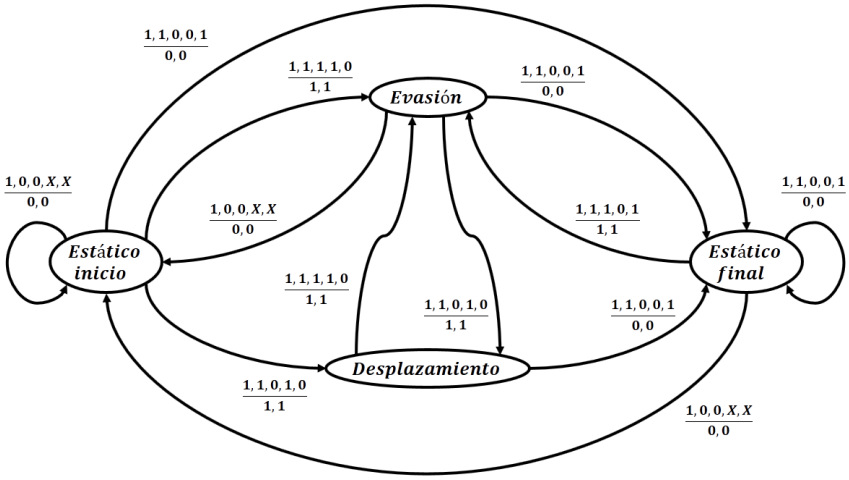
Tabla 14. Entradas y salidas de la máquina de estados del sistema de control de formaciones

Variable	**Significado**
R	Presencia de robot
P	Presencia de punto objetivo
O	Presencia de obstáculo
C_1	Condición $D_f > 0$
C_2	Condición $D_f = 0$
ω_d	Velocidad angular de la rueda derecha
ω_i	Velocidad angular de la rueda izquierda

Fuente: Autor

Donde D_f es la distancia euclidiana de la plataforma robótica al punto objetivo.

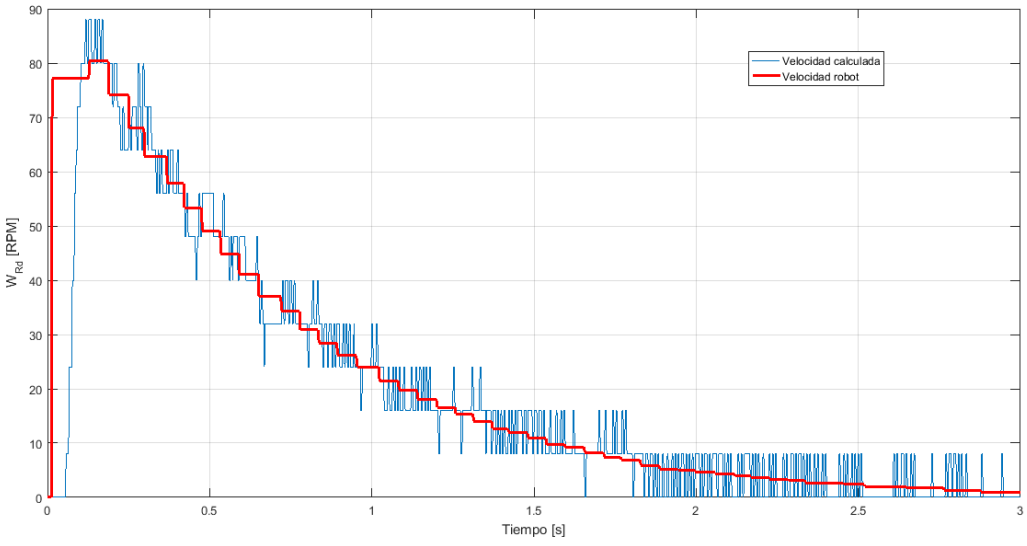
Figura 82. Máquina de estados del sistema de control de formaciones



Fuente: Autor

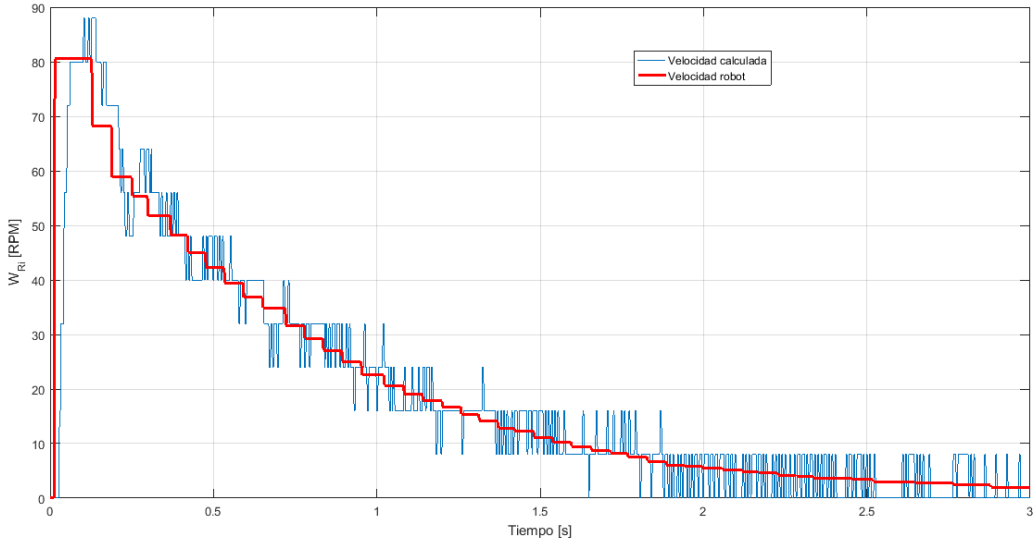
Después de realizar la implementación en los arduinos maestro (Ver ANEXO 23) y esclavos (Ver ANEXO 24), se procedió a comparar las velocidades angulares calculadas por el software con las desarrolladas por la plataforma robótica, dando como resultado las Figura 83 y Figura 84.

Figura 83. Comparación de las velocidades angulares de la rueda derecha



Fuente: Autor

Figura 84. Comparación de las velocidades angulares de la rueda izquierda



Fuente: Autor

Los picos presentes en las velocidades angulares calculadas se deben al alto grado de sensibilidad del modelo ante los cambios de orientación de la plataforma robótica, no obstante, la plataforma realiza la trayectoria correctamente.

7. VALIDACIÓN DEL SISTEMA

En este capítulo se realiza la divulgación de los resultados de la implementación del sistema de control de formaciones en las seis (6) plataformas robóticas construidas. Para ello, se mostrarán a continuación las imágenes del antes y el después, del desplazamiento de las plataformas robóticas. Se escogieron tres formaciones a realizar, una línea recta, un rectángulo y un triángulo.

7.1 LÍNEA RECTA

Se ubican los robots en la posición inicial de la Figura 85 en espera del inicio de la comunicación con el Arduino maestro. Se dejan activas todas las opciones de visualización de la lista de chequeo. Les toma 40.31 [s] el terminar la tarea asignada (Ver Figura 86).

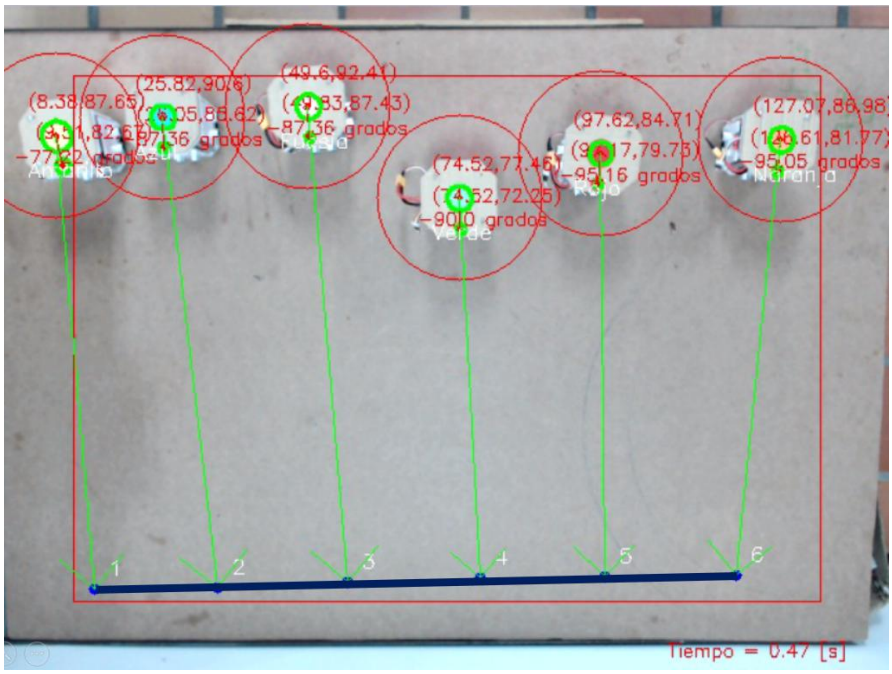
7.2 RECTÁNGULO

Se ubican los robots en la posición inicial de la Figura 87 en espera del inicio de la comunicación con el Arduino maestro. Se deja activa únicamente la opción de asignación en las opciones de visualización de la lista de chequeo. Les toma 59.46 [s] el terminar la tarea asignada (Ver Figura 88).

7.3 TRIÁNGULO

Se ubican cinco (5) robots en la posición inicial de la Figura 89 en espera de instrucciones. Se desactivan todas las opciones de visualización de la lista de chequeo. A los 14 [s] de operación se ingresa la sexta (6) plataforma robótica (Ver Figura 90). Les toma 35.25 [s] el terminar la tarea asignada (Ver Figura 91).

Figura 85. Línea recta. Posición inicial



Fuente: Autor

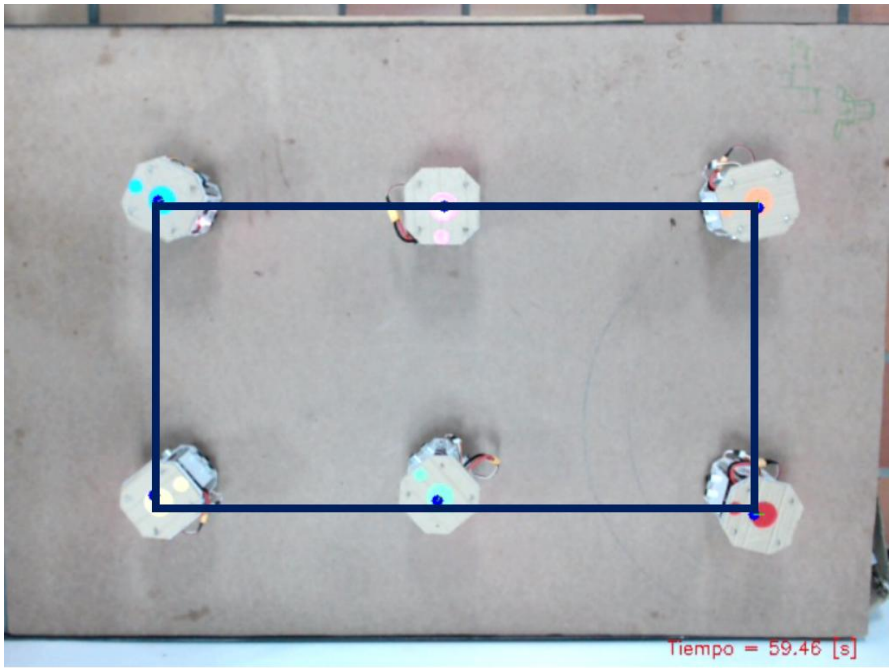
Figura 86. Línea recta. Posición final

Fuente: Autor

Figura 87. Rectángulo. Posición inicial

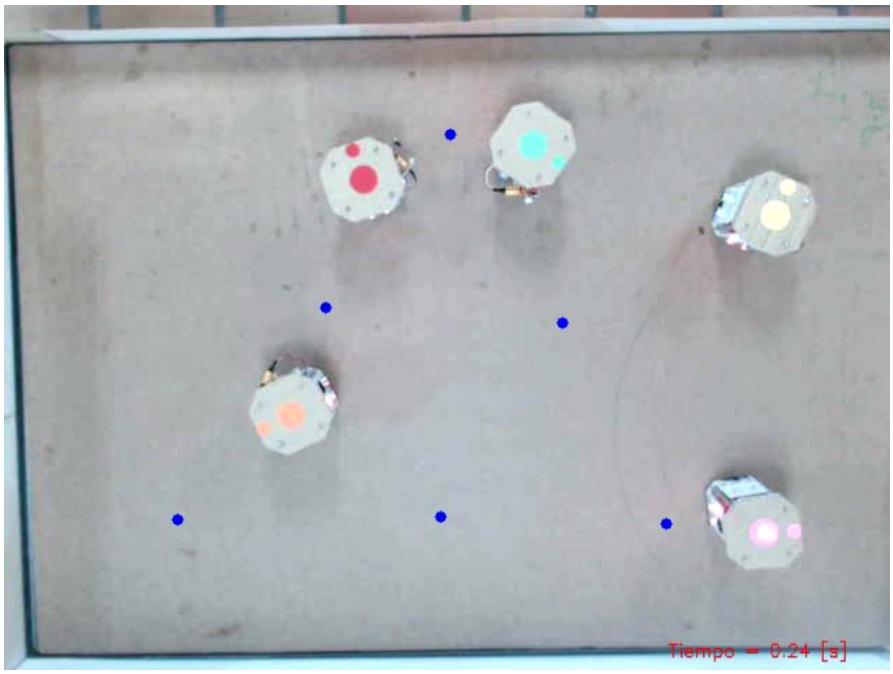
Fuente: Autor

Figura 88. Rectángulo. Posición final



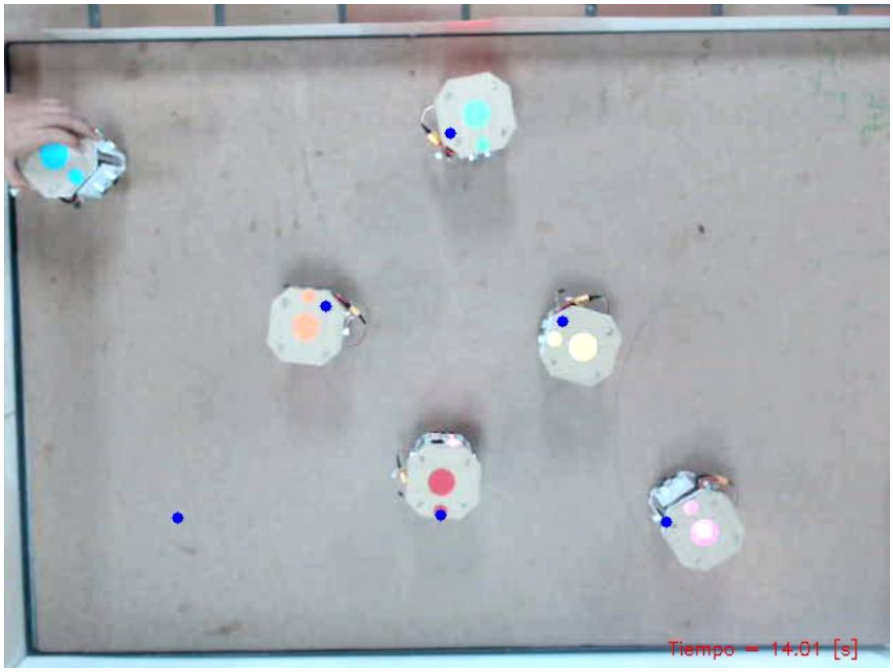
Fuente: Autor

Figura 89. Triángulo. Posición inicial



Fuente: Autor

Figura 90. Triángulo. Ingreso de la sexta (6) plataforma al espacio de trabajo



Fuente: Autor

Figura 91. Triángulo. Posición final

Fuente: Autor

8. CONCLUSIONES

A continuación, se procede a enlistar las conclusiones derivadas de este trabajo, así como también, las recomendaciones y trabajos futuros del mismo:

- Se diseñó una plataforma robótica diferencial de estructura modular con las especificaciones técnicas mencionadas en los capítulos anteriores, esto es, consta de una área reducida dentro del espacio de trabajo; los elementos electrónicos que la componen son de acople no permanente; el trabajo lo realiza en un entorno controlado; el desplazamiento lo desarrolla sobre una superficie plana; el voltaje y corriente de operación son de 12.6 [V] y 913.2 [mA], respectivamente; el peso total es de 726.55 [g] y su tiempo de operación es de aproximadamente 120 minutos.
- Se seleccionó la instrumentación que mejor se acoplaba a los requerimientos de la plataforma robótica mediante matrices de ponderación. En ellas se evaluaron las especificaciones técnicas de los componentes electrónicos comparados según su categoría (comunicación, procesamiento, sensórica, etc.).
- Se seleccionó un sistema de comunicación inalámbrico por radiofrecuencia bajo una topología de árbol que, para este caso, consta de un nodo maestro y seis nodos esclavos. Este sistema se implementó mediante el módulo NRF24L01, también seleccionado a través de una matriz de ponderación.
- Se estableció el “truco inteligente” del Dr. Magnus Egerstedt como la estrategia de control aplicada al control de formación visual desarrollada en el proyecto.
- Se realizaron diversas simulaciones en el software comercial MATLAB®, mediante las cuales se comprobó que 1) el criterio de asignación con el que el sistema multiagente recorre la menor distancia es el del robot más cercano al punto más cercano (R+C a P+C) y 2) la estrategia del Dr. Magnus permite que las plataformas robóticas se desplacen en su espacio de trabajo y realicen la representación de las figuras geométricas indicadas.

- Se diseñó una interfaz gráfica de usuario desde donde el operario puede monitorear y manipular en pantalla la posición y orientación de las plataformas robóticas. Se habilitó una lista de chequeo mediante la cual el usuario puede escoger que parámetros visuales serán observables en pantalla, además de los sistemas de calibración de la cámara web y de configuración de los parámetros de umbralización del software. El usuario también tiene la opción de grabar en video (formato AVI) la sesión de trabajo.
- Se realizó la validación del sistema de control de formación visual con seis (6) plataformas robóticas, realizando diversas representaciones de figuras geométricas y algunas pruebas individuales.
- El precio de adquisición de las plataformas robóticas disponibles hoy día en el mercado, puede superar hasta 18 veces el costo de producción de la plataforma desarrollada en este proyecto, haciendo de esta última, una opción económicamente viable en caso de necesitarse más unidades.
- Se probó accidentalmente los elementos electrónicos de seguridad, esto, sin presentarse ningún daño al sistema eléctrico o algún componente de la plataforma, demostrando así la confiabilidad que eso supone al sistema.
- Se obtuvo la curva de descarga de la batería LiPo de tres (3) celdas (3S) de 2200 [mAh] y 11.1 [V], mediante la cual se estableció el voltaje de 3.6 [V] como criterio de desconexión de la alimentación de la plataforma robótica.
- Dada la geometría y la ubicación del espacio de trabajo del proyecto en el laboratorio de oleoneumática de la Universidad, se presentaron incidentes que malograron en cierta medida la superficie de trabajo de las plataformas. Aun así, el sistema de control de formaciones realiza las tareas indicadas.
- Si bien, los sensores ultrasónicos implementados en las plataformas representan un apoyo a la cámara web cuando se tiene una única plataforma en el espacio de trabajo, estos no son funcionales en presencia de varios robots dado al fenómeno de diafonía desarrollado en ese escenario.

- Los módulos de radiofrecuencia ofrecen una comunicación aceptable en una red de hasta 5 nodos; sobrepasar ese número representa un incremento en el envío de la trama con las velocidades angulares.
- Durante el desarrollo del proyecto, la primera plataforma desarrollada se empleó en la promoción de la carrera de Ingeniería Mecatrónica en dos escenarios escolares, demostrando el potencial de la misma en diferentes escenarios al investigativo.
- Se propone migrar del sistema de reconocimiento por color a uno por código QR, eliminando la dependencia de un ambiente con iluminación controlada.
- Dado el percance en la comunicación de las plataformas robóticas con el nodo central, se propone cambiar los módulos de radiofrecuencia por la segunda alternativa en el ANEXO 7, el módulo Wifi ESP8266.

BIBLIOGRAFÍA

[1] BROOKS, R. A. (1986). A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, RA-2(1): 14-23, Marzo.

[2] YAMAGUCHI H.; T. Arai; G. Beni (2001). A distributed control scheme for multiple robotic vehicles to make group formations. Robotics and Autonomous Systems, 36, pp. 125-147.

[3] FIERRO R.; P. Song; A. Das; V. Kumar (2001). Cooperative control of robot formations. Kluwer Series Applied Optimization, Marzo.

[4] KELLY, Rafael; et al.. Control de una pandilla de robots móviles para el seguimiento de una constelación de puntos objetivo [En línea] Octubre de 2004. VI Congreso Mexicano de Robótica. Disponible en: <http://ebanov.inaut.unsj.edu.ar/publicaciones/Ca1659_04.pdf> [Citado el 26 de marzo de 2018].

[5] MACDONALD, Edward. MULTI-ROBOT ASSIGNMENT AND FORMATION CONTROL. Georgia, 2011, 76 h. Trabajo de grado (Masters of Science). Georgia Institute of Technology. School of Electrical and Computer Engineering. Disponible en: <https://smartech.gatech.edu/bitstream/handle/1853/41200/macdonald_edward_a_201108_mast.pdf> [Citado el 26 de marzo de 2018].

[6] GRITSLAB. Multi-Robot Assignment and Formation Control [videograbación] [En línea] 6 de mayo de 2011. Disponible en: <https://www.youtube.com/watch?v=se318w2LXD0> [Citado el 26 de marzo de 2018].

[7] MADERER, Jason. ROBOTARIUM : Robots for Everyone [En línea] 15 de agosto de 2017. Disponible en: <http://www.news.gatech.edu/features/robotarium-robotics-lab-accessible-all> [Citado el 26 de marzo de 2018].

[8] PICKEM, Daniel; et al.. The Robotarium: A remotely accessible swarm robotics research testbed [En línea] Disponible en: <http://robohub.org/the-robotarium-a-

remotely-accessible-swarm-robotics-research-testbed/> [Citado el 26 de marzo de 2018].

[9] VALLEJO, Marcela; OCHOA, John; JIMÉNEZ, Jovani. Sistemas multi-agentes robóticos: Revisión de metodologías [En línea] Disponible en: <https://revistas.unal.edu.co/index.php/avances/article/view/20499> [Citado el 26 de marzo de 2018].

[10] GONZÁLEZ, Enrique. Robótica cooperativa. Experiencias de sistemas multiagente (SMA). Primera Edición. Editorial Pontificia Universidad Javeriana, Colombia, noviembre 2012. ISBN: 978-958-716-586-9.

[11] MARTÍNEZ, John; VALLEJO, Margarita. Comparación de estrategias de navegación colaborativa para robótica móvil [En línea] Disponible en: <http://repositorio.autonoma.edu.co/jspui/handle/11182/935> [Citado el 26 de marzo de 2018].

[12] MOLINA, Manuel; RODRÍGUEZ, Edgar. FLOTILLA DE ROBOTS PARA TRABAJOS EN ROBÓTICA COOPERATIVA. Trabajo de grado (Ingeniero Mecatrónico). Bogotá D.C.: Universidad Militar Nueva Granada. Facultad de Ingeniería. Programa de Ingeniería Mecatrónica, 2014.

[13] ROBOCUP. RoboCupSoccer [En línea] Disponible en: <http://www.robocup.org/domains/1> [Citado el 26 de marzo de 2018].

[14] CUARTAS, Enrique. La selección Colombia de futbol robótico que estará en RoboCup 2016 [En línea] Disponible en: <http://www.enter.co/cultura-digital/colombia-digital/la-seleccion-colombia-de-futbol-robotico-que-estara-en-robocup-2016/> [Citado el 26 de marzo de 2018].

[15] IEEE. Miembros de IEEE Colombia participan en la RoboCup Brasil 2014 [En línea] Disponible en: <http://www.ieee.org.co/noticia.php?id=96> [Citado el 26 de marzo de 2018].

[16] ZUNT, Dominic. Who did actually invent the word "robot" and what does it mean? [En línea] Disponible en: <http://web.archive.org/web/20150415062618/http://capek.misto.cz/english/robot.html> [Citado el 26 de marzo de 2018].

[17] MATHIA, Karl. Robotics for Electronics Manufacturing : Principles and Applications in Cleanroom Automation. Primera Edición. Cambridge : Cambridge University Press, 2010. Pág. 8. ISBN 978-0-521-87652-0.

[18] GARCÍA, Cándido; OYARZABAL, Rosa. What is a Robot under EU Law? [En línea] Disponible en: <https://www.globalpolicywatch.com/2017/08/what-is-a-robot-under-eu-law/> [Citado el 26 de marzo de 2018].

[19] ZIELINSKA, Teresa; et. al.. Robotics: Concepts, Methodologies, Tools, and Applications. History of Service Robots. Primera Edición. Editorial IGI Global, Pennsylvania, EEUU, 2013. Pág. 2. ISBN: 978-146-664-607-0.

[20] TZAFESTAS, Spyros. Introduction to Mobile Robot Control. Primera Edición. Editorial Elsevier Insights, London, 2014. Pág. 1. ISBN: 978-0-12-417049-0.

[21] ZHANG, Houxiang. Mobile Robotics : Mobile robot classification continued [En línea] Disponible en: <https://tams.informatik.uni-hamburg.de/lehre/2010ss/seminar/ir/PDF/MobilerobotLecture3_Review%20on%20mobile%20robot.pdf> [Citado el 26 de marzo de 2018].

[22] TZAFESTAS, Spyros. Introduction to Mobile Robot Control. Primera Edición. Editorial Elsevier Insights, London, 2014. Pág. 15. ISBN: 978-0-12-417049-0.

[23] Ibíd., p. 31.

[24] KUO, Benjamin. SISTEMAS DE CONTROL AUTOMÁTICO. Séptima edición. Editorial Prentice Hall, México, 1996. Pág. 9. ISBN: 968-880-723-0.

[25] DEWESoft. PID Control [En línea] Disponible en: <https://www.dewesoft.com/pro/course/pid-control-53> [Citado el 26 de marzo de 2018].

[26] MAZZONE, Virginia. Controladores PID [En línea] Disponible en: <http://www.eng.newcastle.edu.au/~jhb519/teaching/caut1/Apuntes/PID.pdf> [Citado el 26 de marzo de 2018].

[27] RIVEROS, Adriana; SOLAQUE, Leonardo. Formación de robots móviles mediante el uso de controladores. En: Ing. USBMed. Julio-diciembre, 2013. Vol. 4, n.º 2, p. 63. ISBN: 2027-5846.

[28] SABIA. Visión artificial e interacción sin mandos [En línea] Disponible en: <http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/VisionArtificial/index.html> [Citado el 26 de marzo de 2018].

[29] MOLINERO, Gregorio. Segmentación de imágenes en color basada en el crecimiento de regiones. Pág. 5. [En línea] Disponible en: <http://bibing.us.es/proyectos/abreproy/11875/fichero/Proyecto+Fin+de+Carrera%252F3.Espacios+de+color.pdf> [Citado el 26 de marzo de 2018].

[30] Ibíd., p. 6.

[31] GÓMEZ, Aure. Modelos, Espacios y Perfiles de Color : MODELO RGB [En línea] Disponible en: <http://www.auregomez.com/tutoriales/modelos-espacios-y-perfiles-de-color/> [Citado el 26 de marzo de 2018].

[32] CUEVAS, Erik; ZALDÍVAR, Daniel; PÉREZ, Marco. Procesamiento digital de imágenes con MATLAB y Simulink. Primera edición: Alfaomega Grupo Editor, México, septiembre 2010. Pág. 454. ISBN: 978-607-707-030-6.

[33] TRAUMABOT. Colour Point Cloud detection and extraction based on HSV colour space [En línea] Disponible en: <http://traumabot.blogspot.com.co/2013/08/colour-point-cloud-detection-and.html> [Citado el 26 de marzo de 2018].

[34] QUEVEDO, Yeimy. Filtrado Espacial en Imágenes [En línea] Disponible en: <https://es.scribd.com/document/95955212/Filtrado-Espacial-en-Imagenes> [Citado el 26 de marzo de 2018].

[35] CUEVAS, Erik; ZALDÍVAR, Daniel; PÉREZ, Marco. Procesamiento digital de imágenes con MATLAB y Simulink. Primera edición: Alfaomega Grupo Editor, México, septiembre 2010. Pág. 138 – 139. ISBN: 978-607-707-030-6.

[36] Ibíd., p. 76 – 77.

[37] OPENCV. Morphological Transformations [En línea] Disponible en: <https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html> [Citado el 26 de marzo de 2018].

[38] Ibíd.

[39] Ibíd.

[40] CUEVAS, Erik; ZALDÍVAR, Daniel; PÉREZ, Marco. Procesamiento digital de imágenes con MATLAB y Simulink. Primera edición: Alfaomega Grupo Editor, México, septiembre 2010. Pág. 403. ISBN: 978-607-707-030-6.

[41] Ibíd., p. 404.

[42] Ibíd., p. 406.

[43] Ibíd., p. 407 – 408.

[44] TDROBÓTICA. Ruedas Pololu 32×7mm - Negras [En línea] Disponible en: <http://tdrobotica.co/ruedas-pololu-327mm-negras/260.html> [Citado el 2 de abril de 2018].

[45] TDROBÓTICA. Micromotor 100:1 con eje extendido / 2.2 kg-cm / 320 rpm [En línea] Disponible en: <http://tdrobotica.co/micromotor-1001-con-eje-extendido22-kg-cm320-rpm/387.html> [Citado el 2 de abril de 2018].

[46] TDROBÓTICA. Encoder magnético [En línea] Disponible en: <http://tdrobotica.co/kit-encoder-magnetico-para-micromotor-con-eje-extendido/114.html> [Citado el 2 de abril de 2018].

[47] TDROBÓTICA. Sensor ultrasónico (HC-SR04) [En línea] Disponible en: <https://www.ardobot.com/productos/sensores/distancia-presencia-huellas-y-corriente/sensor-ultrasonido-hc-sr04.html> [Citado el 2 de abril de 2018].

[48] TDROBÓTICA. Arduino Mega 2560 R3 [En línea] Disponible en: <https://www.ardobot.com/arduino-mega-2560-r3.html> [Citado el 2 de abril de 2018].

[49] TDROBÓTICA. Módulo L298N [En línea] Disponible en: <http://tdrobotica.co/modulo-driver-l298n/543.html> [Citado el 2 de abril de 2018].

[50] TDROBÓTICA. Módulo RF NRF24L01 [En línea] Disponible en: <http://tdrobotica.co/modulo-radiofrecuencia-nrf24l01-24ghz/654.html> [Citado el 2 de abril de 2018].

[51] TDROBÓTICA. Batería LiPo 2200 mAh 11.1V [En línea] Disponible en: <http://tdrobotica.co/bateria-lipo-2200-mah-111v/465.html> [Citado el 2 de abril de 2018].

[52] LOGITECH. HD Pro Webcam C920 [En línea] Disponible en: <https://www.logitech.com/es-mx/product/hd-pro-webcam-c920> [Citado el 2 de abril de 2018].

[53] VISTRÓNICA. Indicador de tensión de Baterías LiPo [En línea] Disponible en: <https://www.vistronica.com/aeromodelismo/indicador-de-tension-de-baterias-lipo-detail.html> [Citado el 2 de abril de 2018].

[54] POLOLU. Pololu Ball Caster with 3/8″ Metal Ball [En línea] Disponible en: <https://www.pololu.com/product/951> [Citado el 2 de abril de 2018].

[55] SOLIDWORKS. Acrílico (Sistema Internacional). Biblioteca de materiales predefinidos. SOLIDWORKS Premium, 2017, Edición x64, SP 1.0.

[56] SOLIDWORKS. Factor de seguridad [En línea] Disponible en: <https://www.solidworks.com/sw/docs/Bridge_Poject_WB_2011_ESP.pdf> Pág. 53 [Citado el 10 de abril de 2018].

[57] SOLIDWORKS. Comprobación del Factor de seguridad [En línea] Disponible en: <http://help.solidworks.com/2013/spanish/SolidWorks/cworks/c_Factor_of_Safety_Check.html> [Citado el 10 de abril de 2018].

[58] ELSAYED, Mohamed. How does SolidWorks calculate the factor of safety? [En línea] Disponible en: <https://www.quora.com/How-does-SolidWorks-calculate-the-factor-of-safety> [Citado el 10 de abril de 2018].

[59] HOBBYKING. Lithium Ion battery safety guidelines : Discharging.

[60] EGERSTEDT, Magnus. Differential Drive Robots [videograbación] [En línea] 15 de diciembre de 2017. Disponible en: <https://www.youtube.com/watch?v=wqUwmnKskJU> [Citado el 26 de abril de 2018].

[61] EGERSTEDT, Magnus. A Clever Trick [videograbación] [En línea] 18 de diciembre de 2017. Disponible en: <https://www.youtube.com/watch?v=GX3A1G2FYZ0> [Citado el 26 de abril de 2018].

ANEXOS

ANEXO 1. COMPARATIVA DE LOS SISTEMA DE LOCOMOCIÓN

				Sistema de locomoción				
				1	2	3	4	5
Factor			Peso	Diferencial	Síncrona	Triciclo	Ackerman	Omnidireccional
1	Imagen	V	1					
		C		5	5	5	5	5
		P		5	5	5	5	5
2	Control	V	5	Sencillo	Complejo	Medio	Medio	Complejo
		C		5	1	3	3	1
		P		25	5	15	15	5
3	Estabilidad estática	V	2	Buena	Óptima	Buena	Óptima	Óptima
		C		4	5	4	5	5
		P		8	10	8	10	10
4	Estabilidad dinámica	V	5	Aceptable	Buena	Buena	Óptima	Óptima
		C		3	4	4	5	5
		P		15	20	20	25	25
5	Mantenimiento	V	3	Sencillo	Elevado	Medio	Medio	Elevado
		C		5	2	4	3	2
		P		15	6	12	9	6
6	Facilidad de implementación	V	5	Sencillo	Complejo	Medio	Medio	Complejo
		C		5	1	3	3	1
		P		25	5	15	15	5
7	Restricciones cinemáticas	V	5	Holonómica	Holonómica	No holonómica	No holonómica	Holonómica
		C		5	5	3	3	5
		P		25	25	15	15	25
8	Grados de libertad sistema	V	5	3	3	3	3	3
		C		5	5	5	5	5
		P		25	25	25	25	25
9	Número de actuadores	V	5	2	3 / 4	2	2 / 4	3 / 4
		C		5	3	5	4	3
		P		25	15	25	20	15
10	Costo	V	5	Económico	Costoso	Económico	Moderado	Costoso
		C		5	2	5	4	2
		P		25	10	25	20	10
Total Ponderado				193	126	165	159	131
Prioridad según T.P.				1	5	2	3	4

* V : Valor / C : Calificación / P : Puntaje

Fuente: Autor

ANEXO 2. COMPARATIVA DE LAS RUEDAS

ANEXO 3. COMPARATIVA DE LOS DISPOSITIVOS DE LOCOMOCIÓN

ANEXO 4. COMPARATIVA DE LOS ENCODERS

ANEXO 5. COMPARATIVA DE LOS SENSORES

ANEXO 6. COMPARATIVA DE LOS SISTEMAS EMBEBIDOS

ANEXO 7. COMPARATIVA DE LOS SISTEMAS DE COMUNICACIÓN

ANEXO 8. COMPARATIVA DE LOS PUENTE H

ANEXO 9. COMPARATIVA DE LAS BATERÍAS

ANEXO 10. CÁLCULOS EMPLEADOS EN LAS COMPARATIVAS

Para establecer la puntuación de cada factor según sea el elemento a analizar se emplea la siguiente ecuación:

$$P = C * \mathrm{W} \tag{57}$$

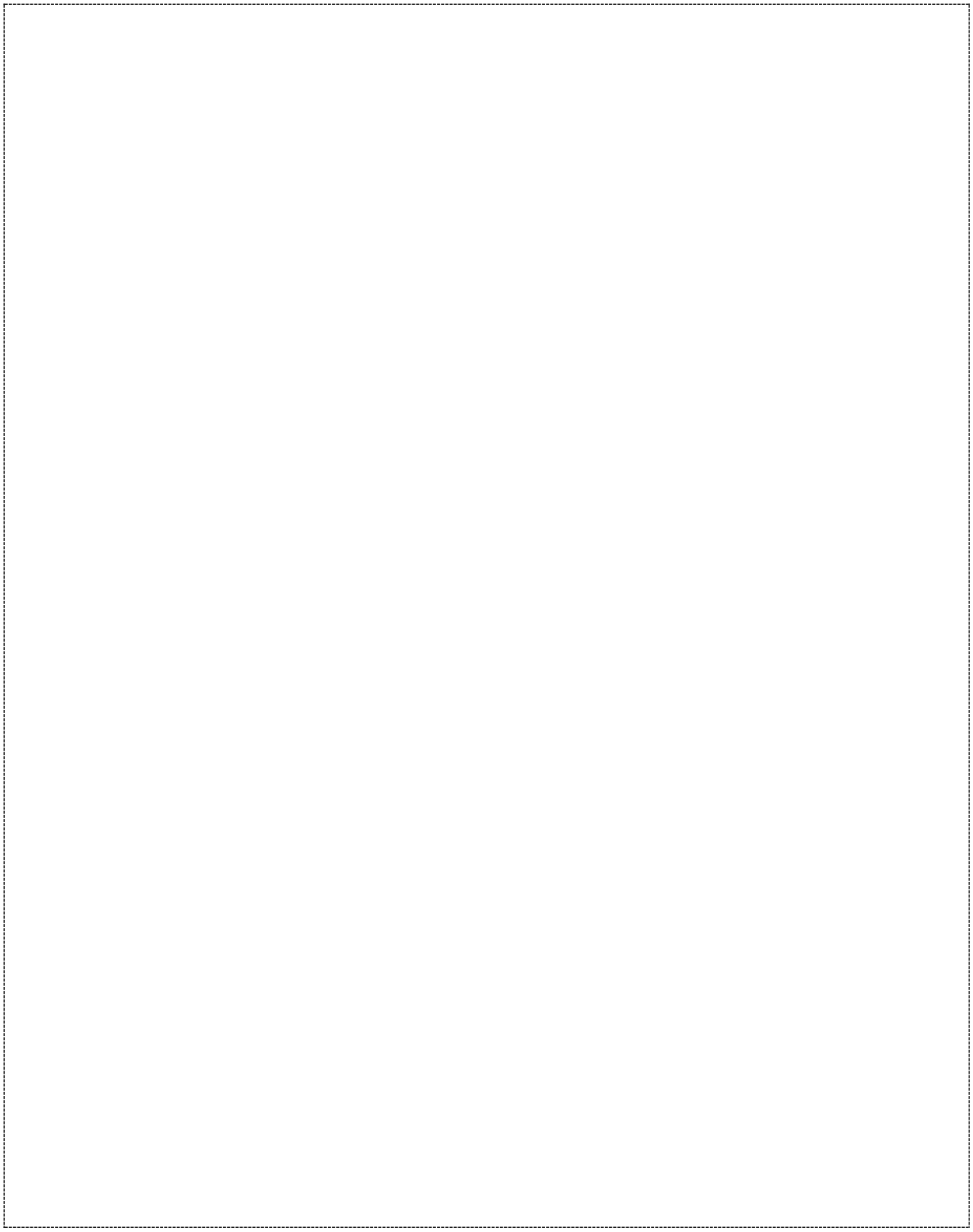
Donde P es la puntuación calculada, C es la calificación dada al factor de cada elemento comparado y W es el peso del factor analizado.

Para determinar que elemento comparado seleccionar, se emplea el total ponderado de las puntuaciones de cada elemento como criterio de selección:

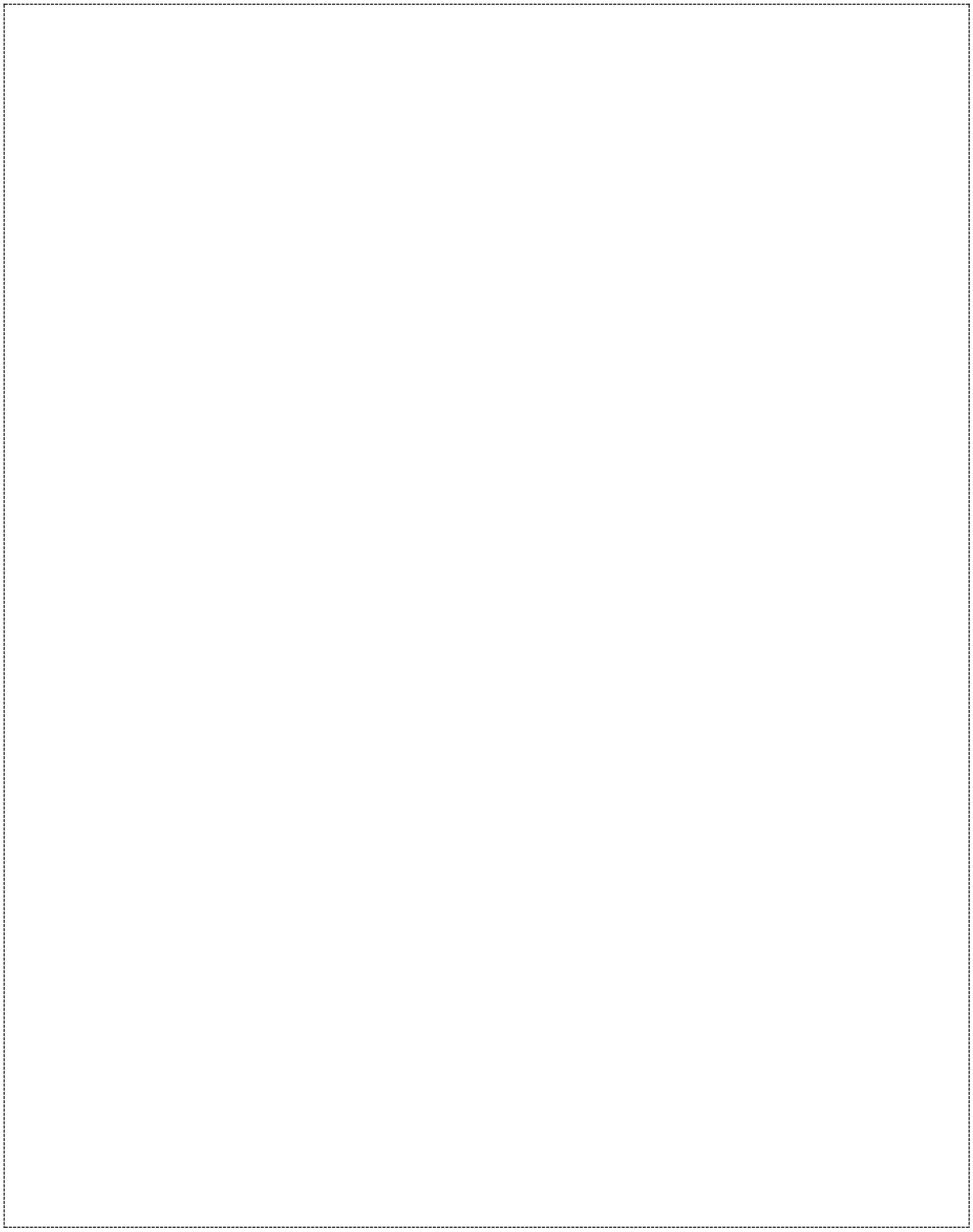
$$TP_m = \sum_{i=1}^{n} P_i \tag{58}$$

Donde TP_m es el total ponderado de cada elemento comparado y n la cantidad de factores analizados por elemento. El componente con mayor TP_m será el implementado en la plataforma robótica.

ANEXO 11. CIRCUITO ELÉCTRIO



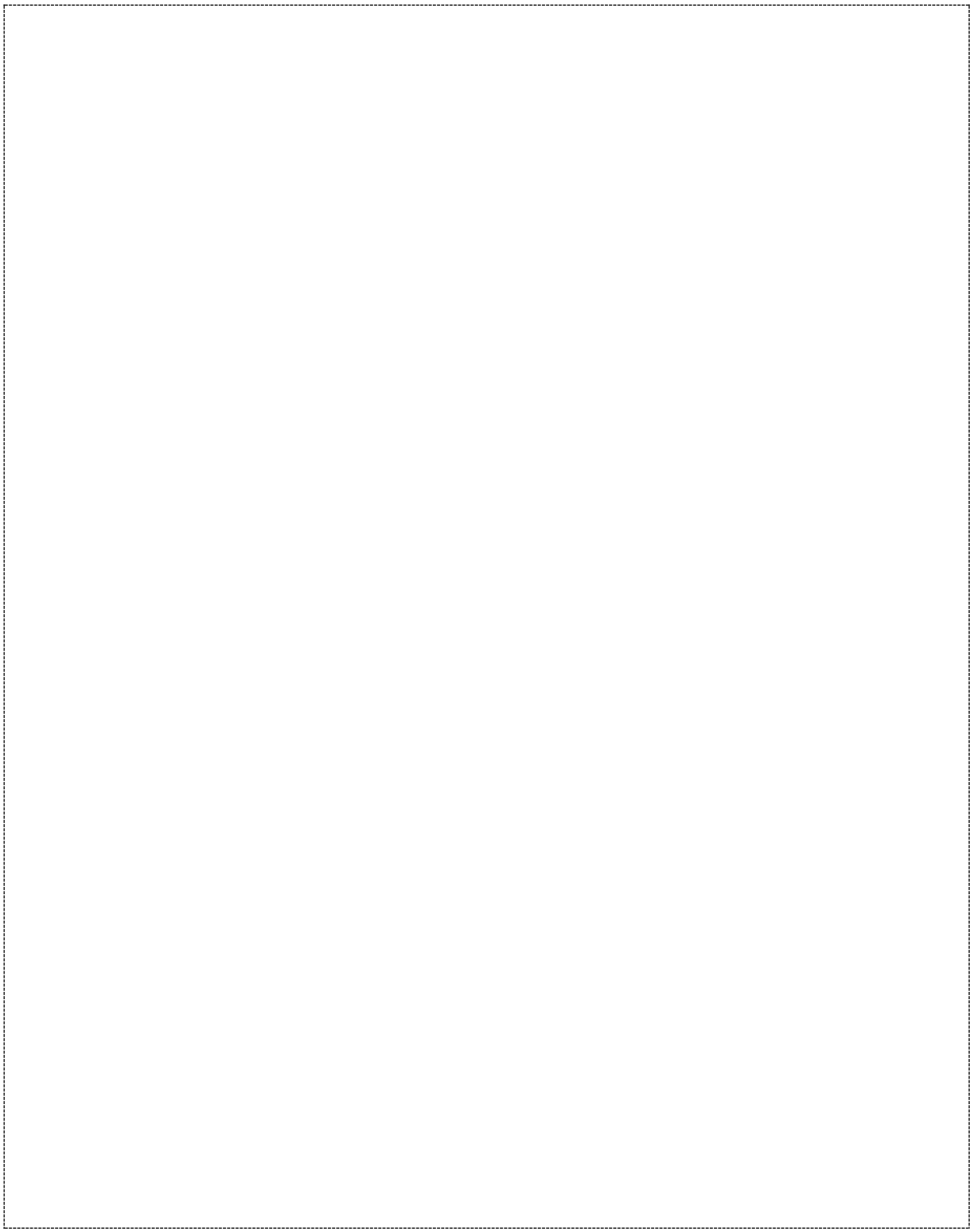
ANEXO 12. PLANO DE DESPIECE



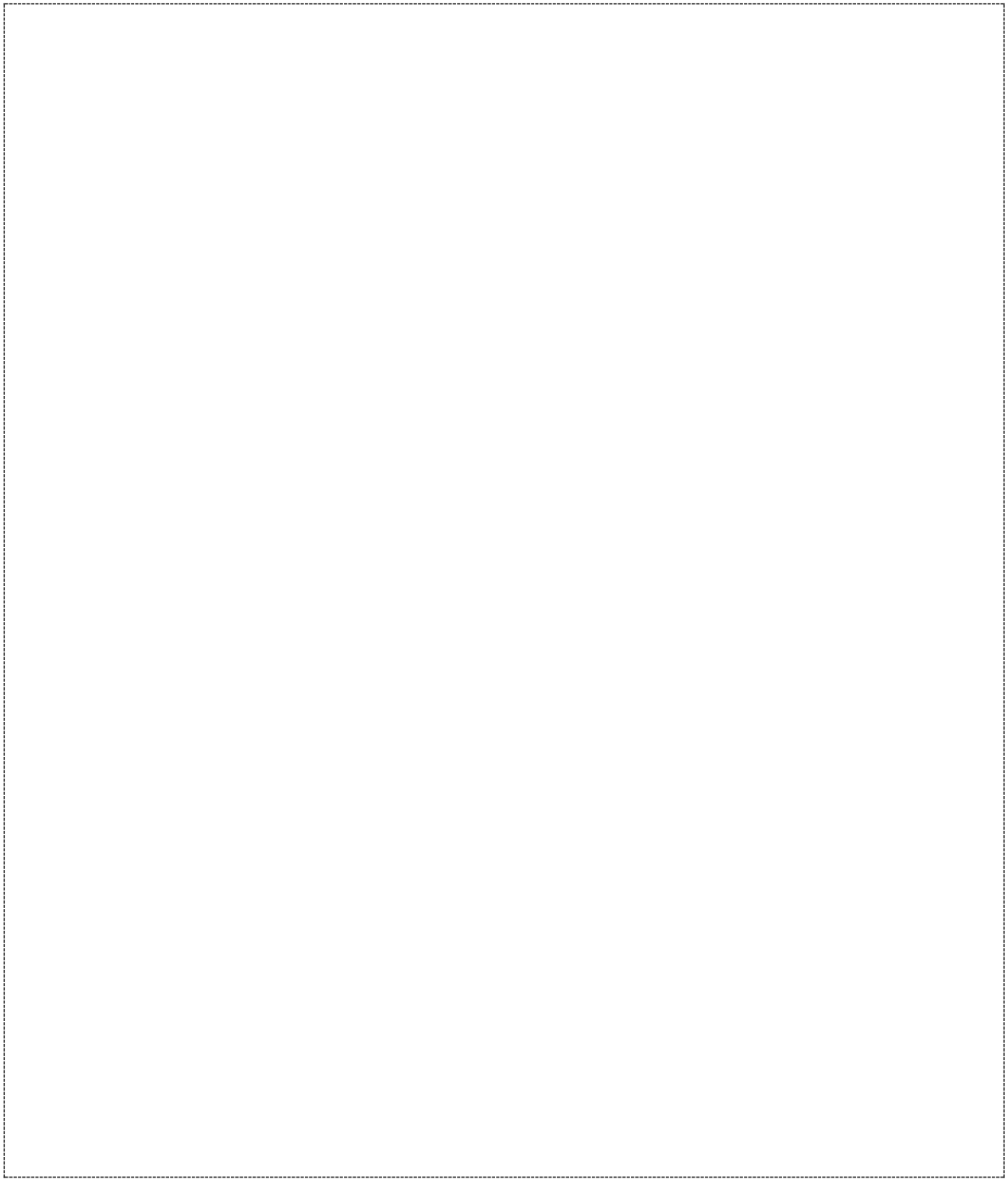
ANEXO 13. PLANO DE ENSAMBLE

ANEXO 14. PLANO DE VISTA EXPLOSIONADA

ANEXO 15. PLANO DEL SOPORTE DE LA CÁMARA WEB



ANEXO 16. PLANO DE LA ESTRUCTURA DE TRABAJO DE LA PLATAFORMA ROBÓTICA



ANEXO 17. CÓDIGO DEL ESCENARIO PA-RA

```
clc;
clear;

Iter = 15;

for run = 1:Iter

nR = 6; % Número de robots
F = Robots(nR); % Matriz coordenadas Puntos randómicos
R = Robots(nR); % Matriz coordenadas Robots randómicos
DE = DistEu(F,R); % Distancias euclidianas no ponderadas

% Dispersión de los puntos
for Lx = 1:nR-1
    DEDispP(Lx,1) = sqrt((F(Lx+1,1)-F(Lx,1))^2+(F(Lx+1,2)-F(Lx,2))^2);
end
DispP(run) = std(DEDispP);
DEDispP = 0;

% Dispersión de los robots
for Ly = 1:nR-1
    DEDispR(Ly,1) = sqrt((R(Ly+1,1)-R(Ly,1))^2+(R(Ly+1,2)-R(Ly,2))^2);
end
DispR(run) = std(DEDispR);
DEDispR = 0;

% DE1 : R+C a P+C
DE1 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = min(DE1(d+1,2:end));
    end
    Vminr = min(md);
    [Cr,Cft]=find(DE1==Vminr);
    Rp=DE1(Cr,1);
    Vminp = min(DE1(Cr,2:end));
    [Crt,Cf]=find(DE1==Vminp);
    Pp=DE1(1,Cf);
    DE1(Cr,:)=[];
    DE1(:,Cf)=[];
    GrapDE1(rep,:) = [R(Rp,:) F(Pp,:)];
    DT1(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE1 = sum(DT1);

% DE2 : R+A a P+C
```

```
DE2 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = max(DE2(d+1,2:end));
    end
    Vmaxr = max(md);
    [Cr,Cft]=find(DE2==Vmaxr);
    Rp=DE2(Cr,1);
    Vminp = min(DE2(Cr,2:end));
    [Crt,Cf]=find(DE2==Vminp);
    Pp=DE2(1,Cf);
    DE2(Cr,:)=[];
    DE2(:,Cf)=[];
    GrapDE2(rep,:) = [R(Rp,:) F(Pp,:)];
    DT2(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE2 = sum(DT2);

% DE3 : R+C a P+A
DE3 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = min(DE3(d+1,2:end));
    end
    Vminr = min(md);
    [Cr,Cft]=find(DE3==Vminr);
    Rp=DE3(Cr,1);
    Vmaxp = max(DE3(Cr,2:end));
    [Crt,Cf]=find(DE3==Vmaxp);
    Pp=DE3(1,Cf);
    DE3(Cr,:)=[];
    DE3(:,Cf)=[];
    GrapDE3(rep,:) = [R(Rp,:) F(Pp,:)];
    DT3(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE3 = sum(DT3);

%DE4 : R+C(DM) a P+C
DE4 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = min(DE4(d+1,2:end));
    end
    Media = mean(md);
    [S1,P]=min(abs(md-Media));
    S2 = md(P);
    [Cr,Cft]=find(DE4==S2);
```

```
    Rp=DE4(Cr,1);
    Vminp = min(DE4(Cr,2:end));
    [Crt,Cf]=find(DE4==Vminp);
    Pp=DE4(1,Cf);
    DE4(Cr,:)=[];
    DE4(:,Cf)=[];
    GrapDE4(rep,:) = [R(Rp,:) F(Pp,:)];
    DT4(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE4 = sum(DT4);

%DE5 : R+C(DM) a P+A
DE5 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = min(DE5(d+1,2:end));
    end
    Media = mean(md);
    [S1,P] = min(abs(md-Media));
    S2 = md(P);
    [Cr,Cft] = find(DE5==S2);
    Rp=DE5(Cr,1);
    Vmaxp = max(DE5(Cr,2:end));
    [Crt,Cf]=find(DE5==Vmaxp);
    Pp=DE5(1,Cf);
    DE5(Cr,:)=[];
    DE5(:,Cf)=[];
    GrapDE5(rep,:) = [R(Rp,:) F(Pp,:)];
    DT5(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE5 = sum(DT5);

%DE6 : RmDR a P+C
DE6 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        Suma(d,1) = sum(DE6(d+1,2:end));
    end
    [S1,P1] = min(Suma);
    [S2,P2] = min(DE6(P1+1,2:end));
    Rp=DE6(P1+1,1);
    Pp=DE6(1,P2+1);
    DE6(P1+1,:)=[];
    DE6(:,P2+1)=[];
    GrapDE6(rep,:) = [R(Rp,:) F(Pp,:)];
    DT6(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    Suma = 0;
```

```
end
DTDE6 = sum(DT6);

%DE7 : RmDR a P+A
DE7 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        Suma(d,1) = sum(DE7(d+1,2:end));
    end
    [S1,P1] = min(Suma);
    [S2,P2] = max(DE7(P1+1,2:end));
    Rp=DE7(P1+1,1);
    Pp=DE7(1,P2+1);
    DE7(P1+1,:)=[];
    DE7(:,P2+1)=[];
    GrapDE7(rep,:) = [R(Rp,:) F(Pp,:)];
    DT7(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    Suma = 0;
end
DTDE7 = sum(DT7);

Distancias (:,run) = [DTDE1; DTDE2; DTDE3; DTDE4; DTDE5; DTDE6; DTDE7];

end

figure(8)
bar(Distancias');
title('Distancias recorridas por ejecución');
xlabel('Iteraciones');
ylabel('Distancia [Unidades]');
legend({'R+C a P+C','R+A a P+C','R+C a P+A','R+C(DM) a P+C','R+C(DM) a
P+A','RmDR a P+C','RmDR a P+A'});
u = size(Distancias);
for k = 1:u(1,1)
    DisTotal(k,1) = sum(Distancias(k,:));
end
Orden = sort(DisTotal);
for E = 1:u
    [Pos,T] = find(DisTotal==Orden(E));
    DisTotal(Pos,2) = E;
end
DisTotal
DispRF = mean(DispR)
DispPF = mean(DispP)
```

ANEXO 18. CÓDIGO DEL ESCENARIO PF-RA

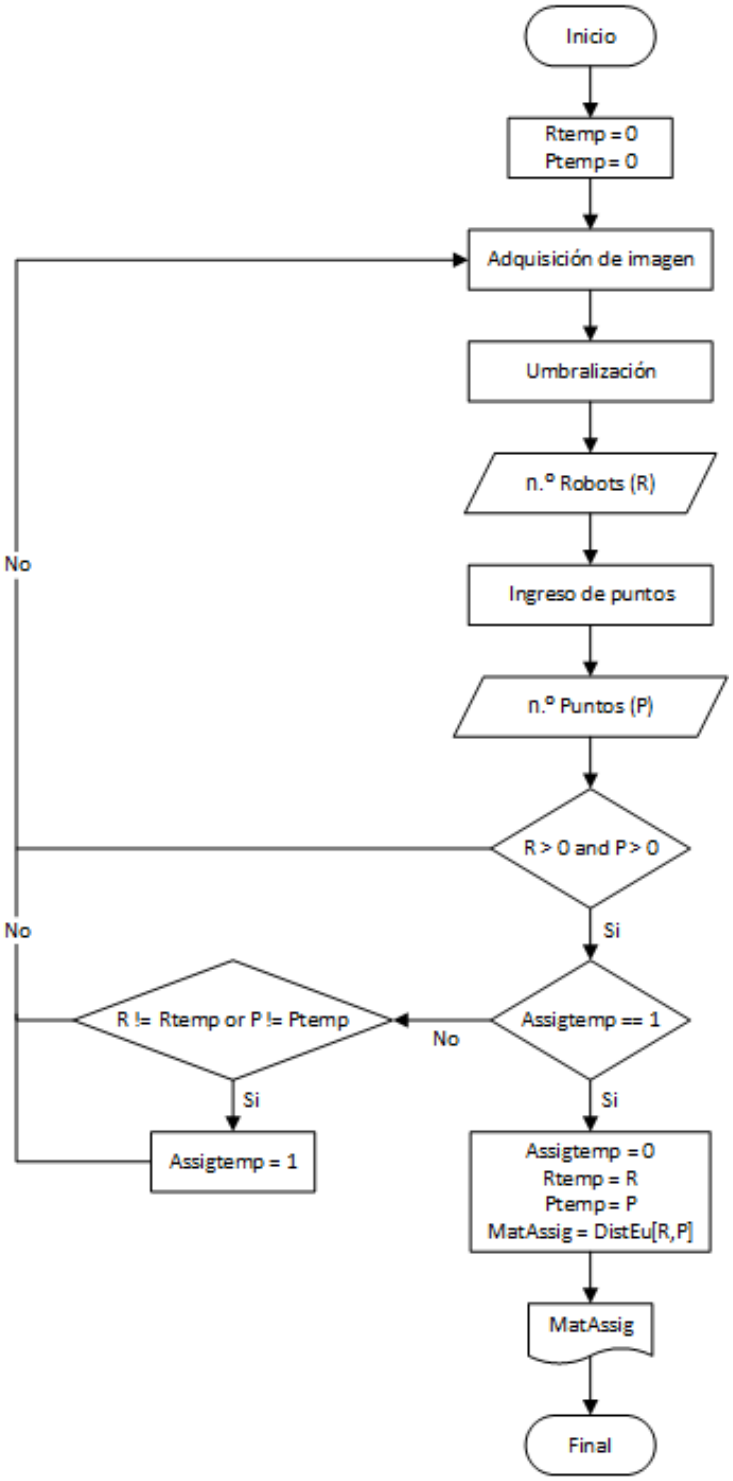
```
clc;
clear;

Iter = 15;

for run = 1:Iter

nR = 6; % Número de robots
F = Figura('Tr'); % Matriz coordenadas Figura: L, T, j, Hx, Tr y Rc
R = Robots(nR); % Matriz coordenadas Robots
DE = DistEu(F,R); % Distancias euclidianas no ponderadas

% Dispersión de los puntos
for Lx = 1:nR-1
    DEDispP(Lx,1) = sqrt((F(Lx+1,1)-F(Lx,1))^2+(F(Lx+1,2)-F(Lx,2))^2);
end
DispP(run) = mean(DEDispP);
DEDispP = 0;

% Dispersión de los robots
for Ly = 1:nR-1
    DEDispR(Ly,1) = sqrt((R(Ly+1,1)-R(Ly,1))^2+(R(Ly+1,2)-R(Ly,2))^2);
end
DispR(run) = mean(DEDispR);
DEDispR = 0;

% DE1 : R+C a P+C
DE1 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = min(DE1(d+1,2:end));
    end
    Vminr = min(md);
    [Cr,Cft]=find(DE1==Vminr);
    Rp=DE1(Cr,1);
    Vminp = min(DE1(Cr,2:end));
    [Crt,Cf]=find(DE1==Vminp);
    Pp=DE1(1,Cf);
    DE1(Cr,:)=[];
    DE1(:,Cf)=[];
    GrapDE1(rep,:) = [R(Rp,:) F(Pp,:)];
    DT1(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE1 = sum(DT1);
```

```
% DE2 : R+A a P+C
DE2 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = max(DE2(d+1,2:end));
    end
    Vmaxr = max(md);
    [Cr,Cft]=find(DE2==Vmaxr);
    Rp=DE2(Cr,1);
    Vminp = min(DE2(Cr,2:end));
    [Crt,Cf]=find(DE2==Vminp);
    Pp=DE2(1,Cf);
    DE2(Cr,:)=[];
    DE2(:,Cf)=[];
    GrapDE2(rep,:) = [R(Rp,:) F(Pp,:)];
    DT2(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE2 = sum(DT2);

% DE3 : R+C a P+A
DE3 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = min(DE3(d+1,2:end));
    end
    Vminr = min(md);
    [Cr,Cft]=find(DE3==Vminr);
    Rp=DE3(Cr,1);
    Vmaxp = max(DE3(Cr,2:end));
    [Crt,Cf]=find(DE3==Vmaxp);
    Pp=DE3(1,Cf);
    DE3(Cr,:)=[];
    DE3(:,Cf)=[];
    GrapDE3(rep,:) = [R(Rp,:) F(Pp,:)];
    DT3(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE3 = sum(DT3);

%DE4 : R+C(DM) a P+C   (Robot con dist. más cerca a la media de la dist.)
DE4 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = min(DE4(d+1,2:end));
    end
    Media = mean(md);
    [S1,P]=min(abs(md-Media));
```

```
    S2 = md(P);
    [Cr,Cft]=find(DE4==S2);
    Rp=DE4(Cr,1);
    Vminp = min(DE4(Cr,2:end));
    [Crt,Cf]=find(DE4==Vminp);
    Pp=DE4(1,Cf);
    DE4(Cr,:)=[];
    DE4(:,Cf)=[];
    GrapDE4(rep,:) = [R(Rp,:) F(Pp,:)];
    DT4(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE4 = sum(DT4);

%DE5 : R+C(DM) a P+A
DE5 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        md(d,1) = min(DE5(d+1,2:end));
    end
    Media = mean(md);
    [S1,P] = min(abs(md-Media));
    S2 = md(P);
    [Cr,Cft] = find(DE5==S2);
    Rp=DE5(Cr,1);
    Vmaxp = max(DE5(Cr,2:end));
    [Crt,Cf]=find(DE5==Vmaxp);
    Pp=DE5(1,Cf);
    DE5(Cr,:)=[];
    DE5(:,Cf)=[];
    GrapDE5(rep,:) = [R(Rp,:) F(Pp,:)];
    DT5(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    md = 0;
end
DTDE5 = sum(DT5);

%DE6 : RmDR a P+C   (robot con la menor distancia recorrida)
DE6 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        Suma(d,1) = sum(DE6(d+1,2:end));
    end
    [S1,P1] = min(Suma);
    [S2,P2] = min(DE6(P1+1,2:end));
    Rp=DE6(P1+1,1);
    Pp=DE6(1,P2+1);
    DE6(P1+1,:)=[];
    DE6(:,P2+1)=[];
    GrapDE6(rep,:) = [R(Rp,:) F(Pp,:)];
    DT6(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
```

```
    Cont = Cont - 1;
    Suma = 0;
end
DTDE6 = sum(DT6);

%DE7 : RmDR a P+A
DE7 = DE;
Cont = nR;
for rep = 1:nR
    for d = 1:Cont
        Suma(d,1) = sum(DE7(d+1,2:end));
    end
    [S1,P1] = min(Suma);
    [S2,P2] = max(DE7(P1+1,2:end));
    Rp=DE7(P1+1,1);
    Pp=DE7(1,P2+1);
    DE7(P1+1,:)=[];
    DE7(:,P2+1)=[];
    GrapDE7(rep,:) = [R(Rp,:) F(Pp,:)];
    DT7(rep,1) = sqrt((R(Rp,1)-F(Pp,1))^2 + (R(Rp,2)-F(Pp,2))^2);
    Cont = Cont - 1;
    Suma = 0;
end
DTDE7 = sum(DT7);

Distancias (:,run) = [DTDE1; DTDE2; DTDE3; DTDE4; DTDE5; DTDE6; DTDE7];

end

figure(8)
bar(Distancias');
title('Distancias recorridas por ejecución');
xlabel('Iteraciones');
ylabel('Distancia [Unidades]');
legend({'R+C a P+C','R+A a P+C','R+C a P+A','R+C(DM) a P+C','R+C(DM) a
P+A','RmDR a P+C','RmDR a P+A'});
u = size(Distancias);
for k = 1:u(1,1)
    DisTotal(k,1) = sum(Distancias(k,:));
end
Orden = sort(DisTotal);
for E = 1:u
    [Pos,T] = find(DisTotal==Orden(E));
    DisTotal(Pos,2) = E;
end
DisTotal
DispRF = mean(DispR)
DispPF = mean(DispP)
```

ANEXO 19. DIAGRAMA DE ASIGNACIÓN DE PUNTOS OBJETIVOS



Fuente: Autor

ANEXO 20. CÓDIGO DEL SISTEMA DE EVASIÓN DE MÚLTIPLES OBSTÁCULOS

```
function AngOA = SEMO(u)
global XObs YObs Xf Yf
Ds = 10; % Distancia de seguridad
n = length(XObs);
for i = 1:1:n
    Dobj(i,1) = sqrt((XObs(i)-u(1))^2 + (YObs(i)-u(2))^2);
end
AngF = atan2(Yf-u(2),Xf-u(1))*180/pi;
AngF = AngF + (AngF < 0)*360;
AngFT = AngF + 180;
if Dobj > Ds
    AngOA = 0;
else
    [~,Dm] = min(Dobj);
    AngObj = atan2(YObs(Dm)-u(2),XObs(Dm)-u(1))*180/pi;
    AngObj = AngObj + (AngObj < 0)*360;
    if AngF >= 0 && AngF <= 180
        if AngObj >= AngF && AngObj <= AngFT
            AngOA = -1.5708;
        else
            AngOA = 1.5708;
        end
    else
        if AngObj < AngF && AngObj > (AngFT-360)
            AngOA = 1.5708;
        else
            AngOA = -1.5708;
        end
    end
end
```

ANEXO 21. CÓDIGO DEL TRUCO INTELIGENTE

```
function [wd,wi] = fcn(u)
Xf = u(5); Yf = u(6);
Dp = 10; % Distancia al punto
L = 7.2; r = 1.6;
X = Xf - u(1); Y = Yf - u(2);
Angle = -u(4)+u(3);
A = [1 0; 0 1/Dp];
R = [cos(Angle) -sin(Angle); sin(Angle) cos(Angle)];
B = [X; Y];
s = A*R*B;
v = s(1)*4; w = s(2)*4;
wd = (2*v + L*w)/(2*r); wi = (2*v - L*w)/(2*r);
end
```

ANEXO 22. CÓDIGO DE LA INTERFAZ GRÁFICA EN LENGUAJE DE PROGRAMACIÓN PYHTON

```
# -*- coding: utf-8 -*-

from PyQt4 import QtCore, QtGui
import argparse
import imutils
import math
import numpy as np
import cv2 as cv
import serial
import time

# Se definene las variables globales
Ancho = 640
Alto = 480
MainStart = 0
CloseVid = 0
CloseVidT = 0
CloseVidCal = 0
CloseVidTCal = 0
CloseMainVid = 0
CentrosCal = 0
DrawObj = 0
# Variables para la asignación de puntos
Rtemp = 0
Ptemp = 0
# Radio de seguridad del robot
Ds = 70
```

```
# Variables de las checkboxs
VBorde = 1
VPosition = 1
VOr = 1
VCont = 1
VProx = 1
VLabel = 1
VAssig = 1
# Kernel rectangular
kernel = cv.getStructuringElement(cv.MORPH_RECT,(3,3)) #
https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html
# Variables para el mouse
pt = None
PF = 0
ContP = 0
Puntos = np.array([[0,0]])
# Variables guardar video
StarRecVideo = 0
# Variables para comunicación
Ready = 0
StartCOM = 0
Arduino = 0.0
TLim = 0.2
CT1 = 1

def Leer():
    File = open('ListSeg.txt','r')
    Cont = 0
    DictSeg = {}
    for lineR in File.read().splitlines():
```

```
        if Cont == 0:
            Color = lineR
            DictSeg[Color] = []
        else:
            DictSeg[Color].append(lineR)
        Cont += 1
        if Cont == 9:
            Cont = 0
    File.close()
    return DictSeg

def Escribir(ListaW):
    File = open('ListSeg.txt','w')
    for SegElem in ListaW:
        File.write(SegElem+'\n')
        for lineW in range(0,8):
            File.write(ListaW[SegElem][lineW]+'\n')
    File.close()

def LeerPx2cm():
    F = open('Px2cm.txt','r')
    Cont = 0
    D = {}
    for L in F.read().splitlines():
        if Cont == 0:
            itm = L
            D[itm] = []
        else:
            D[itm].append(L)
        Cont += 1
```

```
        if Cont == 7:
            Cont = 0
    F.close()
    return D

def EscribirPx2cm(ListPx2cm):
    F = open('Px2cm.txt','w')
    itm = 0
    for E in ListPx2cm:
        F.write(E+'\n')
        if itm == 0:
            for L in range(0,6):
                F.write(ListPx2cm[E][L]+'\n')
        else:
            for L in range(0,2):
                F.write(ListPx2cm[E][L]+'\n')
        itm += 1
    F.close()

def SegGPS(CAP,SD,LC,Px2cmFac):
    global kernel, VCont, VPosition, VOr, VProx
    area = []
    PtX = []
    PtY = []
    Xcm = []
    Ycm = []
    GPS = {}
    i = 0
    index = 0
    Cont = 0
```

```
    D = {}
    _, Image = CAP.read()
    hsv_temp = cv.cvtColor(Image,cv.COLOR_BGR2HSV)
    hsv = cv.GaussianBlur(hsv_temp, (5, 5), 0)
    for Robot in range(0,len(LC)):
        lower =
np.array([int(SD[LC[Robot]][0]),int(SD[LC[Robot]][1]),int(SD[LC[Robot]][2])])
        upper =
np.array([int(SD[LC[Robot]][3]),int(SD[LC[Robot]][4]),int(SD[LC[Robot]][5])])
        mask = cv.inRange(hsv, lower, upper)
        #Filtrar el ruido aplicando un OPEN seguido de un CLOSE
        mask = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
        maks = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
        result = cv.bitwise_and(Image,Image,mask = mask)
        gray = cv.cvtColor(result, cv.COLOR_BGR2GRAY)
        blurred = cv.GaussianBlur(gray, (5, 5), 0)
        thresh = cv.threshold(blurred, 60, 255, cv.THRESH_BINARY)[1]
        cnts = cv.findContours(thresh.copy(), cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if imutils.is_cv2() else cnts[1]
        for a in cnts:
            index = index + 1
        if index == 2:
            for c in cnts:
                M = cv.moments(c)
                area.append(cv.contourArea(c))
                if M["m00"] != 0:
                    PtX.append(int(M["m10"] / M["m00"]))
                    PtY.append(int(M["m01"] / M["m00"]))
                    if VCont == 1:
```

```
                cv.drawContours(Image, [c], -1, (0, 255, 0), 2)
            Hpx = Alto - PtY[i]
            Xcm.append(round(PtX[i]*Px2cmFac,2))
            Ycm.append(round(Hpx*Px2cmFac,2))
            if VPosition == 1:
                cv.circle(Image, (PtX[i], PtY[i]), 2, (0, 0, 255), -1)
                cv.putText(Image, ('('+str(Xcm[i])+','+str(Ycm[i])+')'), (PtX[i]-20,
PtY[i]-20), cv.FONT_HERSHEY_SIMPLEX, 0.43, (0, 0, 255), 1)
            i = i+1
        if i == 2:
            if area[0]>area[1]:
                OrienGrad = math.atan2((Ycm[1]-Ycm[0]),(Xcm[1]-
Xcm[0]))*180/math.pi
                OrienRad = math.atan2((Ycm[1]-Ycm[0]),(Xcm[1]-Xcm[0]))
                if VProx == 1:
                 cv.circle(Image, (PtX[0], PtY[0]), Ds, (0, 0, 255), 1)
                if VOr == 1:
                    cv.arrowedLine(Image, (PtX[0],PtY[0]), (PtX[1],PtY[1]), (0,0,255), 1,
8, 0, 0.1)
                    cv.putText(Image, (str(round(OrienGrad,2))+' grados'), (PtX[0]-30,
PtY[0]+20), cv.FONT_HERSHEY_SIMPLEX, 0.43, (0, 0, 255), 1)
                D[Robot] = [PtX[0],PtY[0],Xcm[0],Ycm[0],OrienGrad, OrienRad]
            if area[1]>area[0]:

                OrienGrad = math.atan2((Ycm[0]-Ycm[1]),(Xcm[0]-
Xcm[1]))*180/math.pi
                OrienRad = math.atan2((Ycm[0]-Ycm[1]),(Xcm[0]-Xcm[1]))
                if VProx == 1:
                    cv.circle(Image, (PtX[1], PtY[1]), Ds, (0, 0, 255), 1)
                if VOr == 1:
```

```
                    cv.arrowedLine(Image, (PtX[1],PtY[1]), (PtX[0],PtY[0]), (0,0,255), 1,
8, 0, 0.1)
                    cv.putText(Image, (str(round(OrienGrad,2))+' grados'), (PtX[1]-30,
PtY[1]+20), cv.FONT_HERSHEY_SIMPLEX, 0.43, (0, 0, 255), 1)
                D[Robot] = [PtX[1],PtY[1],Xcm[1],Ycm[1],OrienGrad, OrienRad]
            i = 0
            Cont += 1
            area = []
            PtX = []
            PtY = []
            Xcm = []
            Ycm = []
        index = 0
    Cont = 0
    return Image, D

# Asignamos los puntos con el mouse
def on_mouse(event, x, y, flags, param):
    global pt, Alto, Ancho, PF, ContP, Puntos
    if event == cv.EVENT_LBUTTONDOWN:
        if x >= 50 and x <= Ancho-50 and y >= 50 and y <= Alto-50:
            pt = (x, y)
            if PF == 0:
                Puntos[0] = pt
                PF = 1
                ContP += 1
            else:
                Puntos = np.append(Puntos, [[pt[0],pt[1]]], 0)
                ContP += 1
```

```
# Obtenemos las coordenatas de los robots en pixeles
def getPxRo(Directory):
    Dots = np.zeros([len(Directory),3])
    C = 0
    for Px in Directory.keys():
        Dots[C][:2] = Directory[Px][:2]
        Dots[C][2] = Px
        C += 1
    return Dots

# Obtenemos las coordenatas de los robots en cm
def getXYRo(Directory):
    Dots = np.zeros([len(Directory),3])
    C = 0
    for Px in Directory.keys():
        Dots[C][:2] = Directory[Px][2:4]
        Dots[C][2] = Px
        C += 1
    return Dots

def getGrad(Directory):
    Or = np.zeros([len(Directory),2])
    C = 0
    for O in Directory.keys():
        Or[C][0] = Directory[O][4]
        Or[C][1] = O
        C += 1
    return Or

def getRad(Directory):
```

```
    Or = np.zeros([len(Directory),2])
    C = 0
    for O in Directory.keys():
        Or[C][0] = Directory[O][5]
        Or[C][1] = O
        C += 1
    return Or

def getXYDots(Dots, Px2cmFac):
    global Alto
    OutM = np.zeros([len(Dots),2])
    for P in range(0,len(Dots)):
        Hpx = Alto - Dots[P][1]
        OutM[P] = np.array([round(Dots[P][0]*Px2cmFac,2),round(Hpx*Px2cmFac,2)])
    return OutM

# Distancias euclidianas no ponderadas
def DE(R,P):
    numr = len(R)
    nump = len(P)
    # Indr = np.array((range(0,numr+1)))
    Indr = R[0:,2:].transpose()
    Cont = np.zeros([nump+1,numr])
    Cont = np.c_[0:nump+1,Cont]
    Cont[0][1:] = Indr
    for m in range(1,numr+1):
        for j in range(1,nump+1):
            Cont[[j],[m]] = round(math.sqrt((P[[j-1],[0]]-R[[m-1],[0]])**2+(P[[j-1],[1]]-R[[m-
1],[1]])**2),2)
    return Cont
```

```
# Asignacion de robot+cercano a punto+cercano
def AsRcPc(Mat,R,P):
    if len(R) > len(P):
        lim = len(P)
    else:
        lim = len(R)
    IndOut = np.zeros([lim,2])
    for RS in range(0,lim):
        minn = (min(map(min,Mat[1:,1:])))
        Ind = np.argwhere(Mat[1:,1:]==minn)
        IndOut[RS] = [Mat[[int(Ind[0][0])+1],[0]],Mat[[0],[int(Ind[0][1])+1]]]
        Mat = np.delete(Mat,int(Ind[0][0])+1,0) # Borra fila
        Mat = np.delete(Mat,int(Ind[0][1])+1,1) # Borrar columna
    return IndOut # [Puntos, Robots]

# Asignacion de robot+alejado a punto+cercano
def AsRaPc(Mat,R,P):
    if len(R) > len(P):
        lim = len(P)
    else:
        lim = len(R)
    IndOut = np.zeros([lim,2])
    for RS in range(0,lim):
        maxn = (max(map(max,Mat[1:,1:])))
        Ind = np.argwhere(Mat[1:,1:]==maxn)
        CRmax = Mat[1:,(int(Ind[0][1])+1)]
        minn = min(CRmax)
        CRmin = np.argwhere(CRmax==minn)
        IndOut[RS] = [Mat[[int(CRmin)+1],[0]],Mat[[0],[int(Ind[0][1])+1]]]
```

```
        Mat = np.delete(Mat,int(CRmin)+1,0) # Borra fila
        Mat = np.delete(Mat,int(Ind[0][1])+1,1) # Borrar columna
    return IndOut # [Puntos, Robots]

def getColor(LC,D):
    if LC[int(D[2])] == 'Rojo':
        ColorFlag = 'R'
    elif LC[int(D[2])] == 'Azul':
        ColorFlag = 'B'
    elif LC[int(D[2])] == 'Verde':
        ColorFlag = 'G'
    elif LC[int(D[2])] == 'Amarillo':
        ColorFlag = 'Y'
    elif LC[int(D[2])] == 'Naranja':
        ColorFlag = 'O'
    elif LC[int(D[2])] == 'Fucsia':
        ColorFlag = 'F'
    comm = '<'
    comm += ColorFlag
    comm += str(round(D[0],2)).zfill(7)
    comm += ','
    comm += str(round(D[1],2)).zfill(7)
    comm += '>'
    return comm

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s
```

```
try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

# Ventana de calibración
class Ui_CalibrationWindow(object):
    def setupUi(self, CalibrationWindow):
        CalibrationWindow.setObjectName(_fromUtf8("CalibrationWindow"))
        CalibrationWindow.setFixedSize(261, 461)
        self.ParametersBoxCal = QtGui.QGroupBox(CalibrationWindow)
        self.ParametersBoxCal.setGeometry(QtCore.QRect(10, 147, 241, 181))
        self.ParametersBoxCal.setObjectName(_fromUtf8("ParametersBoxCal"))
        self.HLLabelCal = QtGui.QLabel(self.ParametersBoxCal)
        self.HLLabelCal.setGeometry(QtCore.QRect(17, 22, 17, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.HLLabelCal.setFont(font)
        self.HLLabelCal.setTextFormat(QtCore.Qt.AutoText)

self.HLLabelCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCor
e.Qt.AlignVCenter)
        self.HLLabelCal.setObjectName(_fromUtf8("HLLabelCal"))
        self.VLNumCal = QtGui.QLabel(self.ParametersBoxCal)
        self.VLNumCal.setGeometry(QtCore.QRect(203, 76, 22, 16))
```

```
self.VLNumCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.
Qt.AlignVCenter)
        self.VLNumCal.setObjectName(_fromUtf8("VLNumCal"))
        self.SLLabelCal = QtGui.QLabel(self.ParametersBoxCal)
        self.SLLabelCal.setGeometry(QtCore.QRect(17, 49, 16, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.SLLabelCal.setFont(font)

self.SLLabelCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCor
e.Qt.AlignVCenter)
        self.SLLabelCal.setObjectName(_fromUtf8("SLLabelCal"))
        self.VLLabelCal = QtGui.QLabel(self.ParametersBoxCal)
        self.VLLabelCal.setGeometry(QtCore.QRect(17, 76, 16, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.VLLabelCal.setFont(font)

self.VLLabelCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCor
e.Qt.AlignVCenter)
        self.VLLabelCal.setObjectName(_fromUtf8("VLLabelCal"))
        self.VHLabelCal = QtGui.QLabel(self.ParametersBoxCal)
        self.VHLabelCal.setGeometry(QtCore.QRect(17, 157, 18, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.VHLabelCal.setFont(font)
```

```
self.VHLabelCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCor
e.Qt.AlignVCenter)
        self.VHLabelCal.setObjectName(_fromUtf8("VHLabelCal"))
        self.VHNumCal = QtGui.QLabel(self.ParametersBoxCal)
        self.VHNumCal.setGeometry(QtCore.QRect(203, 157, 22, 16))

self.VHNumCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore
.Qt.AlignVCenter)
        self.VHNumCal.setObjectName(_fromUtf8("VHNumCal"))
        self.HHLabelCal = QtGui.QLabel(self.ParametersBoxCal)
        self.HHLabelCal.setGeometry(QtCore.QRect(17, 103, 19, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.HHLabelCal.setFont(font)

self.HHLabelCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCor
e.Qt.AlignVCenter)
        self.HHLabelCal.setObjectName(_fromUtf8("HHLabelCal"))
        self.SHNumCal = QtGui.QLabel(self.ParametersBoxCal)
        self.SHNumCal.setGeometry(QtCore.QRect(203, 130, 22, 16))

self.SHNumCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore
.Qt.AlignVCenter)
        self.SHNumCal.setObjectName(_fromUtf8("SHNumCal"))
        self.SHLabelCal = QtGui.QLabel(self.ParametersBoxCal)
        self.SHLabelCal.setGeometry(QtCore.QRect(17, 130, 18, 16))
        font = QtGui.QFont()
        font.setBold(True)
```

```
        font.setWeight(75)
        self.SHLabelCal.setFont(font)

self.SHLabelCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCor
e.Qt.AlignVCenter)
        self.SHLabelCal.setObjectName(_fromUtf8("SHLabelCal"))
        self.HLNumCal = QtGui.QLabel(self.ParametersBoxCal)
        self.HLNumCal.setGeometry(QtCore.QRect(203, 22, 22, 16))
        self.HLNumCal.setLayoutDirection(QtCore.Qt.LeftToRight)

self.HLNumCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore
.Qt.AlignVCenter)
        self.HLNumCal.setObjectName(_fromUtf8("HLNumCal"))
        self.HHNumCal = QtGui.QLabel(self.ParametersBoxCal)
        self.HHNumCal.setGeometry(QtCore.QRect(203, 103, 22, 16))

self.HHNumCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore
.Qt.AlignVCenter)
        self.HHNumCal.setObjectName(_fromUtf8("HHNumCal"))
        self.SLNumCal = QtGui.QLabel(self.ParametersBoxCal)
        self.SLNumCal.setGeometry(QtCore.QRect(203, 49, 22, 16))

self.SLNumCal.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.
Qt.AlignVCenter)
        self.SLNumCal.setObjectName(_fromUtf8("SLNumCal"))
        self.HLBarraCal = QtGui.QScrollBar(self.ParametersBoxCal)
        self.HLBarraCal.setGeometry(QtCore.QRect(46, 22, 150, 19))
        self.HLBarraCal.setMaximum(179)
        self.HLBarraCal.setOrientation(QtCore.Qt.Horizontal)
        self.HLBarraCal.setObjectName(_fromUtf8("HLBarraCal"))
```

```
self.SLBarraCal = QtGui.QScrollBar(self.ParametersBoxCal)
self.SLBarraCal.setGeometry(QtCore.QRect(46, 49, 150, 19))
self.SLBarraCal.setMaximum(255)
self.SLBarraCal.setOrientation(QtCore.Qt.Horizontal)
self.SLBarraCal.setObjectName(_fromUtf8("SLBarraCal"))
self.VLBarraCal = QtGui.QScrollBar(self.ParametersBoxCal)
self.VLBarraCal.setGeometry(QtCore.QRect(46, 76, 150, 19))
self.VLBarraCal.setMaximum(255)
self.VLBarraCal.setOrientation(QtCore.Qt.Horizontal)
self.VLBarraCal.setObjectName(_fromUtf8("VLBarraCal"))
self.HHBarraCal = QtGui.QScrollBar(self.ParametersBoxCal)
self.HHBarraCal.setGeometry(QtCore.QRect(46, 103, 150, 19))
self.HHBarraCal.setMaximum(179)
self.HHBarraCal.setSliderPosition(179)
self.HHBarraCal.setOrientation(QtCore.Qt.Horizontal)
self.HHBarraCal.setObjectName(_fromUtf8("HHBarraCal"))
self.SHBarraCal = QtGui.QScrollBar(self.ParametersBoxCal)
self.SHBarraCal.setGeometry(QtCore.QRect(46, 130, 150, 19))
self.SHBarraCal.setMaximum(255)
self.SHBarraCal.setSliderPosition(255)
self.SHBarraCal.setOrientation(QtCore.Qt.Horizontal)
self.SHBarraCal.setObjectName(_fromUtf8("SHBarraCal"))
self.VHBarraCal = QtGui.QScrollBar(self.ParametersBoxCal)
self.VHBarraCal.setGeometry(QtCore.QRect(46, 157, 150, 19))
self.VHBarraCal.setMaximum(255)
self.VHBarraCal.setSliderPosition(255)
self.VHBarraCal.setOrientation(QtCore.Qt.Horizontal)
self.VHBarraCal.setObjectName(_fromUtf8("VHBarraCal"))
self.OptionsBoxCal = QtGui.QGroupBox(CalibrationWindow)
self.OptionsBoxCal.setGeometry(QtCore.QRect(10, 337, 241, 115))
```

```
self.OptionsBoxCal.setObjectName(_fromUtf8("OptionsBoxCal"))
self.SaveButtonCal = QtGui.QPushButton(self.OptionsBoxCal)
self.SaveButtonCal.setGeometry(QtCore.QRect(22, 64, 90, 35))
self.SaveButtonCal.setObjectName(_fromUtf8("SaveButtonCal"))
self.CancelButtonCal = QtGui.QPushButton(self.OptionsBoxCal)
self.CancelButtonCal.setGeometry(QtCore.QRect(127, 64, 90, 35))
self.CancelButtonCal.setObjectName(_fromUtf8("CancelButtonCal"))
self.VideoButtonCal = QtGui.QPushButton(self.OptionsBoxCal)
self.VideoButtonCal.setGeometry(QtCore.QRect(22, 20, 90, 35))
self.VideoButtonCal.setObjectName(_fromUtf8("VideoButtonCal"))
self.CalibrationButton = QtGui.QPushButton(self.OptionsBoxCal)
self.CalibrationButton.setGeometry(QtCore.QRect(127, 20, 90, 35))
self.CalibrationButton.setObjectName(_fromUtf8("CalibrationButton"))
self.DistanceDotsBox = QtGui.QGroupBox(CalibrationWindow)
self.DistanceDotsBox.setGeometry(QtCore.QRect(10, 7, 241, 61))
self.DistanceDotsBox.setObjectName(_fromUtf8("DistanceDotsBox"))
self.DistanceDots = QtGui.QTextEdit(self.DistanceDotsBox)
self.DistanceDots.setGeometry(QtCore.QRect(30, 20, 180, 30))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.DistanceDots.setFont(font)
self.DistanceDots.setObjectName(_fromUtf8("DistanceDots"))
self.Px2cmBox = QtGui.QGroupBox(CalibrationWindow)
self.Px2cmBox.setGeometry(QtCore.QRect(10, 77, 241, 61))
self.Px2cmBox.setObjectName(_fromUtf8("Px2cmBox"))
self.Px2cmBrowser = QtGui.QTextBrowser(self.Px2cmBox)
self.Px2cmBrowser.setGeometry(QtCore.QRect(30, 20, 180, 30))
font = QtGui.QFont()
```

```
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.Px2cmBrowser.setFont(font)
        self.Px2cmBrowser.setObjectName(_fromUtf8("Px2cmBrowser"))

        Px2cmDict = LeerPx2cm()
        ListKeys = []
        for ListK in Px2cmDict:
            ListKeys.append(ListK)
        # Iniciamos la distancia entre los puntos
        self.DistanceDots.setPlainText(Px2cmDict[ListKeys[1]][0])
        self.DistanceDots.setFocus()
        self.DistanceDots.setAlignment(QtCore.Qt.AlignCenter)
        # Iniciamos el factor de conversión Px2cm
        self.Px2cmBrowser.setPlainText(Px2cmDict[ListKeys[1]][1])
        self.Px2cmBrowser.setAlignment(QtCore.Qt.AlignCenter)
        # Iniciamos las barras y textos
        self.HLBarraCal.setSliderPosition(int(Px2cmDict[ListKeys[0]][0]))
        self.HLNumCal.setText(_translate("CalibrationWindow",
Px2cmDict[ListKeys[0]][0], None))
        self.SLBarraCal.setSliderPosition(int(Px2cmDict[ListKeys[0]][1]))
        self.SLNumCal.setText(_translate("CalibrationWindow",
Px2cmDict[ListKeys[0]][1], None))
        self.VLBarraCal.setSliderPosition(int(Px2cmDict[ListKeys[0]][2]))
        self.VLNumCal.setText(_translate("CalibrationWindow",
Px2cmDict[ListKeys[0]][2], None))
        self.HHBarraCal.setSliderPosition(int(Px2cmDict[ListKeys[0]][3]))
        self.HHNumCal.setText(_translate("CalibrationWindow",
Px2cmDict[ListKeys[0]][3], None))
```

```
        self.SHBarraCal.setSliderPosition(int(Px2cmDict[ListKeys[0]][4]))
        self.SHNumCal.setText(_translate("CalibrationWindow",
Px2cmDict[ListKeys[0]][4], None))
        self.VHBarraCal.setSliderPosition(int(Px2cmDict[ListKeys[0]][5]))
        self.VHNumCal.setText(_translate("CalibrationWindow",
Px2cmDict[ListKeys[0]][5], None))

        self.retranslateUi(CalibrationWindow)
        QtCore.QObject.connect(self.SaveButtonCal,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.SalvarCal)
        QtCore.QObject.connect(self.SaveButtonCal,
QtCore.SIGNAL(_fromUtf8("clicked()")), CalibrationWindow.close)
        QtCore.QObject.connect(self.VideoButtonCal,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.StarVideoCal)
        QtCore.QObject.connect(self.CancelButtonCal,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.CerrarCal)
        QtCore.QObject.connect(self.CancelButtonCal,
QtCore.SIGNAL(_fromUtf8("clicked()")), CalibrationWindow.close)
        QtCore.QObject.connect(self.HLBarraCal,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.HLNumCal.setNum)
        QtCore.QObject.connect(self.SLBarraCal,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.SLNumCal.setNum)
        QtCore.QObject.connect(self.VLBarraCal,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.VLNumCal.setNum)
        QtCore.QObject.connect(self.HHBarraCal,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.HHNumCal.setNum)
        QtCore.QObject.connect(self.SHBarraCal,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.SHNumCal.setNum)
        QtCore.QObject.connect(self.VHBarraCal,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.VHNumCal.setNum)
```

```
        QtCore.QObject.connect(self.CalibrationButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.Calibration)
        QtCore.QMetaObject.connectSlotsByName(CalibrationWindow)

    def StarVideoCal(self):
        global Alto, Ancho, CloseVidCal, CloseVidTCal, CentrosCal, kernel

        # Variables locales
        area = []
        PtX = []
        PtY = []
        i = 0
        index = 0

        CloseVidTCal = 1
        # Iniciamos el video
        cap = cv.VideoCapture(0)
        # Establecemos la resolución de la cámara web
        cap.set(cv.CAP_PROP_FRAME_WIDTH, Ancho)
        cap.set(cv.CAP_PROP_FRAME_HEIGHT, Alto)
        while CloseVidCal == 0:
            _, frame = cap.read()
            # Convertimos al espacio de color HSV
            hsv_temp = cv.cvtColor(frame,cv.COLOR_BGR2HSV)
            # Pasamos por un filtro gaussiano
            hsv = cv.GaussianBlur(hsv_temp, (5, 5), 0)
            # Máscara
            lower =
np.array([int(self.HLNumCal.text()),int(self.SLNumCal.text()),int(self.VLNumCal.text
())])
```

```
            upper =
np.array([int(self.HHNumCal.text()),int(self.SHNumCal.text()),int(self.VHNumCal.te
xt())])
            mask = cv.inRange(hsv, lower, upper)
            #Filtrar el ruido aplicando un OPEN seguido de un CLOSE
            mask = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
            maks = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
            result = cv.bitwise_and(frame,frame,mask = mask)
            cv.imshow('Calibration Window',result)

            if CentrosCal == 1:
                image = result
                gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
                blurred = cv.GaussianBlur(gray, (5, 5), 0)
                thresh = cv.threshold(blurred, 60, 255, cv.THRESH_BINARY)[1]

                # Enconctramos los contornos en la imagen umbralizada
                cnts = cv.findContours(thresh.copy(), cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
                cnts = cnts[0] if imutils.is_cv2() else cnts[1]

                for a in cnts:
                    index = index + 1

                if index == 2:

                    # Ciclo con contornos
                    for c in cnts:
                        # Calculamos el centro de los contornos
                        M = cv.moments(c)
```

```
                area.append(cv.contourArea(c))
                PtX.append(int(M["m10"] / M["m00"]))
                PtY.append(int(M["m01"] / M["m00"]))

                # Dibujamos el contorno y el centro de la forma de la imagen
                cv.drawContours(image, [c], -1, (0, 255, 0), 2)
                cv.circle(image, (PtX[i], PtY[i]), 2, (0, 0, 255), -1)
                Hpx = Alto - PtY[i]
                cv.putText(image, ('('+str(PtX[i])+','+str(PtY[i])+')'), (PtX[i]-20, PtY[i]-
20), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
                i = i+1

            Df = float(str(self.DistanceDots.toPlainText())) # Distancia entre los
objetos de calibración en [cm]

            if area[0]>area[1]:
                Dpx = math.sqrt((PtY[1]-PtY[0])**2 + (PtX[1]-PtX[0])**2)
                Px2cm = Df/Dpx
            if area[1]>area[0]:
                Dpx = math.sqrt((PtY[0]-PtY[1])**2 + (PtX[0]-PtX[1])**2)
                Px2cm = Df/Dpx

            self.Px2cmBrowser.setPlainText(str(Px2cm))
            self.Px2cmBrowser.setAlignment(QtCore.Qt.AlignCenter)
            self.Px2cmBrowser.setStyleSheet("color: red")

            # Determinamos el área mayor
            if area[0]>area[1]:
                cv.arrowedLine(image, (PtX[0],PtY[0]), (PtX[1],PtY[1]), (0,0,255), 1,
8, 0, 0.1)
```

```
                if area[1]>area[0]:
                    cv.arrowedLine(image, (PtX[1],PtY[1]), (PtX[0],PtY[0]), (0,0,255), 1,
8, 0, 0.1)

                # Se reinician las variables pertinentes
                i = 0
                area = []
                PtX = []
                PtY = []

            index = 0
            # Mostramos la imagen
            cv.imshow('Calibration Window', image)
        if cv.waitKey(1) & 0xFF == 27:
            break
    cap.release()
    cv.destroyWindow('Calibration Window')
    CloseVidCal = 0
    CloseVidTCal = 0
    CentrosCal = 0

def CerrarCal(self):
    global CloseVidCal
    CloseVidCal = 1

def SalvarCal(self):
    global CloseVidCal, CloseVidTCal
    Px2cmDict = LeerPx2cm()
    ListKeys = []
    for ListK in Px2cmDict:
```

```
            ListKeys.append(ListK)
        Px2cmDict['HSV'] = [self.HLNumCal.text(),
self.SLNumCal.text(),self.VLNumCal.text(),
        self.HHNumCal.text(), self.SHNumCal.text(), self.VHNumCal.text()]
        Px2cmDict['Px2cm'] =
[str(self.DistanceDots.toPlainText()),str(self.Px2cmBrowser.toPlainText())]
        EscribirPx2cm(Px2cmDict)
        if CloseVidTCal == 1:
            CloseVidCal = 1

    def Calibration(self):
        global CloseVidTCal, CentrosCal
        if CloseVidTCal == 1:
            CentrosCal = 1

    def retranslateUi(self, CalibrationWindow):
        CalibrationWindow.setWindowTitle(_translate("CalibrationWindow",
"Calibración", None))
        self.ParametersBoxCal.setTitle(_translate("CalibrationWindow", "Parámetros
HSV [High - Low]", None))
        self.HLLabelCal.setText(_translate("CalibrationWindow", "HL:", None))
        self.SLLabelCal.setText(_translate("CalibrationWindow", "SL:", None))
        self.VLLabelCal.setText(_translate("CalibrationWindow", "VL:", None))
        self.VHLabelCal.setText(_translate("CalibrationWindow", "VH:", None))
        self.HHLabelCal.setText(_translate("CalibrationWindow", "HH:", None))
        self.SHLabelCal.setText(_translate("CalibrationWindow", "SH:", None))
        self.OptionsBoxCal.setTitle(_translate("CalibrationWindow", "Opciones",
None))
        self.SaveButtonCal.setText(_translate("CalibrationWindow", "Guardar",
None))
```

```
        self.CancelButtonCal.setText(_translate("CalibrationWindow", "Cancelar",
None))
        self.VideoButtonCal.setText(_translate("CalibrationWindow", "Video", None))
        self.CalibrationButton.setText(_translate("CalibrationWindow", "Calibrar",
None))
        self.DistanceDotsBox.setTitle(_translate("CalibrationWindow", "Distancia
entre los puntos [cm]", None))
        self.Px2cmBox.setTitle(_translate("CalibrationWindow", "Factor de conversión
[Px2cm]", None))

# Ventana de lista
class Ui_ListWindow(object):

    # Abrir ventana de edición de parámetros
    def OpenSegmentation(self):
        if self.Lista.currentRow() != -1:
            CurrentList = Leer()
            ListaColores = []
            for ListE in CurrentList:
                ListaColores.append(ListE)
            CurrentList[ListaColores[self.Lista.currentRow()]][6] = '1'
            Escribir(CurrentList)
            self.NeWindow = QtGui.QMainWindow()
            self.ui = Ui_EditSegmentation()
            self.ui.setupUi(self.NeWindow)
            self.NeWindow.show()

    def setupUi(self, ListWindow):
        ListWindow.setObjectName(_fromUtf8("ListWindow"))
        ListWindow.setFixedSize(422, 160)
```

```
        sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Fixed,
QtGui.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(ListWindow.sizePolicy().hasHeightForWidth())
        ListWindow.setSizePolicy(sizePolicy)
        self.centralwidget = QtGui.QWidget(ListWindow)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.LabelBox = QtGui.QGroupBox(self.centralwidget)
        self.LabelBox.setGeometry(QtCore.QRect(10, 10, 261, 61))
        self.LabelBox.setObjectName(_fromUtf8("LabelBox"))
        self.Etiqueta = QtGui.QTextEdit(self.LabelBox)
        self.Etiqueta.setGeometry(QtCore.QRect(19, 23, 135, 24))
        self.Etiqueta.setObjectName(_fromUtf8("Etiqueta"))
        self.Etiqueta.setAlignment(QtCore.Qt.AlignCenter)
        self.AddButton = QtGui.QPushButton(self.LabelBox)
        self.AddButton.setGeometry(QtCore.QRect(170, 24, 75, 23))
        self.AddButton.setObjectName(_fromUtf8("AddButton"))
        self.ListBox = QtGui.QGroupBox(self.centralwidget)
        self.ListBox.setGeometry(QtCore.QRect(280, 10, 131, 131))
        self.ListBox.setObjectName(_fromUtf8("ListBox"))
        self.Lista = QtGui.QListWidget(self.ListBox)
        self.Lista.setGeometry(QtCore.QRect(10, 20, 111, 101))
        self.Lista.setObjectName(_fromUtf8("Lista"))
        self.OptionsBox = QtGui.QGroupBox(self.centralwidget)
        self.OptionsBox.setGeometry(QtCore.QRect(10, 81, 261, 60))
        self.OptionsBox.setObjectName(_fromUtf8("OptionsBox"))
        self.EditButton = QtGui.QPushButton(self.OptionsBox)
        self.EditButton.setGeometry(QtCore.QRect(10, 22, 72, 23))
        self.EditButton.setObjectName(_fromUtf8("EditButton"))
```

```
        self.DeleteButton = QtGui.QPushButton(self.OptionsBox)
        self.DeleteButton.setGeometry(QtCore.QRect(93, 22, 72, 23))
        self.DeleteButton.setObjectName(_fromUtf8("DeleteButton"))
        self.UpdateButton = QtGui.QPushButton(self.OptionsBox)
        self.UpdateButton.setGeometry(QtCore.QRect(176, 22, 75, 23))
        self.UpdateButton.setObjectName(_fromUtf8("UpdateButton"))
        ListWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtGui.QMenuBar(ListWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 422, 21))
        self.menubar.setObjectName(_fromUtf8("menubar"))
        ListWindow.setMenuBar(self.menubar)
        self.statusbar = QtGui.QStatusBar(ListWindow)
        self.statusbar.setObjectName(_fromUtf8("statusbar"))
        ListWindow.setStatusBar(self.statusbar)

        self.retranslateUi(ListWindow)
        QtCore.QObject.connect(self.DeleteButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.DeleteItem)
        QtCore.QObject.connect(self.EditButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.OpenSegmentation)
        QtCore.QObject.connect(self.UpdateButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.Actualizar)
        QtCore.QObject.connect(self.AddButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.AddList)
        QtCore.QMetaObject.connectSlotsByName(ListWindow)

        # Iniciamos con la lista del archivo
        CurrentList = Leer()
        Inc = 0
        for ListE in CurrentList:
```

```
            if (CurrentList[CurrentList.keys()[Inc]][-1]) == '1':
                self.Lista.addItem(CurrentList.keys()[Inc]+'...[OK]')
            else:
                self.Lista.addItem(CurrentList.keys()[Inc])
            Inc += 1

    def AddList(self):
        if self.Etiqueta.toPlainText() != '':
            # Actualizamos el directorio
            CurrentList = Leer()
            ListaColores = []
            for ListE in CurrentList:
                ListaColores.append(ListE)
            if (self.Etiqueta.toPlainText() in ListaColores) == False:
                CurrentList.update({str(self.Etiqueta.toPlainText()): ['0', '0', '0', '179', '255',
'255', '0', '0']})
                self.Lista.clear()
                Escribir(CurrentList)
                # Actualizamos la lista
                CurrentList = Leer()
                Inc = 0
                for ListE in CurrentList:
                    if (CurrentList[CurrentList.keys()[Inc]][-1]) == '1':
                        self.Lista.addItem(CurrentList.keys()[Inc]+'...[OK]')
                    else:
                        self.Lista.addItem(CurrentList.keys()[Inc])
                    Inc += 1
                self.Etiqueta.setText('')
                self.Etiqueta.setFocus()
                self.Etiqueta.setAlignment(QtCore.Qt.AlignCenter)
```

```
def DeleteItem(self):
    if self.Lista.currentRow() != -1:
        CurrentList = Leer()
        ListaColores = []
        for ListE in CurrentList:
            ListaColores.append(ListE)
        del CurrentList[ListaColores[self.Lista.currentRow()]]
        self.Lista.takeItem(self.Lista.currentRow())
        Escribir(CurrentList)
        # Actualizamos la lista
        self.Lista.clear()
        CurrentList = Leer()
        Inc = 0
        for ListE in CurrentList:
            if (CurrentList[CurrentList.keys()[Inc]][-1]) == '1':
                self.Lista.addItem(CurrentList.keys()[Inc]+'...[OK]')
            else:
                self.Lista.addItem(CurrentList.keys()[Inc])
            Inc += 1

def Actualizar(self):
    # Actualizamos la lista
    self.Lista.clear()
    CurrentList = Leer()
    Inc = 0
    for ListE in CurrentList:
        if (CurrentList[CurrentList.keys()[Inc]][-1]) == '1':
            self.Lista.addItem(CurrentList.keys()[Inc]+'...[OK]')
        else:
```

```
                self.Lista.addItem(CurrentList.keys()[Inc])
            Inc += 1

    def retranslateUi(self, ListWindow):
        ListWindow.setWindowTitle(_translate("ListWindow", "Lista de elementos",
None))
        self.LabelBox.setTitle(_translate("ListWindow", "Etiqueta", None))
        self.AddButton.setText(_translate("ListWindow", "Añadir", None))
        self.ListBox.setTitle(_translate("ListWindow", "Lista", None))
        self.OptionsBox.setTitle(_translate("ListWindow", "Opciones", None))
        self.EditButton.setText(_translate("ListWindow", "Editar", None))
        self.DeleteButton.setText(_translate("ListWindow", "Borrar", None))
        self.UpdateButton.setText(_translate("ListWindow", "Actualizar", None))

# Ventana de edición
class Ui_EditSegmentation(object):
    def setupUi(self, EditSegmentation):
        EditSegmentation.setObjectName(_fromUtf8("EditSegmentation"))
        EditSegmentation.setFixedSize(267, 390)
        sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Preferred,
QtGui.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(EditSegmentation.sizePolicy().hasHeightForWidth())
        EditSegmentation.setSizePolicy(sizePolicy)
        self.centralwidget = QtGui.QWidget(EditSegmentation)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.ParametersBox = QtGui.QGroupBox(self.centralwidget)
        self.ParametersBox.setGeometry(QtCore.QRect(13, 80, 241, 181))
```

```
        self.ParametersBox.setObjectName(_fromUtf8("ParametersBox"))
        self.HLLabel = QtGui.QLabel(self.ParametersBox)
        self.HLLabel.setGeometry(QtCore.QRect(17, 20, 17, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.HLLabel.setFont(font)
        self.HLLabel.setTextFormat(QtCore.Qt.AutoText)

self.HLLabel.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt
.AlignVCenter)
        self.HLLabel.setObjectName(_fromUtf8("HLLabel"))
        self.VLNum = QtGui.QLabel(self.ParametersBox)
        self.VLNum.setGeometry(QtCore.QRect(203, 74, 22, 16))

self.VLNum.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.
AlignVCenter)
        self.VLNum.setObjectName(_fromUtf8("VLNum"))
        self.SLLabel = QtGui.QLabel(self.ParametersBox)
        self.SLLabel.setGeometry(QtCore.QRect(17, 47, 16, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.SLLabel.setFont(font)

self.SLLabel.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt
.AlignVCenter)
        self.SLLabel.setObjectName(_fromUtf8("SLLabel"))
        self.VLLabel = QtGui.QLabel(self.ParametersBox)
        self.VLLabel.setGeometry(QtCore.QRect(17, 74, 16, 16))
```

```
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.VLLabel.setFont(font)

self.VLLabel.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt
.AlignVCenter)
        self.VLLabel.setObjectName(_fromUtf8("VLLabel"))
        self.VHLabel = QtGui.QLabel(self.ParametersBox)
        self.VHLabel.setGeometry(QtCore.QRect(17, 155, 18, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.VHLabel.setFont(font)

self.VHLabel.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Q
t.AlignVCenter)
        self.VHLabel.setObjectName(_fromUtf8("VHLabel"))
        self.VHNum = QtGui.QLabel(self.ParametersBox)
        self.VHNum.setGeometry(QtCore.QRect(203, 155, 22, 16))

self.VHNum.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.
AlignVCenter)
        self.VHNum.setObjectName(_fromUtf8("VHNum"))
        self.HHLabel = QtGui.QLabel(self.ParametersBox)
        self.HHLabel.setGeometry(QtCore.QRect(17, 101, 19, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.HHLabel.setFont(font)
```

```
self.HHLabel.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.HHLabel.setObjectName(_fromUtf8("HHLabel"))
        self.SHNum = QtGui.QLabel(self.ParametersBox)
        self.SHNum.setGeometry(QtCore.QRect(203, 128, 22, 16))

self.SHNum.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.SHNum.setObjectName(_fromUtf8("SHNum"))
        self.SHLabel = QtGui.QLabel(self.ParametersBox)
        self.SHLabel.setGeometry(QtCore.QRect(17, 128, 18, 16))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.SHLabel.setFont(font)

self.SHLabel.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.SHLabel.setObjectName(_fromUtf8("SHLabel"))
        self.HLNum = QtGui.QLabel(self.ParametersBox)
        self.HLNum.setGeometry(QtCore.QRect(203, 20, 22, 16))
        self.HLNum.setLayoutDirection(QtCore.Qt.LeftToRight)

self.HLNum.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)
        self.HLNum.setObjectName(_fromUtf8("HLNum"))
        self.HHNum = QtGui.QLabel(self.ParametersBox)
        self.HHNum.setGeometry(QtCore.QRect(203, 101, 22, 16))
```

```
self.HHNum.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt
.AlignVCenter)
        self.HHNum.setObjectName(_fromUtf8("HHNum"))
        self.SLNum = QtGui.QLabel(self.ParametersBox)
        self.SLNum.setGeometry(QtCore.QRect(203, 47, 22, 16))

self.SLNum.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.
AlignVCenter)
        self.SLNum.setObjectName(_fromUtf8("SLNum"))
        self.HLBarra = QtGui.QScrollBar(self.ParametersBox)
        self.HLBarra.setGeometry(QtCore.QRect(46, 20, 150, 19))
        self.HLBarra.setMaximum(179)
        self.HLBarra.setOrientation(QtCore.Qt.Horizontal)
        self.HLBarra.setObjectName(_fromUtf8("HLBarra"))
        self.SLBarra = QtGui.QScrollBar(self.ParametersBox)
        self.SLBarra.setGeometry(QtCore.QRect(46, 47, 150, 19))
        self.SLBarra.setMaximum(255)
        self.SLBarra.setOrientation(QtCore.Qt.Horizontal)
        self.SLBarra.setObjectName(_fromUtf8("SLBarra"))
        self.VLBarra = QtGui.QScrollBar(self.ParametersBox)
        self.VLBarra.setGeometry(QtCore.QRect(46, 74, 150, 19))
        self.VLBarra.setMaximum(255)
        self.VLBarra.setOrientation(QtCore.Qt.Horizontal)
        self.VLBarra.setObjectName(_fromUtf8("VLBarra"))
        self.HHBarra = QtGui.QScrollBar(self.ParametersBox)
        self.HHBarra.setGeometry(QtCore.QRect(46, 101, 150, 19))
        self.HHBarra.setMaximum(179)
        self.HHBarra.setSliderPosition(179)
        self.HHBarra.setOrientation(QtCore.Qt.Horizontal)
```

```
self.HHBarra.setObjectName(_fromUtf8("HHBarra"))
self.SHBarra = QtGui.QScrollBar(self.ParametersBox)
self.SHBarra.setGeometry(QtCore.QRect(46, 128, 150, 19))
self.SHBarra.setMaximum(255)
self.SHBarra.setSliderPosition(255)
self.SHBarra.setOrientation(QtCore.Qt.Horizontal)
self.SHBarra.setObjectName(_fromUtf8("SHBarra"))
self.VHBarra = QtGui.QScrollBar(self.ParametersBox)
self.VHBarra.setGeometry(QtCore.QRect(46, 155, 150, 19))
self.VHBarra.setMaximum(255)
self.VHBarra.setSliderPosition(255)
self.VHBarra.setOrientation(QtCore.Qt.Horizontal)
self.VHBarra.setObjectName(_fromUtf8("VHBarra"))
self.LabelBoxEdit = QtGui.QGroupBox(self.centralwidget)
self.LabelBoxEdit.setGeometry(QtCore.QRect(13, 10, 241, 61))
self.LabelBoxEdit.setObjectName(_fromUtf8("LabelBoxEdit"))
self.EtiquetaEdit = QtGui.QTextEdit(self.LabelBoxEdit)
self.EtiquetaEdit.setGeometry(QtCore.QRect(30, 20, 180, 29))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.EtiquetaEdit.setFont(font)
self.EtiquetaEdit.setObjectName(_fromUtf8("EtiquetaEdit"))
self.OptionsBoxEdit = QtGui.QGroupBox(self.centralwidget)
self.OptionsBoxEdit.setGeometry(QtCore.QRect(13, 270, 241, 101))
self.OptionsBoxEdit.setObjectName(_fromUtf8("OptionsBoxEdit"))
self.SaveButton = QtGui.QPushButton(self.OptionsBoxEdit)
self.SaveButton.setGeometry(QtCore.QRect(30, 20, 75, 30))
self.SaveButton.setObjectName(_fromUtf8("SaveButton"))
```

```
        self.CancelButton = QtGui.QPushButton(self.OptionsBoxEdit)
        self.CancelButton.setGeometry(QtCore.QRect(138, 20, 75, 30))
        self.CancelButton.setObjectName(_fromUtf8("CancelButton"))
        self.VideoButton = QtGui.QPushButton(self.OptionsBoxEdit)
        self.VideoButton.setGeometry(QtCore.QRect(84, 58, 75, 30))
        self.VideoButton.setObjectName(_fromUtf8("VideoButton"))
        EditSegmentation.setCentralWidget(self.centralwidget)
        self.menubar = QtGui.QMenuBar(EditSegmentation)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 267, 21))
        self.menubar.setObjectName(_fromUtf8("menubar"))
        EditSegmentation.setMenuBar(self.menubar)
        self.statusbar = QtGui.QStatusBar(EditSegmentation)
        self.statusbar.setObjectName(_fromUtf8("statusbar"))
        EditSegmentation.setStatusBar(self.statusbar)

        CurrentList = Leer()
        ListaColores = []
        for ListE in CurrentList:
            ListaColores.append(ListE)
        for L in range(0,len(ListaColores)):
            if (CurrentList[ListaColores[L]][6]) == '1':
                # Iniciamos el nombre del color a editar
                self.EtiquetaEdit.setPlainText(CurrentList.keys()[L])
                self.EtiquetaEdit.setFocus()
                self.EtiquetaEdit.setAlignment(QtCore.Qt.AlignCenter)
                # Iniciamos las barras y textos
                self.HLBarra.setSliderPosition(int(CurrentList[ListaColores[L]][0]))
                self.HLNum.setText(_translate("EditSegmentation",
CurrentList[ListaColores[L]][0], None))
                self.SLBarra.setSliderPosition(int(CurrentList[ListaColores[L]][1]))
```

```
            self.SLNum.setText(_translate("EditSegmentation",
CurrentList[ListaColores[L]][1], None))
            self.VLBarra.setSliderPosition(int(CurrentList[ListaColores[L]][2]))
            self.VLNum.setText(_translate("EditSegmentation",
CurrentList[ListaColores[L]][2], None))
            self.HHBarra.setSliderPosition(int(CurrentList[ListaColores[L]][3]))
            self.HHNum.setText(_translate("EditSegmentation",
CurrentList[ListaColores[L]][3], None))
            self.SHBarra.setSliderPosition(int(CurrentList[ListaColores[L]][4]))
            self.SHNum.setText(_translate("EditSegmentation",
CurrentList[ListaColores[L]][4], None))
            self.VHBarra.setSliderPosition(int(CurrentList[ListaColores[L]][5]))
            self.VHNum.setText(_translate("EditSegmentation",
CurrentList[ListaColores[L]][5], None))
            break

        self.retranslateUi(EditSegmentation)
        QtCore.QObject.connect(self.HLBarra,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.HLNum.setNum)
        QtCore.QObject.connect(self.SLBarra,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.SLNum.setNum)
        QtCore.QObject.connect(self.VLBarra,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.VLNum.setNum)
        QtCore.QObject.connect(self.HHBarra,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.HHNum.setNum)
        QtCore.QObject.connect(self.SHBarra,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.SHNum.setNum)
        QtCore.QObject.connect(self.VHBarra,
QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.VHNum.setNum)
```

```
        QtCore.QObject.connect(self.SaveButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.Salvar)
        QtCore.QObject.connect(self.SaveButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), EditSegmentation.close)
        QtCore.QObject.connect(self.CancelButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.Cerrar)
        QtCore.QObject.connect(self.CancelButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), EditSegmentation.close)
        QtCore.QObject.connect(self.VideoButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.StarVideo)
        QtCore.QMetaObject.connectSlotsByName(EditSegmentation)

    def Salvar(self):
        global CloseVid, CloseVidT
        CurrentList = Leer()
        ListaColores = []
        for ListE in CurrentList:
            ListaColores.append(ListE)
        for L in range(0,len(ListaColores)):
            if (CurrentList[ListaColores[L]][6]) == '1':
                CurrentList[str(self.EtiquetaEdit.toPlainText())] =
CurrentList.pop(CurrentList.keys()[L])
                CurrentList.update({str(self.EtiquetaEdit.toPlainText()):
[self.HLNum.text(), self.SLNum.text(),self.VLNum.text(),
                    self.HHNum.text(), self.SHNum.text(), self.VHNum.text(), '0', '1']})
                Escribir(CurrentList)
                if CloseVidT == 1:
                    CloseVid = 1

    def StarVideo(self):
```

```
        global Alto, Ancho, CloseVid, CloseVidT, kernel
        CloseVidT = 1
        # Iniciamos el video
        cap = cv.VideoCapture(0)
        # Establecemos la resolución de la cámara web
        cap.set(cv.CAP_PROP_FRAME_WIDTH, Ancho)
        cap.set(cv.CAP_PROP_FRAME_HEIGHT, Alto)
        while CloseVid == 0:
            _, frame = cap.read()
            # Convertimos al espacio de color HSV
            hsv_temp = cv.cvtColor(frame,cv.COLOR_BGR2HSV)
            # Pasamos por un filtro gaussiano
            hsv = cv.GaussianBlur(hsv_temp, (5, 5), 0)
            # Máscara
            lower =
np.array([int(self.HLNum.text()),int(self.SLNum.text()),int(self.VLNum.text())])
            upper =
np.array([int(self.HHNum.text()),int(self.SHNum.text()),int(self.VHNum.text())])
            mask = cv.inRange(hsv, lower, upper)
            #Filtrar el ruido aplicando un OPEN seguido de un CLOSE
            mask = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
            maks = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
            result = cv.bitwise_and(frame,frame,mask = mask)
            cv.imshow('Segmentation Window',result)
            if cv.waitKey(1) & 0xFF == 27:
                break
        cap.release()
        cv.destroyWindow('Segmentation Window')
        CloseVid = 0
        CloseVidT = 0
```

```
    def Cerrar(self):
        global CloseVid
        CurrentList = Leer()
        ListaColores = []
        for ListE in CurrentList:
            ListaColores.append(ListE)
        for L in range(0,len(ListaColores)):
            if CurrentList[ListaColores[L]][6] == '1':
                CurrentList[ListaColores[L]][6] = '0'
                break
        Escribir(CurrentList)
        CloseVid = 1

    def retranslateUi(self, EditSegmentation):
        EditSegmentation.setWindowTitle(_translate("EditSegmentation", "Editor de
parámetros", None))
        self.ParametersBox.setTitle(_translate("EditSegmentation", "Parámetros HSV
[High - Low]", None))
        self.HLLabel.setText(_translate("EditSegmentation", "HL:", None))
        self.SLLabel.setText(_translate("EditSegmentation", "SL:", None))
        self.VLLabel.setText(_translate("EditSegmentation", "VL:", None))
        self.VHLabel.setText(_translate("EditSegmentation", "VH:", None))
        self.HHLabel.setText(_translate("EditSegmentation", "HH:", None))
        self.SHLabel.setText(_translate("EditSegmentation", "SH:", None))
        self.LabelBoxEdit.setTitle(_translate("EditSegmentation", "Etiqueta", None))
        self.OptionsBoxEdit.setTitle(_translate("EditSegmentation", "Opciones",
None))
        self.SaveButton.setText(_translate("EditSegmentation", "Guardar", None))
        self.CancelButton.setText(_translate("EditSegmentation", "Cancelar", None))
```

```
        self.VideoButton.setText(_translate("EditSegmentation", "Video", None))

class Ui_Principal(object):

    def OpenSeg(self):
        self.NeWindow = QtGui.QMainWindow()
        self.ui = Ui_ListWindow()
        self.ui.setupUi(self.NeWindow)
        self.NeWindow.show()
        self.StopMainVideo()

    def OpenCal(self):
        self.NeWindow = QtGui.QMainWindow()
        self.ui = Ui_CalibrationWindow()
        self.ui.setupUi(self.NeWindow)
        self.NeWindow.show()
        self.StopMainVideo()

    def StartMainVideo(self):
        global Alto, Ancho, CloseMainVid, MainStart, DrawObj, PF, ContP, Ds,
VBorde, VAssig, VLabel, Rtemp, Ptemp
        global Puntos, StarRecVideo,fourcc, Vout, SaveCont, Ready, StartCOM,
Arduino, StartTime, TLim, CT1
        self.StartMainVideoButton.setStyleSheet("color: red")
        MainStart = 1
        SegDirectory = Leer()
        ListaColores = []
        for ListE in SegDirectory:
            ListaColores.append(ListE)
        Px2cmDict = LeerPx2cm()
```

```
        ListKeys = []
        for ListK in Px2cmDict:
            ListKeys.append(ListK)
        Px2cm = float(str(Px2cmDict[ListKeys[1]][1]))
        cap = cv.VideoCapture(0)
        cap.set(cv.CAP_PROP_FRAME_WIDTH, Ancho)
        cap.set(cv.CAP_PROP_FRAME_HEIGHT, Alto)
        while CloseMainVid == 0:
            Out = SegGPS(cap,SegDirectory,ListaColores,Px2cm)
            RobotsPx = getPxRo(Out[1])
            if VLabel == 1:
                for R in range(0,len(RobotsPx)):
                    cv.putText(Out[0], str(ListaColores[Out[1].keys()[R]]),
(int(RobotsPx[R][0])-20, int(RobotsPx[R][1])+30), cv.FONT_HERSHEY_SIMPLEX,
0.5, (255, 255, 255), 1)
            if DrawObj == 1:
                if VBorde == 1:
                    cv.rectangle(Out[0], (50,50), (Ancho-50,Alto-50), (0, 0, 255),
thickness=1, lineType=8, shift=0)
                cv.setMouseCallback('Main Window', on_mouse)
                if ContP > 0:
                    for P in range(0,len(Puntos)):
                        cv.circle(Out[0], (Puntos[P][0], Puntos[P][1]), 4, (255, 0, 0), -1)
                        if VLabel == 1:
                            cv.putText(Out[0], str(P+1), (Puntos[P][0]+10, Puntos[P][1]-10),
cv.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
            # Asignación de los robots
            if len(RobotsPx)>0 and ContP>0:
                Ready = 1 # Bandera para comunicación
```

```
            if (len(RobotsPx) != Rtemp) or (ContP != Ptemp): # Mantener la
asignación hasta que cambien las condiciones
                Assigtemp = 1
            if Assigtemp == 1:
                Ord = AsRcPc(DE(RobotsPx,Puntos),RobotsPx,Puntos) # [Puntos,
Robots]
                Assigtemp = 0
                Rtemp = len(RobotsPx)
                Ptemp = ContP
            if VAssig == 1:
                for arrow in Ord:
                    PosRob = np.argwhere(RobotsPx==arrow[1])
                    cv.arrowedLine(Out[0],
(int(RobotsPx[PosRob[0][0]][0]),int(RobotsPx[PosRob[0][0]][1])),
                            (int(Puntos[int(arrow[0])-1][0]),int(Puntos[int(arrow[0])-1][1])),
(0,255,0), 1, 8, 0, 0.1)
        else:
            Ready = 0
        # Segmento de comunicación
        if StartCOM == 1:
             Data = np.zeros([len(RobotsPx),3])
            ConR = 0
            zero = 0
            if Ready == 1:
                for RobotPrincipal in RobotsPx[0:,2:]:
                    RobotsXY = getXYRo(Out[1])
                    PuntosXY = getXYDots(Puntos,Px2cm)
                    OrRad = getRad(Out[1])
                    RobTemp = RobotsPx
                    RobTempXY = RobotsXY
```

```
                    # El robot está en el plano cartesiano
                    if (RobotPrincipal in RobotsPx[0:,2:]) == True:
                        # El robot tiene asignado un punto
                        if (RobotPrincipal in Ord[0:,1:]) == True:
                            SelRob = np.argwhere(RobTemp[0:,2:]==RobotPrincipal)
                            RobTemp = np.delete(RobTemp,SelRob[0][0],0)
                            RobTempXY = np.delete(RobTempXY,SelRob[0][0],0)
                            DObs = np.zeros([len(RobTemp),1])
                            AssigDot = np.argwhere(Ord[0:,1:]==RobotPrincipal)
                            FAD = int(Ord[AssigDot[0][0]][0])-1
                            if len(DObs)>0:
                                for Obs in range(0,len(RobTemp)):
                                    DObs[Obs] = math.sqrt((RobTemp[Obs][0]-
RobotsPx[SelRob[0][0]][0])**2 + (RobTemp[Obs][1]-RobotsPx[SelRob[0][0]][1])**2)
                                    # cv.arrowedLine(Out[0],
(int(RobotsPx[SelRob[0][0]][0]),int(RobotsPx[SelRob[0][0]][1])),
(int(RobTemp[Obs][0]),int(RobTemp[Obs][1])),(0,0,255), 1, 8, 0, 0.1)
                                AnglePlus = 1.5708
                                AngF = math.atan2((PuntosXY[FAD][1]-
RobotsXY[SelRob[0][0]][1]),(PuntosXY[FAD][0]-
RobotsXY[SelRob[0][0]][0]))*180/math.pi
                                AngF = AngF + (AngF < 0)*360
                                AngFT = AngF + 180
                                if all(DObs>Ds) == True:
                                    AngOA = 0
                                else:
                                    locmin = (min(map(min,DObs)))
                                    Indmin = np.argwhere(DObs==locmin)
```

```
                        AngObs = math.atan2((RobTempXY[Indmin[0][0]][1]-
RobotsXY[SelRob[0][0]][1]),(RobTempXY[Indmin[0][0]][0]-
RobotsXY[SelRob[0][0]][0]))*180/math.pi
                        AngObs = AngObs + (AngObs < 0)*360
                        # cv.arrowedLine(Out[0],
(int(RobotsPx[SelRob[0][0]][0]),int(RobotsPx[SelRob[0][0]][1])),
(int(RobTemp[Indmin[0][0]][0]),int(RobTemp[Indmin[0][0]][1])),(255,0,0), 1, 8, 0,
0.1)
                        if AngF>=0 and AngF<=180:
                            if AngObs>=AngF and AngObs<=AngFT:
                                AngOA = -AnglePlus
                            else:
                                AngOA = AnglePlus
                        else:
                            if AngObs<AngF and AngObs>(AngFT-360):
                                AngOA = AnglePlus
                            else:
                                AngOA = -AnglePlus
                    else:
                        AngOA = 0
                    Dp = 10.0 # Distancia al punto guía virtual
                    L = 7.2 # Distancia entre las ruedas
                    r = 1.6 # Radio de las ruedas
                    X = PuntosXY[FAD][0] - RobotsXY[SelRob[0][0]][0]
                    Y = PuntosXY[FAD][1] - RobotsXY[SelRob[0][0]][1]
                    OrPos = np.argwhere(OrRad==RobotPrincipal)
                    Angle = -OrRad[OrPos[0][0]][0] + AngOA
                    A = np.matrix(((1,0),(0,1/Dp)))
                    R = np.matrix(((math.cos(Angle),-
math.sin(Angle)),(math.sin(Angle),math.cos(Angle))))
```

```
                B = np.matrix(((X,),(Y,)))
                S = A*R*B
                v = float(S[0])*4.0
                w = float(S[1])*4.0
                wd = (2*v + L*w)/(2*r)
                wi = (2*v - L*w)/(2*r)
            else:
                wd = 0
                wi = 0
        else:
            wd = 0
            wi = 0
        Data[ConR] = [wd,wi,int(RobotPrincipal)]
        ConR += 1
else:
    zero = 1
if zero == 0:
    if CT1 == 1:
        DataT = Data
        CT1 = 0
        Send = 0
        SendD = getColor(ListaColores,DataT[Send])
        Arduino.write(bytes(b'%r' % SendD))
        print SendD
        ST = time.time()
    PT = time.time()-ST
    if PT >= TLim:
        if len(DataT) == 1:
            print '-----'
            CT1 = 1
```

```
                    else:
                        Send += 1
                        SendD = getColor(ListaColores,DataT[Send])
                        Arduino.write(bytes(b'%r' % SendD))
                        print SendD
                        ST = time.time()
                        if Send == len(DataT)-1:
                            print '-----'
                            CT1 = 1
                else:
                    SendD = '<#>'
                    Arduino.write(bytes(b'%r' % SendD))
                    # Pendiente reiniciar variables de tiempo, se está quedando una
pegada
                    print SendD
                    print '-----'
                    CT1 = 1

            if StarRecVideo == 1:
                Passtime = time.time()-StartTime
                cv.putText(Out[0], ('Tiempo = '+str(round(Passtime,2))+' [s]'), (Ancho-
160, Alto-10), cv.FONT_HERSHEY_SIMPLEX, 0.43, (0, 0, 255), 1)
                Vid = Out[0]
                Vout.write(Vid)
            cv.imshow('Main Window', Out[0])
            if cv.waitKey(1) & 0xFF == 27:
                self.StopProcess()
                break
        cap.release()
        cv.destroyWindow('Main Window')
```

```
        CloseMainVid = 0
        MainStart = 0
        DrawObj = 0
        PF = 0
        ContP = 0
        Puntos = np.array([[0,0]])
        Ready = 0
        if Arduino != 0:
            Arduino.close()
        self.StartMainVideoButton.setStyleSheet("color: black")
        self.ActiveObjButton.setStyleSheet("color: black")

    def StopMainVideo(self):
        global MainStart, CloseMainVid, StarRecVideo
        if  MainStart == 1:
            CloseMainVid = 1
            self.ActiveObjButton.setStyleSheet("color: black")
            if StarRecVideo == 1:
                self.StopRecordVideo()

    def ActiveObj(self):
        global MainStart, DrawObj
        if  MainStart == 1:
            self.ActiveObjButton.setStyleSheet("color: red")
            DrawObj = 1

    def CleanObj(self):
        global MainStart, PF, ContP, Puntos
        if  MainStart == 1:
            PF = 0
```

```
            ContP = 0
            Puntos = np.array([[0,0]])

    def SaveRecordVideo(self):
        global StarRecVideo, fourcc, Vout, SaveCont, Ancho, Alto, MainStart,
StartTime
        if MainStart == 1:
            self.SaveVideoButton.setStyleSheet("color: red")
            F = open('VidSave.txt','r')
            SaveCont = F.read()
            SaveCont = str(int(SaveCont)+1)
            F.close()
            fourcc = cv.VideoWriter_fourcc(*'MJPG')
            Vout = cv.VideoWriter('Salida_'+SaveCont+'.avi', fourcc, 10.0, (Ancho,Alto))
            StarRecVideo = 1
            StartTime = time.time()

    def StopRecordVideo(self):
        global StarRecVideo, fourcc, Vout, SaveCont, MainStart
        if MainStart == 1:
            self.SaveVideoButton.setStyleSheet("color: black")
            if StarRecVideo == 1:
                StarRecVideo = 0
                F = open('VidSave.txt','w')
                F.write(SaveCont)
                F.close()
                Vout.release()

    def CloseMainWindow(self):
        global CloseMainVid, StarRecVideo
```

```
    CloseMainVid = 1
    if StarRecVideo == 1:
        self.StopRecordVideo()
    cv.destroyAllWindows()

def StartProcess(self):
    global Ready, StartCOM, Arduino
    if Ready == 1:
        self.StartProcessButton.setStyleSheet("color: red")
        Arduino = serial.Serial('COM24', 115200, timeout = 1)
        print 'Opening Serial port...'
        time.sleep(1)
        print 'Start...'
        StartCOM = 1

def StopProcess(self):
    global StartCOM, Arduino
    if StartCOM == 1:
        self.StartProcessButton.setStyleSheet("color: Black")
        StartCOM = 0
        Arduino.close()

def ContCheck(self):
    global VCont
    if VCont == 1:
        VCont = 0
    else:
        VCont = 1

def OrCheck(self):
```

```
    global VOr
    if VOr == 1:
        VOr = 0
    else:
        VOr = 1

def PosCheck(self):
    global VPosition
    if VPosition == 1:
        VPosition = 0
    else:
        VPosition = 1

def ProxCheck(self):
    global VProx
    if VProx == 1:
        VProx = 0
    else:
        VProx = 1

def EdgeCheck(self):
    global VBorde
    if VBorde == 1:
        VBorde = 0
    else:
        VBorde = 1

def AssigCheck(self):
    global VAssig
    if VAssig == 1:
```

```
            VAssig = 0
        else:
            VAssig = 1

    def LabelCheck(self):
        global VLabel
        if VLabel == 1:
            VLabel = 0
        else:
            VLabel = 1

    def setupUi(self, Principal):
        Principal.setObjectName(_fromUtf8("Principal"))
        Principal.setWindowModality(QtCore.Qt.NonModal)
        Principal.setFixedSize(258, 330) #
        Principal.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)
        Principal.setAutoFillBackground(False)
        Principal.setToolButtonStyle(QtCore.Qt.ToolButtonIconOnly)
        Principal.setAnimated(True)
        Principal.setDocumentMode(False)
        Principal.setTabShape(QtGui.QTabWidget.Rounded)
        Principal.setDockNestingEnabled(False)

Principal.setDockOptions(QtGui.QMainWindow.AllowTabbedDocks|QtGui.QMainW
indow.AnimatedDocks)
        Principal.setUnifiedTitleAndToolBarOnMac(False)
        self.centralwidget = QtGui.QWidget(Principal)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.GeneralVideoBox = QtGui.QGroupBox(self.centralwidget)
        self.GeneralVideoBox.setGeometry(QtCore.QRect(9, 2, 241, 191))
```

```
self.GeneralVideoBox.setObjectName(_fromUtf8("GeneralVideoBox"))
self.VisibleBox = QtGui.QGroupBox(self.GeneralVideoBox)
self.VisibleBox.setGeometry(QtCore.QRect(110, 14, 121, 164))
self.VisibleBox.setObjectName(_fromUtf8("VisibleBox"))
self.EdgeCheckBox = QtGui.QCheckBox(self.VisibleBox)
self.EdgeCheckBox.setGeometry(QtCore.QRect(20, 20, 70, 17))
self.EdgeCheckBox.setChecked(True)
self.EdgeCheckBox.setObjectName(_fromUtf8("EdgeCheckBox"))
self.PositionCheckBox = QtGui.QCheckBox(self.VisibleBox)
self.PositionCheckBox.setGeometry(QtCore.QRect(20, 40, 70, 17))
self.PositionCheckBox.setChecked(True)
self.PositionCheckBox.setObjectName(_fromUtf8("PositionCheckBox"))
self.OrientationCheckBox = QtGui.QCheckBox(self.VisibleBox)
self.OrientationCheckBox.setGeometry(QtCore.QRect(20, 60, 91, 17))
self.OrientationCheckBox.setChecked(True)
self.OrientationCheckBox.setObjectName(_fromUtf8("OrientationCheckBox"))
self.ContourCheckBox = QtGui.QCheckBox(self.VisibleBox)
self.ContourCheckBox.setGeometry(QtCore.QRect(20, 80, 81, 17))
self.ContourCheckBox.setChecked(True)
self.ContourCheckBox.setObjectName(_fromUtf8("ContourCheckBox"))
self.DotLabelCheckBox = QtGui.QCheckBox(self.VisibleBox)
self.DotLabelCheckBox.setGeometry(QtCore.QRect(20, 120, 101, 17))
self.DotLabelCheckBox.setChecked(True)
self.DotLabelCheckBox.setObjectName(_fromUtf8("DotLabelCheckBox"))
self.ProxCheckBox = QtGui.QCheckBox(self.VisibleBox)
self.ProxCheckBox.setGeometry(QtCore.QRect(20, 100, 101, 17))
self.ProxCheckBox.setChecked(True)
self.ProxCheckBox.setObjectName(_fromUtf8("ProxCheckBox"))
self.AssignationCheckBox = QtGui.QCheckBox(self.VisibleBox)
self.AssignationCheckBox.setGeometry(QtCore.QRect(20, 140, 70, 17))
```

```
        self.AssignationCheckBox.setChecked(True)

self.AssignationCheckBox.setObjectName(_fromUtf8("AssignationCheckBox"))
        self.VideoBox = QtGui.QGroupBox(self.GeneralVideoBox)
        self.VideoBox.setGeometry(QtCore.QRect(10, 20, 91, 71))
        self.VideoBox.setTitle(_fromUtf8(""))
        self.VideoBox.setObjectName(_fromUtf8("VideoBox"))
        self.StartMainVideoButton = QtGui.QPushButton(self.VideoBox)
        self.StartMainVideoButton.setGeometry(QtCore.QRect(8, 9, 75, 23))

self.StartMainVideoButton.setObjectName(_fromUtf8("StartMainVideoButton"))
        self.StopMainVideoButton = QtGui.QPushButton(self.VideoBox)
        self.StopMainVideoButton.setGeometry(QtCore.QRect(8, 39, 75, 23))

self.StopMainVideoButton.setObjectName(_fromUtf8("StopMainVideoButton"))
        self.RecordBox = QtGui.QGroupBox(self.GeneralVideoBox)
        self.RecordBox.setGeometry(QtCore.QRect(10, 97, 91, 81))
        self.RecordBox.setObjectName(_fromUtf8("RecordBox"))
        self.SaveVideoButton = QtGui.QPushButton(self.RecordBox)
        self.SaveVideoButton.setGeometry(QtCore.QRect(8, 20, 75, 23))
        self.SaveVideoButton.setObjectName(_fromUtf8("SaveVideoButton"))
        self.StopVideoButton = QtGui.QPushButton(self.RecordBox)
        self.StopVideoButton.setGeometry(QtCore.QRect(8, 50, 75, 23))
        self.StopVideoButton.setObjectName(_fromUtf8("StopVideoButton"))
        self.ObjectivesBox = QtGui.QGroupBox(self.centralwidget)
        self.ObjectivesBox.setGeometry(QtCore.QRect(135, 198, 115, 92))
        self.ObjectivesBox.setObjectName(_fromUtf8("ObjectivesBox"))
        self.ActiveObjButton = QtGui.QPushButton(self.ObjectivesBox)
        self.ActiveObjButton.setGeometry(QtCore.QRect(20, 21, 75, 23))
        self.ActiveObjButton.setObjectName(_fromUtf8("ActiveObjButton"))
```

```
self.CleanObjButton = QtGui.QPushButton(self.ObjectivesBox)
self.CleanObjButton.setGeometry(QtCore.QRect(20, 55, 75, 23))
self.CleanObjButton.setObjectName(_fromUtf8("CleanObjButton"))
self.ProcessBox = QtGui.QGroupBox(self.centralwidget)
self.ProcessBox.setGeometry(QtCore.QRect(9, 198, 115, 92))
self.ProcessBox.setObjectName(_fromUtf8("ProcessBox"))
self.StartProcessButton = QtGui.QPushButton(self.ProcessBox)
self.StartProcessButton.setGeometry(QtCore.QRect(20, 21, 75, 23))
self.StartProcessButton.setObjectName(_fromUtf8("StartProcessButton"))
self.StopProcessButton = QtGui.QPushButton(self.ProcessBox)
self.StopProcessButton.setGeometry(QtCore.QRect(20, 55, 75, 23))
self.StopProcessButton.setObjectName(_fromUtf8("StopProcessButton"))
Principal.setCentralWidget(self.centralwidget)
self.menubar = QtGui.QMenuBar(Principal)
self.menubar.setGeometry(QtCore.QRect(0, 0, 258, 21))
self.menubar.setObjectName(_fromUtf8("menubar"))
self.menuArchivo = QtGui.QMenu(self.menubar)
self.menuArchivo.setObjectName(_fromUtf8("menuArchivo"))
self.menuVideo = QtGui.QMenu(self.menubar)
self.menuVideo.setObjectName(_fromUtf8("menuVideo"))
Principal.setMenuBar(self.menubar)
self.statusbar = QtGui.QStatusBar(Principal)
self.statusbar.setObjectName(_fromUtf8("statusbar"))
Principal.setStatusBar(self.statusbar)
self.actionSalir = QtGui.QAction(Principal)
self.actionSalir.setObjectName(_fromUtf8("actionSalir"))
self.actionSalir.triggered.connect(self.CloseMainWindow) #
self.actionSalir.triggered.connect(Principal.close) #
self.actionSegmentation = QtGui.QAction(Principal)
self.actionSegmentation.setObjectName(_fromUtf8("actionSegmentation"))
```

```
        self.actionSegmentation.triggered.connect(self.OpenSeg) #
        self.actionCalibration = QtGui.QAction(Principal)
        self.actionCalibration.setObjectName(_fromUtf8("actionCalibration"))
        self.actionCalibration.triggered.connect(self.OpenCal) #
        self.menuArchivo.addAction(self.actionSalir)
        self.menuVideo.addAction(self.actionSegmentation)
        self.menuVideo.addAction(self.actionCalibration)
        self.menubar.addAction(self.menuArchivo.menuAction())
        self.menubar.addAction(self.menuVideo.menuAction())

        self.retranslateUi(Principal)
        QtCore.QObject.connect(self.ActiveObjButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.ActiveObj)
        QtCore.QObject.connect(self.CleanObjButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.CleanObj)
        QtCore.QObject.connect(self.StartProcessButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.StartProcess)
        QtCore.QObject.connect(self.StopProcessButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.StopProcess)
        QtCore.QObject.connect(self.StartMainVideoButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.StartMainVideo)
        QtCore.QObject.connect(self.StopMainVideoButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.StopMainVideo)
        QtCore.QObject.connect(self.SaveVideoButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.SaveRecordVideo)
        QtCore.QObject.connect(self.StopVideoButton,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.StopRecordVideo)
        QtCore.QObject.connect(self.AssignationCheckBox,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.AssigCheck)
```

```
        QtCore.QObject.connect(self.DotLabelCheckBox,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.LabelCheck)
        QtCore.QObject.connect(self.ProxCheckBox,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.ProxCheck)
        QtCore.QObject.connect(self.ContourCheckBox,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.ContCheck)
        QtCore.QObject.connect(self.OrientationCheckBox,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.OrCheck)
        QtCore.QObject.connect(self.PositionCheckBox,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.PosCheck)
        QtCore.QObject.connect(self.EdgeCheckBox,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.EdgeCheck)
        QtCore.QMetaObject.connectSlotsByName(Principal)

    def retranslateUi(self, Principal):
        Principal.setWindowTitle(_translate("Principal", "Control de formaciones",
None))
        self.GeneralVideoBox.setTitle(_translate("Principal", "Video", None))
        self.VisibleBox.setTitle(_translate("Principal", "En pantalla", None))
        self.EdgeCheckBox.setText(_translate("Principal", "Borde", None))
        self.PositionCheckBox.setText(_translate("Principal", "Posición", None))
        self.OrientationCheckBox.setText(_translate("Principal", "Orientación", None))
        self.ContourCheckBox.setText(_translate("Principal", "Contorno", None))
        self.DotLabelCheckBox.setText(_translate("Principal", "Etiquetas", None))
        self.ProxCheckBox.setText(_translate("Principal", "Proximidad", None))
        self.AssignationCheckBox.setText(_translate("Principal", "Asignación", None))
        self.StartMainVideoButton.setText(_translate("Principal", "Iniciar", None))
        self.StopMainVideoButton.setText(_translate("Principal", "Detener", None))
        self.RecordBox.setTitle(_translate("Principal", "Grabación", None))
        self.SaveVideoButton.setText(_translate("Principal", "Iniciar", None))
```

```
        self.StopVideoButton.setText(_translate("Principal", "Detener", None))
        self.ObjectivesBox.setTitle(_translate("Principal", "Puntos", None))
        self.ActiveObjButton.setText(_translate("Principal", "Activar", None))
        self.CleanObjButton.setText(_translate("Principal", "Limpiar", None))
        self.ProcessBox.setTitle(_translate("Principal", "Proceso", None))
        self.StartProcessButton.setText(_translate("Principal", "Iniciar", None))
        self.StopProcessButton.setText(_translate("Principal", "Detener", None))
        self.menuArchivo.setTitle(_translate("Principal", "Archivo", None))
        self.menuVideo.setTitle(_translate("Principal", "Video", None))
        self.actionSalir.setText(_translate("Principal", "Salir", None))
        self.actionSegmentation.setText(_translate("Principal", "Segmentación",
None))
        self.actionCalibration.setText(_translate("Principal", "Calibración", None))

if __name__ == "__main__":
    import sys
    app = QtGui.QApplication(sys.argv)
    Principal = QtGui.QMainWindow()
    ui = Ui_Principal()
    ui.setupUi(Principal)
    Principal.show()
    sys.exit(app.exec_())
```

ANEXO 23. CÓDIGO DEL ARDUINO MEGA MAESTRO

```
#include <RF24Network.h>
#include <RF24.h>
#include <SPI.h>

//Variables para la recepción desde Python
const byte NumChars = 17;
char ReChars[NumChars];
boolean newData = false;
float Ws[7][2] = {{0,0},{0,0},{0,0},{0,0},{0,0},{0,0},{0,0}};
int Color;
int ColorFlag = 0;

//Declaremos los pines CE y el CSN
#define CE_PIN 48
#define CSN_PIN 49
//MISO: 50 / MOSI: 51 / SCK: 52

//Declaramos el vector con los datos a enviar
float datos_E[2], datos_E1[2], datos_E2[2], datos_E3[2], datos_E4[2], datos_E5[2],
datos_E6[2];

//creamos el objeto radio (NRF24L01)
RF24 radio(CE_PIN, CSN_PIN);

RF24Network network(radio);          // Network uses that radio

const uint16_t this_node = 00;        // Address of our node in Octal format
const uint16_t other_node1 = 01;       // Address of the other node in Octal format
```

```
const uint16_t other_node2 = 02;
const uint16_t other_node3 = 03;
const uint16_t other_node4 = 04;
const uint16_t other_node5 = 05;
const uint16_t other_node6 = 011; // Hijo de (01)
RF24NetworkHeader azul(other_node1);
RF24NetworkHeader rojo(other_node2);
RF24NetworkHeader amarillo(other_node3);
RF24NetworkHeader fucsia(other_node4);
RF24NetworkHeader naranja(other_node5);
RF24NetworkHeader verde(other_node6);

void setup(void)
{
  Serial.begin(115200);
  SPI.begin();
  radio.begin();
  network.begin(90,this_node);
}

//Obtenemos el string con los valores de las velocidades
void RecFromPy()
{
  static boolean recvInProgress = false;
  static byte ndx = 0;
  char startMarker = '<';
  char endMarker = '>';
  char rc;

  while (Serial.available() > 0 && newData == false)
```

```
{
  rc = Serial.read();
  if (recvInProgress == true)
  {
    if (rc != endMarker)
    {
      if (ColorFlag == 0)
      {
        if(rc=='B')
          Color = 0;
        else if(rc=='R')
          Color = 1;
        else if(rc=='Y')
          Color = 2;
        else if(rc=='F')
          Color = 3;
        else if(rc=='O')
          Color = 4;
        else if(rc=='G')
          Color = 5;
        else if(rc=='#')
          Color = 6;
        ColorFlag = 1;
      }
      else
      {
        ReChars[ndx] = rc;
        ndx++;
        if (ndx >= NumChars)
        {
```

```
            ndx = NumChars - 1;
          }
        }
      }
      else
      {
        String Str = ReChars;
        Ws[Color][0] = Str.toFloat();
        String Str2 = Str.substring(8,16);
        Ws[Color][1] = Str2.toFloat();
        ReChars[ndx] = '\0';
        recvInProgress = false;
        newData = true;
        ColorFlag = 0;
        ndx = 0;
      }
    }
    else if (rc == startMarker)
    {
      recvInProgress = true;
    }
  }
}

void loop(void)
{
  //Obtenemos las velocidades de las ruedas
  RecFromPy();
  if (Color<6)
  {
```

```
network.update(); // Check the network regularly
if (Color == 0)
{
  datos_E1[0] = Ws[0][0];
  datos_E1[1] = Ws[0][1];
  network.write(azul,&datos_E1,sizeof(datos_E1));
}
if (Color == 1)
{
  datos_E2[0] = Ws[1][0];
  datos_E2[1] = Ws[1][1];
  network.write(rojo,&datos_E2,sizeof(datos_E2));
}
if (Color == 2)
{
  datos_E3[0] = Ws[2][0];
  datos_E3[1] = Ws[2][1];
  network.write(amarillo,&datos_E3,sizeof(datos_E3));
}
if (Color == 3)
{
  datos_E4[0] = Ws[3][0];
  datos_E4[1] = Ws[3][1];
  network.write(fucsia,&datos_E4,sizeof(datos_E4));
}
if (Color == 4)
{
  datos_E5[0] = Ws[4][0];
  datos_E5[1] = Ws[4][1];
  network.write(naranja,&datos_E5,sizeof(datos_E5));
```

```
    }
    if (Color == 5)
    {
      datos_E6[0] = Ws[5][0];
      datos_E6[1] = Ws[5][1];
      network.write(verde,&datos_E6,sizeof(datos_E6));
    }
    newData = false;
  }
  else
  {
    datos_E[0] = Ws[6][0];
    datos_E[1] = Ws[6][1];
    network.write(azul,&datos_E,sizeof(datos_E));
    network.write(rojo,&datos_E,sizeof(datos_E));
    network.write(amarillo,&datos_E,sizeof(datos_E));
    network.write(fucsia,&datos_E,sizeof(datos_E));
    network.write(naranja,&datos_E,sizeof(datos_E));
    network.write(verde,&datos_E,sizeof(datos_E));
    newData = false;
  }
}
```

ANEXO 24. CÓDIGO DEL ARDUINO MEGA ESCLAVO

```
#include <RF24Network.h>
#include <RF24.h>
#include <SPI.h>
#include <TimerFive.h>

//Declaramos las variables de los encoders
volatile int Con_1 = 0;
volatile int Con_2 = 0;
float t = 25000; //us
float RPM_Rd = 0;
float RPM_Ri = 0;
float RPMmax = 320.00;
float RPMmin = -320.00;

//Variables
int Sentido_Rd = 0;
int Sentido_Ri = 0;
float Sp_Rd = 0; // Valor del Setpoint Rd
float Sp_Ri = 0; // Valor del Setpoint Ri
float Pv_Rd = 0; // Valor de la Variable del Proceso
float Pv_Ri = 0; // Valor de la Variable del Proceso
float Kp_Rd = 0;
float Kp_Ri = 0;
float Ki_Rd = 0;
float Ki_Ri = 0;
float Kd_Rd = 0;
float Kd_Ri = 0;
float A_p_Rd = 0;
```

```
float A_p_Ri = 0;
float A_d_Rd = 0;
float A_d_Ri = 0;
float A_i_Rd = 0;
float A_i_Ri = 0;
float Uc_Rd = 0;
float Uc_Ri = 0;
float un_Rd = 0;
float un_Ri = 0;
float U0_Rd = 0;
float U0_Ri = 0;
float en_Rd = 0; // Error
float en_Ri = 0; // Error
float en_1_Rd = 0; // Error una muestra anterior
float en_1_Ri = 0; // Error una muestra anterior
float I_error_Rd = 0; //Memoria integral del error
float I_error_Ri = 0; //Memoria integral del error
float I_error_1_Rd = 0; //Memoria integral del error una muestra anterior
float I_error_1_Ri = 0; //Memoria integral del error una muestra anterior

//Declaremos los pines CE y el CSN del NRF24L01
#define CE_PIN 48
#define CSN_PIN 49

//Declaramos los pines para los motores
#define M1A 10
#define M1B 11
#define M2A 12
#define M2B 13
```

```
//Declaramos el vector con los datos recibidos
float datos_R[2];

//Declaramos el vector con los datos a enviar
float datos_E[6];

//creamos el objeto radio (NRF24L01)
RF24 radio(CE_PIN, CSN_PIN);

RF24Network network(radio);      // Network uses that radio
const uint16_t this_node = 01;    // Address of our node in Octal format ( 04,031,
etc)
// Este es el número que se debe cambiar según sea el robot (01, 02, …)
const uint16_t other_node = 00;   // Address of the other node in Octal format
// Este número corresponde al nodo maestro

void setup(void)
{
  //Declaramos las interrupciones
  Timer5.initialize(t);
  Timer5.attachInterrupt(ISR_Enc);
  attachInterrupt(4,Enc_1,RISING); //Pin 19 Encoder 1 Rd
  attachInterrupt(5,Enc_2,RISING); //Pin 18 Encoder 2 Ri

  //Declaramos el tipo de modo de los pines de los motores
  pinMode(M1A, OUTPUT);
  pinMode(M1B, OUTPUT);
  pinMode(M2A, OUTPUT);
  pinMode(M2B, OUTPUT);
  Serial.begin(115200);
```

```
  SPI.begin();
  radio.begin();
  network.begin(90,this_node);
}

void loop(void)
{
  network.update();                  // Check the network regularly
  while (network.available())
  {     // Is there anything ready for us?
    RF24NetworkHeader azul;        // If so, grab it and print it out
    network.read(azul,&datos_R,sizeof(datos_R));
    // Se reemplaza el término “azul” por el color del identificador de la plataforma
    datos_E[0] = datos_R[0];
    datos_E[1] = datos_R[1];
    datos_E[2] = RPM_Rd;
    datos_E[3] = RPM_Ri;
    datos_E[4] = un_Rd;
    datos_E[5] = un_Ri;
    Serial.print(datos_E[0]);
    Serial.print(" ");
    Serial.print(datos_E[1]);
    Serial.print(" ");
    Serial.print(datos_E[2]);
    Serial.print(" ");
    Serial.print(datos_E[3]);
    Serial.print(" ");
    Serial.print(datos_E[4]);
    Serial.print(" ");
    Serial.println(datos_E[5]);
```

```
    delay(10);  // Para dar tiempo al emisor
  }
}

//Funciones de los encoders
void Enc_1()
{
  Con_1++;
}

void Enc_2()
{
  Con_2++;
}

void ISR_Enc()
{
  noInterrupts();
  //Controlador
  Kp_Rd = 0.1233;
  Ki_Rd = 0.0378;
  Kd_Rd = -0.0711;
  Kp_Ri = Kp_Rd;
  Ki_Ri = Ki_Rd;
  Kd_Ri = Kd_Rd;
  Sp_Rd = datos_R[0];
  Sp_Ri = datos_R[1];
  if (Sp_Rd > RPMmax)
    Sp_Rd = RPMmax;
  if (Sp_Rd < RPMmin)
```

```
  Sp_Rd = RPMmin;
if (Sp_Ri > RPMmax)
  Sp_Ri = RPMmax;
if (Sp_Ri < RPMmin)
  Sp_Ri = RPMmin;
if(Sp_Rd>=0)
  Sentido_Rd = 0;
if(Sp_Rd<0)
{
  Sp_Rd = abs(Sp_Rd);
  Sentido_Rd = 1;
}
if(Sp_Ri>=0)
  Sentido_Ri = 0;
if(Sp_Ri<0)
{
  Sp_Ri = abs(Sp_Ri);
  Sentido_Ri = 1;
}
Pv_Rd = RPM_Rd;
Pv_Ri = RPM_Ri;
en_Rd = (Sp_Rd-Pv_Rd);
en_Ri = (Sp_Ri-Pv_Ri);
A_p_Rd = Kp_Rd*en_Rd;
A_p_Ri = Kp_Ri*en_Ri;
A_d_Rd = Kd_Rd*(en_Rd-en_1_Rd);
A_d_Ri = Kd_Ri*(en_Ri-en_1_Ri);
I_error_Rd = en_Rd+I_error_1_Rd;
I_error_Ri = en_Ri+I_error_1_Ri;
A_i_Rd = Ki_Rd*I_error_Rd;
```

```
A_i_Ri = Ki_Ri*I_error_Ri;
Uc_Rd = A_p_Rd+A_d_Rd+A_i_Rd;
Uc_Ri = A_p_Ri+A_d_Ri+A_i_Ri;
if (Uc_Rd < 0.0)
  un_Rd = 0.0;
if (Uc_Rd >0 && Uc_Rd <= 100)
  un_Rd = Uc_Rd;
if (Uc_Rd > 100)
  un_Rd = 100.0;
if (Uc_Ri < 0.0)
  un_Ri = 0.0;
if (Uc_Ri >0 && Uc_Ri <= 100)
  un_Ri = Uc_Ri;
if (Uc_Ri > 100)
  un_Ri = 100.0;
U0_Rd = (un_Rd*135)/100;
U0_Ri = (un_Ri*135)/100;
if(Sentido_Rd == 0)
{
  analogWrite(M2A, U0_Rd); //Rd_Ad
  analogWrite(M2B, 0); //Rd_At
}
if(Sentido_Ri == 0)
{
  analogWrite(M1A, U0_Ri); //Ri_Ad
  analogWrite(M1B, 0); //Ri_At
}
if(Sentido_Rd == 1)
{
  analogWrite(M2A, 0); //Rd_Ad
```

```
    analogWrite(M2B, U0_Rd); //Rd_At
  }
  if(Sentido_Ri == 1)
  {
    analogWrite(M1A, 0); //Ri_Ad
    analogWrite(M1B, U0_Ri); //Ri_At
  }
  en_1_Rd = en_Rd;
  en_1_Ri = en_Ri;
  I_error_1_Rd = I_error_Rd;
  I_error_1_Ri = I_error_Ri;

  // Encoder
  RPM_Rd = (Con_1*200000)/t;
  RPM_Ri = (Con_2*200000)/t;
  Con_1 = 0;
  Con_2 = 0;
  interrupts();
}
```

ANEXO 25. PRESUPUESTO GENERAL DEL PROYECTO

Comprador	Elemento	Cantidad	Precio Unitario	Subtotal
Facultad	Arduino mega 2560	7	$ 46.000	$ 322.000
CIIO	Ruedas Pololu 32x7 mm (par)	12	$ 12.600	$ 151.200
	Sensor ultrasonico HC- SR04	20	$ 6.700	$ 134.000
	Encoder Magnetico para micromotor (par)	12	$ 21.000	$ 252.000
	Driver L298N	6	$ 16.000	$ 96.000
	Modulo tranceptor NRF24L01	7	$ 6.700	$ 46.900
	Adaptador tranceptor NRF24L01	7	$ 5.900	$ 41.300
	Micromotor 100:1 con eje extendido 2,2kg-cm	12	$ 43.000	$ 516.000
	Soporte para micromotor plastico (par)	12	$ 12.600	$ 151.200
	Ball caster 3/8 metalica pololu	12	$ 5.000	$ 60.000
	Cámara Logitech C920	1	$ 245.000	$ 245.000
Estudiante	Corte acrílico modelo CAD	7	$ 35.000	$ 245.000
	Impresión 3D modelo CAD	1	$ 47.000	$ 47.000
	Manufactura y envío de la PCB	1	$ 28.950	$ 28.950
	Sensor ultrasonico HC- SR04	12	$ 7.000	$ 84.000
	Baterías 2200mAh 20C 11.1V	6	$ 77.000	$ 462.000
	Extensión USB 2.0 Activa 10m	1	$ 55.900	$ 55.900
	Indicador de tensión de Baterías LiPo	6	$ 9.300	$ 55.800
	Envío indicadores de tensión de Baterías LiPo	1	$ 8.500	$ 8.500
	Protoboard	1	$ 14.000	$ 14.000
	Tercera mano	1	$ 20.000	$ 20.000
	PLA 1.75mm blanco cálido	1	$ 85.010	$ 85.010
	Envío PLA 1.75mm blanco cálido	1	$ 7.700	$ 7.700
	Metro cinta ribbon	1	$ 2.500	$ 2.500
	Metro cable No. 14	5	$ 900	$ 4.500
	Metro cable No. 18	5	$ 500	$ 2.500
	Metro cable protoboard	30	$ 550	$ 16.500
	Pacha de 2 pines	6	$ 1.800	$ 10.800
	Metro termoencogible No. 2	2	$ 700	$ 1.400
	Metro termoencogible No. 3	2	$ 350	$ 700
	Metro termoencogible No. 4	2	$ 700	$ 1.400
	Metro termoencogible No. 5	2	$ 800	$ 1.600
	Interruptor	6	$ 1.200	$ 7.200
	Hilera 40-pines macho	5	$ 800	$ 4.000
	Hilera 40-pines hembra	5	$ 1.300	$ 6.500
	Hilera 40-pines macho con ángulo	3	$ 800	$ 2.400
	Hilera 80-pines macho	3	$ 800	$ 2.400
	Hilera 36-pines hembra	3	$ 1.000	$ 3.000

ANEXO 25. (Continuación)

Comprador	Elemento	Cantidad	Precio Unitario	Subtotal
Estudiante	Bornera azul de 2 pines	12	$ 600	$ 7.200
	Bornera azul de 3 pines	6	$ 800	$ 4.800
	Conector Mol de 4 pines	12	$ 350	$ 4.200
	Conector Mol de 2 pines	12	$ 250	$ 3.000
	Resistencia 330Ω a 1W	6	$ 150	$ 900
	Resistencia 330Ω a 1/4W	24	$ 50	$ 1.200
	Resistencia 2.2kΩ a 1/4W	24	$ 50	$ 1.200
	Fusible 2.5A	10	$ 200	$ 2.000
	Fusible 500mA	10	$ 200	$ 2.000
	Portafusible	30	$ 250	$ 7.500
	Optoacoplador 4N35	30	$ 1.550	$ 46.500
	LED Rojo 5mm	8	$ 150	$ 1.200
	Condensador de tantalio 6.8uF	12	$ 1.500	$ 18.000
	Diodo rectificador 1N4004	12	$ 100	$ 1.200
	Diodo rectificador 1N5408	6	$ 300	$ 1.800
	Diodo rectificador 6A6	6	$ 450	$ 2.700
	Terminal XT60 con cable macho y hembra	1	$ 12.500	$ 12.500
	Terminal XT60 macho y hembra	6	$ 6.900	$ 41.400
	Tornillo y tuerca hexagonal M3 x 16 [mm]	150	$ 50	$ 7.500
	Tornillo 2-56 UNC x 7/16" (paquete 25 und.)	4	$ 3.600	$ 14.400
	Tornillo 2-56 UNC x 1" (paquete 25 und.)	1	$ 7.300	$ 7.300
	Tuerca hexagonal 3/16"	26	$ 60	$ 1.560
	Tuerca hexagonal 2-56 UNC (paquete 25 und.)	5	$ 3.300	$ 16.500
	Tuerca hexagonal de seguridad 3/16"	26	$ 80	$ 2.080
	Tramo tubo aluminio redondo 5/16" x 13 [cm]	1	$ 6.500	$ 6.500
	Metro varilla roscada 3/16" x 16 [cm]	6	$ 2.500	$ 15.000
	Envío tornillos y tuercas 2-56 UNC	1	$ 10.000	$ 10.000
	Lámina MDF 150x104x0.9 cm	1	$ 50.000	$ 50.000
	Balso Lámina 6x50	2	$ 2.750	$ 5.500
	Balso Lámina 6x75	1	$ 4.400	$ 4.400
	Caja alfiler niquelado x 50 gr	2	$ 1.400	$ 2.800
	Balso cuadrado 10x10	14	$ 950	$ 13.300
	Balso cuadrado 5x5	6	$ 550	$ 3.300
	Balso Lámina 3x75	2	$ 3.300	$ 6.600
	Cartón paja blanco 70x10	1	$ 2.200	$ 2.200
	Silicona barra gruesa	8	$ 900	$ 7.200
	Cedro redondo 10	1	$ 3.500	$ 3.500
	Pegante madera 110 gr	1	$ 2.800	$ 2.800
	Canaleta 2m	1	$ 4.150	$ 4.150
			Total	**$ 3.542.750**

ANEXO 26. PRESUPUESTO POR PLATAFORMA ROBÓTICA

Elemento	Cantidad	Precio Unitario	Subtotal
Arduino mega 2560	1	$ 46.000	$ 46.000
Ruedas Pololu 32x7 mm (par)	1	$ 12.600	$ 12.600
Sensor ultrasonico HC- SR04	4	$ 6.700	$ 26.800
Encoder Magnetico para micromotor (par)	1	$ 21.000	$ 21.000
Driver L298N	1	$ 16.000	$ 16.000
Modulo tranceptor NRF24L01	1	$ 6.700	$ 6.700
Adaptador tranceptor NRF24L01	1	$ 5.900	$ 5.900
Micromotor 100:1 con eje extendido 2,2kg-cm	2	$ 43.000	$ 86.000
Soporte para micromotor plastico (par)	1	$ 12.600	$ 12.600
Ball caster 3/8 metalica pololu	2	$ 5.000	$ 10.000
Corte acrílico modelo CAD	1	$ 35.000	$ 35.000
Manufactura y envío de la PCB	0,17	$ 28.950	$ 4.825
Baterías 2200mAh 20C 11.1V	1	$ 77.000	$ 77.000
Indicador de tensión de Baterías LiPo	1	$ 9.300	$ 9.300
Envío indicadores de tensión de Baterías LiPo	0,17	$ 8.500	$ 1.417
Metro cinta ribbon	0,17	$ 2.500	$ 417
Metro cable No. 14	0,83	$ 900	$ 750
Metro cable No. 18	0,83	$ 500	$ 417
Metro cable protoboard	5	$ 550	$ 2.750
Pacha de 2 pines	1	$ 1.800	$ 1.800
Metro termoencogible No. 2	0,33	$ 700	$ 233
Metro termoencogible No. 3	0,33	$ 350	$ 117
Metro termoencogible No. 4	0,33	$ 700	$ 233
Metro termoencogible No. 5	0,33	$ 800	$ 267
Interruptor	1	$ 1.200	$ 1.200
Hilera 40-pines macho	0,83	$ 800	$ 667
Hilera 40-pines hembra	0,83	$ 1.300	$ 1.083
Hilera 40-pines macho con ángulo	0,5	$ 800	$ 400
Hilera 80-pines macho	0,5	$ 800	$ 400
Hilera 36-pines hembra	0,5	$ 1.000	$ 500
Bornera azul de 2 pines	2	$ 600	$ 1.200
Bornera azul de 3 pines	1	$ 800	$ 800
Conector Mol de 4 pines	2	$ 350	$ 700
Conector Mol de 2 pines	2	$ 250	$ 500
Resistencia 330Ω a 1W	1	$ 150	$ 150
Resistencia 330Ω a 1/4W	4	$ 50	$ 200

ANEXO 26. (Continuación)

Elemento	Cantidad	Precio Unitario	Subtotal
Resistencia 2.2kΩ a 1/4W	4	$ 50	$ 200
Fusible 2.5A	1	$ 200	$ 200
Fusible 500mA	1	$ 200	$ 200
Portafusible	4	$ 250	$ 1.000
Optoacoplador 4N35	4	$ 1.550	$ 6.200
LED Rojo 5mm	1	$ 150	$ 150
Condensador de tantalio 6.8uF	2	$ 1.500	$ 3.000
Diodo rectificador 1N4004	2	$ 100	$ 200
Diodo rectificador 1N5408	1	$ 300	$ 300
Diodo rectificador 6A6	1	$ 450	$ 450
Terminal XT60 macho y hembra	1	$ 6.900	$ 6.900
Tornillo y tuerca hexagonal M3 x 16 [mm]	20	$ 50	$ 1.000
Tornillo 2-56 UNC x 7/16" (paquete 25 und.)	0,67	$ 3.600	$ 2.400
Tornillo 2-56 UNC x 1" (paquete 25 und.)	0,17	$ 7.300	$ 1.217
Tuerca hexagonal 3/16"	4	$ 60	$ 240
Tuerca hexagonal 2-56 UNC (paquete 25 und.)	0,83	$ 3.300	$ 2.750
Tuerca hexagonal de seguridad 3/16"	4	$ 80	$ 320
Tramo tubo aluminio redondo 5/16" x 13 [cm]	0,17	$ 6.500	$ 1.083
Metro varilla roscada 3/16" x 16 [cm]	1	$ 2.500	$ 2.500
Envío tornillos y tuercas 2-56 UNC	0,17	$ 10.000	$ 1.667
		Total	**$ 417.902**

ANEXO 27. PARTICIPACIÓN EN EVENTOS

Evento: Muestra robótica

Lugar: Colegio Carl Rogers, Bucaramanga

Fecha: 26 de octubre de 2017

Evento: Muestra robótica

Lugar: Biblioteca Isaías Duarte Cancino, Girón

Fecha: 10 de noviembre de 2017