Universidad Nacional de La Plata

Facultad de Informática

Tesina de grado

Navegación autónoma de Vehículo de 4 ruedas y Cuadricóptero conectados al Cloud

Autores:
Marcos Boggia
Manuel Costanzo

Director:
Ismael Pablo Rodriguez
Co-Director:
Armando De Giusti

Noviembre 2018

Índice general

Índice de figuras

Capítulo 1

Objetivos

1.1. Objetivos generales

La presente tesina tiene como objetivo desarrollar un sistema Multi-Robot haciendo uso del paradigma Cloud Robotics, analizando sus ventajas y desventajas. Este sistema debe coordinar un Auto Robot y un cuadricóptero logrando la navegación autónoma y la colaboración de los mismos para cumplir un objetivo planteado, evitando las limitaciones existentes en el paradigma de la robótica.

Asimismo, para la realización del trabajo es objeto de estudio el desarrollo de los algoritmos para procesamiento de imágenes, detección de obstáculos y planificación de caminos aplicables a robots en tiempo real; como así también el ensamblaje y/o despliegue de los mismos.

1.2. Objetivos específicos

A continuación, se enumeran las diferentes tareas a realizar para desarrollar este trabajo:

- Ensamblar un Auto Robot de 4 ruedas junto con sus sensores integrados.
- Desplegar un dispositivo de vuelo no tripulado (Dron).
- Analizar las ventajas y desventajas de la utilización de Cloud Robotics para sistemas de Multi-Robots.
- Estudiar la evolución de los algoritmos de planificación de caminos aplicables a robots en tiempo real.

- Desarrollar algoritmos que permitan el desplazamiento de los robots a través de sus sensores y actuadores.
- Diseñar e implementar un sistema Multi-Robots que trabaje colaborativamente con el Cloud.
- Desarrollar algoritmos de procesamiento de imágenes para detección de figuras geométricas.
- Desarrollar algoritmos de planificación y transformación de caminos aplicables a robots en tiempo real.
- Desarrollar una interfaz WEB para simulación de caminos y ejecución y visualización en tiempo real del sistema.
- Integrar el sistema completo y gestionar el desplazamiento del Auto Robot desde su posición inicial al destino, evitando colisionar con los obstáculos.

Capítulo 2

Motivacion

Los avances en el paradigma de Cloud Computing han provocado un factor disruptivo de las TI en la industria tecnológica. Según el Instituto Nacional de Estándares y Tecnologías del Departamento de Comercio de los EEUU (NIST), como en varias publicaciones de diversos autores, se ha definido a Cloud Computing como: "un paradigma informático de cómputo distribuido, que proporciona grandes conjuntos de recursos virtuales (como ser hardware, plataformas de desarrollo, almacenamiento y/o aplicaciones), fácilmente accesibles y utilizables por medio de una interfaz de administración web. Estos recursos son proporcionados como servicios (del inglés, "as a service") y pueden ser dinámicamente reconfigurados para adaptarse a una carga de trabajo variable (escalabilidad), logrando una mejor utilización y evitando el sobre o sub dimensionamiento (elasticidad). El acceso a los recursos se realiza bajo la demanda de los usuarios, en base a un modelo de autoservicio"[1]. Este paradigma, brinda al menos tres modelos de despliegue: Cloud Público, Cloud Privado y Cloud Híbrido [2] [3].

Por otro lado, considerando la gran influencia de la robótica en la sociedad y la diversidad de servicios ofrecidos por robots, nos encontramos que los mismos presentan grandes limitaciones en consumo de energía, poder de cómputo, capacidad de almacenamiento, toma de decisiones, tareas cognitivas, entre otras.

En el año 2010, comenzaron a surgir proyectos de investigación (ej: RoboEarth [4]), que integraban las tecnologías de Cloud con los sistemas de robots. Es así, que James Kuffner propone el concepto de Cloud Robotics, basado en combinar las tecnologías de robots con el paradigma de Cloud Computing [5]. La idea de Cloud Robotics, es que permite por medio de aplicaciones tratar los datos de los componentes de hardware del robot (sensores, actuadores, cámaras, memoria, etc.), sin importar las limitaciones de cómputo y almacenamiento de las placas de desarrollo de los robots [6]. En otras palabras, este con-

cepto permite a los robots obtener resultados de tareas de cómputo intensivo, tales como: procesamiento de imágenes, reconocimiento de voz, determinación de rutas, confección de mapas, acciones cognitivas, etc., sin tratamiento local, sino en el Cloud.

Estos paradigmas brindan la capacidad de establecer escenarios para sistemas de Multi-Robots, donde cada robot se integra de un hardware mínimo con conectividad inalámbrica, donde los datos de los sensores y la adquisición de imágenes se procesan en el Cloud, y los actuadores de cada robot realizan las operaciones necesarias [7].

Estas ventajas provistas por el paradigma Cloud Robotics, resuelven una problemática importante presentada en los entornos Multi-Robots. Como se mencionó anteriormente, esta problemática está relacionada con el límite de poder de cómputo y el consumo energético que presentan los robots, ya que si se quisiera implementar un sistema complejo que requiere de la utilización de hardware de altas prestaciones en un entorno Multi-Robots, repercutirá en un abundante consumo de energía, lo que conlleva a un aumento tanto de costos como así también de tamaño. De esta manera, se dificulta y complejiza en gran medida la implementación y mantenimiento de estos sistemas lo cual afecta su viabilidad para uso doméstico y/o industrial.

En la presente tesina, se desea diseñar un sistema Multi-Robots, en donde cada uno de los actores cumpla con las funcionalidades mínimas indispensables para poder desplazarse en un entorno definido. Es deseable que estos robots sean económicos y requieran un bajo consumo energético.

El sistema a resolver en este trabajo propone procesar imágenes en tiempo real proporcionadas por la cámara de un dron, aplicando diferentes filtros para la detección de figuras geométricas que representan objetos del mundo real, distinguiendo entre un robot, que representa a un vehículo de 4 ruedas, diferentes obstáculos y el destino a alcanzar. En base a esta detección, se requiere computar el camino mediante algoritmos de planificación y por último calcular el movimiento que el vehículo debe realizar, con el objetivo de alcanzar el destino evitando los obstáculos presentes en el escenario. Estas acciones deberán ser realizadas en tiempo real, lo que obliga a computar el procesamiento anteriormente mencionado de forma eficiente.

Si se quisiera plantear una solución a este sistema de forma local, se pueden considerar dos opciones: la primera consiste en responsabilizar al Auto Robot de la toma de decisiones, el procesamiento de imágenes y la planificación de caminos, esto puede repercutir, como se expresó anteriormente, en la necesidad de la obtención de un hardware mucho más potente, lo que conlleva la confección de un Auto Robot más costoso y cada vez de mayor tamaño y consumo energético. Si se quisiera configurar al Auto Robot con las funciones mínimas

indispensables, se puede considerar una segunda alternativa, que se fundamenta en la incorporación de una computadora local, donde se comunicará con los robots a través de una red Wi-Fi local.

Las dos opciones presentan inconvenientes en cuanto a la escalabilidad del sistema; si se desea, por ejemplo, complejizar el escenario para detectar objetos del mundo real entre cientos de miles existentes, a través de un procesamiento de imágenes con alta demanda computacional, será necesario reemplazar los componentes hardware del Auto Robot (para la opción 1), o la computadora local (opción 2) por componentes más potentes, con la dificultad y el costo que esto conlleva. Por otro lado, en caso de tener un nivel de procesamiento menor, esta solución no permite ajustar dinámicamente el uso de los recursos mínimos necesarios para ejecutar el sistema (elasticidad), potencialmente generando un mayor gasto de energía.

Si se llevase la responsabilidad de computar todo el procesamiento del sistema al Cloud, no solo se reduce la complejidad de los robots, si no que también subsana los problemas previamente mencionados, debido a que dos de las características fundamentales del concepto de Cloud Computing es la de elasticidad, que se refiere a la capacidad de aumentar o reducir recursos de infraestructura de forma dinámica según sea necesario, y escalabilidad, que permite aumentar el tamaño de carga del trabajo dentro de la infraestructura existente, sin condicionar el rendimiento.

Además, otra motivación importante del uso del Cloud es la utilización de servicios como el de IoT para comunicar los dispositivos o "cosas" presentes en el sistema. Este servicio permite diseñar e implementar una conexión entre los elementos del sistema Multi-Robots de forma sencilla, segura y escalable utilizando el protocolo MQTT, explicado en capítulos posteriores.

En síntesis, resulta de interés analizar el uso del paradigma Cloud Computing para el desarrollo de un sistema en tiempo real sobre un entorno Multi-Robots, debido a los beneficios comentados anteriormente.

Capítulo 3

Introducción

3.1. Paradigmas

3.1.1. Internet Of Things

Es un nuevo paradigma cuya definición deriva de considerar "objetos" o "cosas" conectadas a Internet por medio de redes Wireless, sean estas de tecnología WiFi o 4G LTE, utilizando protocolos de comunicación estándar. Dichas "cosas" pueden considerarse etiquetas de identificación por radio frecuencia (RFID), sensores, actuadores, teléfonos móviles, placas de desarrollos, etc. [8].

3.1.2. Cloud Computing

La computación en la nube es una tecnología que tiene como objetivo ofrecer servicios mediante la conectividad y gran escala de internet.

"La nube", ofrece a los individuos y a las empresas de todos los tamaños un conjunto de recursos de computación con buen mantenimiento, seguro, de fácil acceso y bajo demanda, como servidores, almacenamiento de datos y solución de aplicaciones. Todos estos servicios son requeridos y utilizados a través de una software que brinda una plataforma web [9].

Existen cuatro diferentes categorías de nubes:

- Nube privada: constituida de una sola organización con su propia nube de servidores y software para la utilización sin un punto de acceso público.
- Nube pública: diversos individuos o empresas pueden usar los recursos ofrecidos, a través de internet. El proveedor de la nube es el responsable por el mantenimiento y seguridad.

- Nube híbrida: compuesta por dos o más infraestructuras de nubes distintas que permanecen como entidades únicas, pero que están unidas por una tecnología estandarizada o propietaria.
- Nube comunitaria: diferentes empresas u organizaciones reúnen en conjunto sus recursos en la nube, para cooperar y resolver un problema particular.

3.1.3. Cloud Robotics

La robótica en la nube es un nuevo paradigma que se fundamenta en la combinación de las tecnologías de Cloud Computing y la robótica. La esencia de Cloud Robotics es incorporar el poder de cómputo, capacidad de almacenamiento y otras tecnologías de la nube en pos de beneficiarse de la infraestructura convergente y los servicios compartidos por la robótica.

Este paradigma novedoso, permite la confección de robots más inteligentes y escalables, debido a que el "cerebro" de los robots estará en la nube, por lo que será necesario la incorporación de componentes de hardware mínimo, repercutiendo en el bajo consumo energético y en robots más económicos.

3.2. Amazon

3.2.1. Amazon Web Services

Amazon Web Services (AWS) es una plataforma de cloud computing desarrollada por Amazon, lanzada en 2006. Ésta ofrece una gama servicios incluyendo infraestructura como servicio (IaaS), plataforma como servicio (PaaS) y software como servicio (SaaS), con las características de poder aumentar sus capacidades (escalables) y donde se paga solo lo que se usa (elasticidad). Esta plataforma se apoya en varios data centers desplegados en zonas de disponibilidad distribuidas en distintas regiones del mundo [10].

3.2.2. Amazon Web Services EC2

Amazon Elastic Compute Cloud (EC2) es una plataforma que provee servidores virtuales (instancias) para procesamiento computacional. Este servicio ofrece decenas de tipos de instancias con diferentes capacidades de almacenamiento y procesamiento, orientados a cargas de trabajos o aplicaciones específicas.

Además, AWS dispone de una variedad de herramientas que extienden las funcionalidades de EC2. Por ejemplo, "Auto Scaling tool" ayuda a mantener

un nivel de rendimiento y disponibilidad de los servidores escalando el poder de cómputo de los mismos de forma dinámica [11].

3.2.2.1. Tipos de instancias de EC2

Amazon ofrece una gran variedad de tipos de instancia enfocadas para uso general, con optimización informática, instancias optimizadas para memoria, instancias para informática acelerada o destinadas a almacenamiento.

Para la presente tesina fue utilizada las instancias de propósito general (tipo T2, T3, M5 y M4):

- Instancia T2: proporcionan un nivel base de rendimiento de la CPU. Poseen procesadores Intel Xeon de alta frecuencia, posibilidad de ampliar la CPU, costo económico, equilibrio de recursos informáticos, de memoria y red.

- Instancia T3: son instancias que proporcionan un nivel estándar de rendimiento de la CPU con capacidad de ampliar el uso de la misma en cualquier momento el tiempo que sea necesario. Están basadas en procesadores escalables Intel Xeon de alta frecuencia y en el sistema Nitro de AWS, están diseñadas para aplicaciones con uso moderado de CPU y picos de uso transitorios.

- Instancia M4: basadas en procesadores Intel Xeon E5-2686 v4 (Broadwell) de 2,3 GHz o Intel Xeon E5-2676 v3 (Haswell) de 2,4 GHz, son optimizados para EBS de manera predeterminada sin costos adicionales, dan soporte para redes mejoradas.

- Instancia M5: basados en procesadores Intel Xeon Platinum 8175 de 2,5 GHz con conjunto nuevo de instrucciones de extensiones vectoriales avanzadas de Intel (AXV-512), hasta 25 Gbps de ancho de banda de red con redes mejoradas,a almacenamiento de instancias ofrecido mediante SSD basado en NVMe o EBS que están conectados físicamente al servidor host.

A medida que se avanza en los tipos de instancias, crecen las características como así también el precio. Para este trabajo fue contratada la instancia de tipo T3.small. Esta instancia posee 2 CPU Core, 2GB de RAM y 8GB de almacenamiento SSD.

3.2.3. Amazon Internet Of Things

AWS Internet of Things es una plataforma diseñada para comunicar, administrar y operar grandes conjuntos de dispositivos con conexión a internet a otros servicios AWS en la nube. Ofrece mecanismos de conectividad y seguridad para la transmisión de datos. También, permite que los datos, una vez enviados a la plataforma, puedan ser procesados por las aplicaciones de análisis masivo de datos (Elastic MapReduce), análisis predictivo para aprendizaje automático (Amazon Machine Learning), almacenamiento en bases de datos, etc.

AWS IoT permite conectar fácilmente dispositivos al Cloud y a otros dispositivos. El servicio es compatible con los protocolos: HTTP, WebSockets y MQTT. Amazon ha optado por este último, por ende AWS IoT utiliza la especificación del protocolo MQTT v.3.1.1 con QoS 0 y 1.

Cada dispositivo debe ser registrado como una "cosa" en AWS IoT, para lo cual se emitirá un certificado y un par de llaves privada-pública para el mismo. El certificado y la llave privada, junto con el certificado de la Entidad Certificantes (CA) de Amazon, deberán ser instalados en el dispositivo con el fin que este pueda conectarse al servicio de AWS IoT previa autenticación necesaria [12].

3.3. Protocolos de comunicación

3.3.1. Capa de transporte

Los sockets son mecanismos de comunicación que permite la interacción entre procesos sin la obligatoriedad de pertenecer a una misma máquina. Un socket es un canal de comunicación donde se intercambian datos entre dos programas, haciendo uso de funciones read y write para consumir datos y escribir en el canal respectivamente.

Un socket puede caracterizarse por ser orientado a la conexión (TCP, Transmission Control Protocol), o no orientado a la conexión (UDP, User Datagram Protocol)

3.3.1.1. TCP

Conocido también como protocolo de control de transmisión. Se encarga de crear "conexiones" entre sí para que se genere un flujo de datos. Este proceso garantiza que los datos sean entregados en destino sin errores y en el mismo orden en el que salieron, gracias a algoritmos de detección y recuperación de

errores. Además, cuenta con control de flujo y control de congestión para evitar mal uso en la red [13].

3.3.1.2. UDP

Sus siglas en español: protocolo de datagramas de usuario, es un protocolo no orientado a conexión de la capa de transporte que utiliza el modelo de entrega de "mejor esfuerzo". Este protocolo es muy simple ya que no proporciona detección de errores ni controles relacionado al uso de la red; únicamente posee los campos necesarios para poder transmitir el dato de extremo a extremo. Usualmente más veloz que TCP y usado para conexiones de baja latencia y tolerantes a pérdida de paquetes [14].

3.3.2. Capa de aplicación

3.3.2.1. HTTP

Protocolo de aplicación que se transporta a través de TCP. Es un componente fundamental de la navegación web. Su funcionamiento se basa en peticiones de recursos, como documentos HTML, Javascript, CSS o imágenes, realizadas por un cliente(generalmente un Web Browser) que luego son gestionadas por un servidor web [15].

3.3.2.2. MQTT

Message Queuing Telemetry Transport [16] es un protocolo de comunicación ligero, especialmente diseñado para tolerar conexiones intermitentes y reducir los requisitos de ancho de banda de la red. Desde finales del año 2014, fue presentado como un estándar abierto OASIS [17] y se ha convertido en el protocolo por excelencia para IoT. Soporta comunicación segura con TLS. Maneja tres niveles de calidad de servicio:

- QoS 0: A lo sumo una vez la entrega del mensaje.
- QoS 1: Al menos una vez la entrega del mensaje.
- QoS 2: Exactamente una vez la entrega del mensaje.

Este protocolo, implementa la comunicación de mensajes por medio de la publicación/suscripción sobre un tópico específico (canal de comunicación), como se puede observar en la figura siguiente:

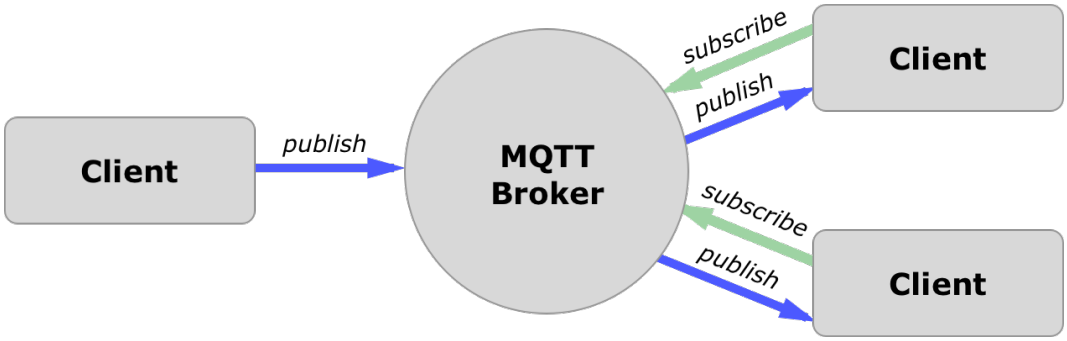

Figura 3.1: Comunicación MQTT

Es utilizado junto a los servicios ofrecidos por AWS IoT con el fin de comunicar a los dispositivos de forma segura, rápida y simple.

3.3.2.3. RTP

"Real Time Transport Protocol" es un protocolo de transporte en tiempo real, que define un formato de paquete estándar para la transmisión de audio y video sobre internet. Este protocolo es utilizado ampliamente en sistemas de comunicación y entretenimiento que involucran medios de transmisión, como la telefonía, aplicaciones de videoconferencias, servicios de televisión WEB, entre otros. Utiliza UDP como protocolo de transporte debido a que reduce tiempos de envío, ya que UDP no maneja sesiones ni mecanismos que garanticen la recepción de paquetes. El protocolo RTP, proporciona un mecanismo de control a través del protocolo RTCP, que se encarga del envío de datos de control entre el emisor y el receptor de una secuencia RTP, aproximadamente cada 5 segundos [18].

3.4. Placas de desarrollo

3.4.1. Raspberry Pi

3.4.1.1. Definición

La placa Raspberry Pi (RPI) es un pequeño ordenador del tamaño de una tarjeta de crédito. Cuenta con un procesador, un chip gráfico y memoria RAM montados sobre una placa base. Esta placa fue lanzada en el 2006 con fines educativos en el ámbito de la informática en todo el mundo. Su pequeño tamaño no es un aspecto condicionante, ya que permite la incorporación de una amplia variedad de sensores, lo que posibilita la utilización de estas placas

para diversos fines, como educación, domótica y automatización, sistemas de seguridad y seguimiento para vigilancia, realización de streaming de audio y video en tiempo real mediante la combinación con cámaras, tecnologías IoT para agricultura, seguridad informática, entre otros [19].

3.4.1.2. Familias y modelos

RPI tiene 2 diferentes familias: RPI original y RPI Zero. La diferencia radica en que la segunda, es una versión reducida de la original, donde se limita la potencia, funcionalidad y tamaño, repercutiendo también en un menor costo[20].

A continuacion, se describe la familia de las RPI originales:

- Raspberry Pi 1: es el primer modelo, cuenta con un procesador Broadcom BCM2835 de un solo núcleo y a 700Mhz, cuenta con 256 MB de RAM y una GPU Broadcom VideoCore IV. El sistema operativo debe instalarse en una SD y posee 1 puerto USB, sin conectividad vía Ethernet. Posteriormente fue lanzado el modelo Raspberry Pi 1 B, que amplía a 512 GB de RAM, con 2 puertos USB y conectividad Ethernet. Ambos modelos disponen de 8 x GPIO, SPI, I2C y UART.

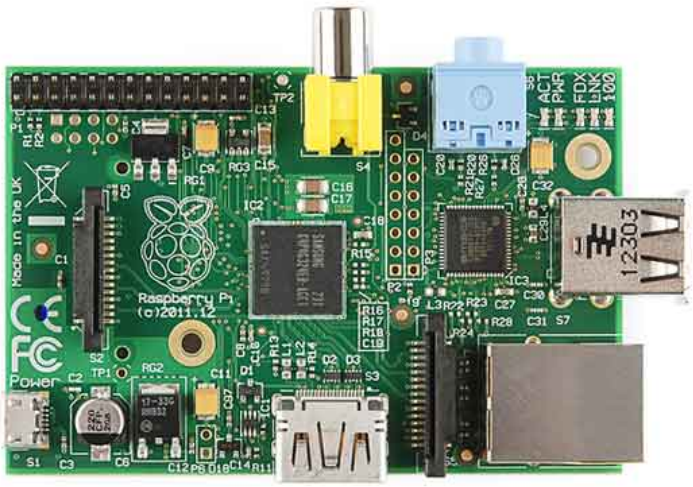

Figura 3.2: Placa Raspberry Pi 1

- Raspberry Pi 2: en 2011 es lanzada la segunda versión de la familia RPI, tiene la misma GPU que la primera versión pero cambia el procesador por un modelo mejorado, el BCM2836, que cuenta con 4 cores a 900 Mhz. Tiene una memoria SDRAM de 1 GB. Cuenta además con 4 puertos

USB, puerto Ethernet 10/100 Mb. El número de pines GPIO se amplía a 17, manteniendo las funciones SPI, I2C y UART.

Figura 3.3: Placa Raspberry Pi 2

- Raspberry Pi 3 B: el modelo RPI 3 B fue lanzado para mejorar la conectividad. Este modelo incluye conexión vía Bluetooth 4.1 y Wifi 802.11n. Fue mejorada la potencia incorporando un SoC Broadcom BCM2837, y un procesador ARMv8 de cuatro núcleos a 1.2GHz de 64 bits. El chip gráfico es el mismo (VideoCore IV), posee 1GB de SDRAM, puerto Ethernet 10/100 Mb, 4 puertos USB y 17 GPIO con funciones SPI, I2C y UART.

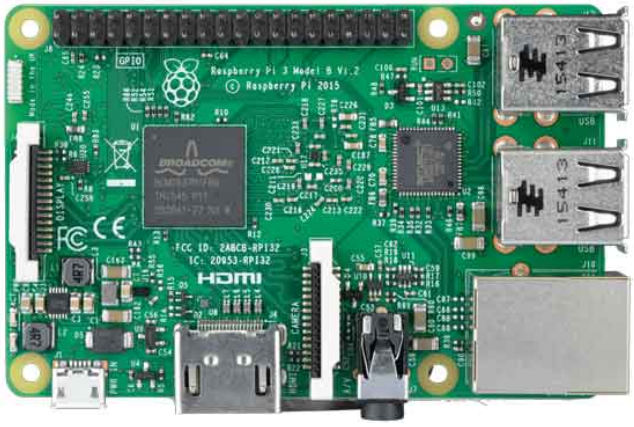

Figura 3.4: Placa Raspberry Pi 3 B

- Raspberry Pi 3 B+: último modelo lanzado en Marzo del 2018. Presenta un rediseño completo de la placa, manteniendo el mismo tamaño y la misma ubicación de los elementos que la versión anterior. Incluye un procesador más potente a 1.4 Ghz, incorporando también un nuevo modelo de Bluetooth 4.2, BLE, Wi-Fi a doble banda 2.4 Ghz y 5 Ghz, y además la tarjeta de red puede alcanzar los 300 Mbps al funcionar sobre USB 2.0

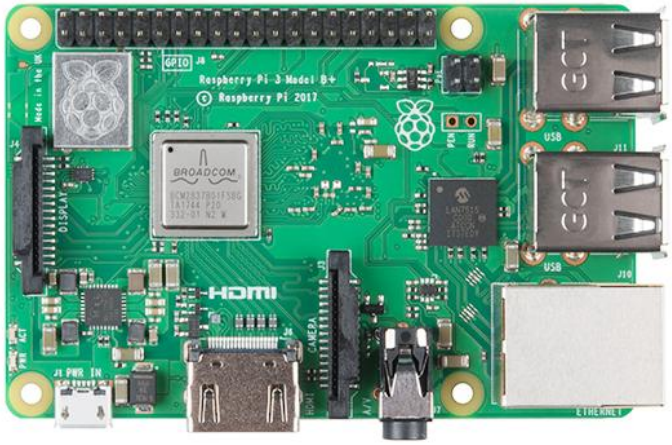

Figura 3.5: Placa Raspberry Pi 3 B+

3.4.2. Arduino

3.4.2.1. Definición

Es una plataforma open-source generalmente usada para proyectos de electrónica e informática. Consiste en una placa microcontroladora programable de bajo costo y su software (IDE) para programar la misma usando el lenguaje de C++. Cuenta con un procesador de bajas prestaciones, una cantidad pequeña de memoria SRAM y memoria no volátil suficiente para almacenar código compilado a través de su IDE. Además, permite conectar sensores y actuadores a una serie de pines analógicos y digitales para que puedan ser controlados según se programe. Se alimenta por USB o por "jack" usando entre 5 y 12 voltios.

Esta plataforma cuenta con varios modelos de placas microcontroladoras. La más conocida y usada es la Arduino Uno, la que cuenta con dos versiones R2 y R3.

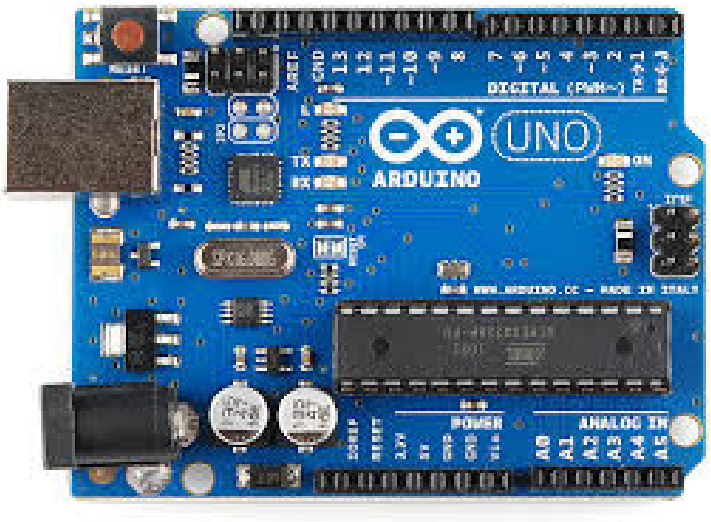

Figura 3.6: Placa Arduino UNO R3

3.5. Robótica

3.5.1. Definición

La robótica es una ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del uso de inteligencia. La robótica tiene como objetivo diseñar y construir sistemas capaces de realizar tareas propias del ser humano. Esta ciencia combina múltiples disciplinas, entre las que se pueden nombrar la informática, la electrónica, la mecánica y la ingeniería.

3.5.2. Robots Móviles

En los últimos años la robótica móvil ha evolucionado en gran medida, como consecuencia de los recientes avances tecnológicos. Los robots de hoy en día generalmente son de costos elevados y cumplen funciones específicas; el desafío actual se centra en la confección de robots más generales y económicos, con el fin de que sean accesibles para todas las personas que desean resolver algún problema en sus tareas cotidianas, o simplemente necesiten automatizar una acción.

Los robots móviles están integrados necesariamente por un sistema operativo, una plataforma de desarrollo y aplicaciones concretas, que permitan generar la lógica necesaria para gestionar de forma autónoma sus acciones y

comportamiento. La estructura de un robot móvil está conformada por diferentes subestructuras:

- Estructura mecánica: estructura con ruedas, patas y orugas
- Actuadores: motores, luces, brazos, ruedas o cualquier elemento que permita la interacción con el entorno.
- Sensores: sonar, láser, cámaras o cualquier sensor que permita la obtención de información del entorno.
- Inteligencia: métodos y algoritmos que basado en los datos de los sensores, permite modificar las acciones o el comportamiento del robot.

3.5.3. Vehículo autónomo

Existen distintos tipos de vehículos los cuales pueden ser clasificados dependiendo de múltiples características. Por ejemplo, dependiendo de los terrenos en los cuales tiene capacidad de transitar, se pueden dividir en "on-road" para terrenos pavimentados o lisos y "off-road" para terrenos de tierra, roca o con imperfecciones. Por otro lugar, hay vehículos que no requieren de un suelo para poder movilizarse. Estos pueden ser aéreos como aviones, naves, drones; de agua como barcos, submarinos entre otros; y por ultimo anfibios, los cuales pueden transitar sobre un suelo o por la superficie del agua.

Una de las categorías que resulta de interés para este trabajo son los vehículos autónomos. Estos son aquellos que tiene capacidad de realizar tareas por sus propios medios, desprendiendo diferentes niveles de autonomía:

- Nivel 0: provee un nivel nulo de autonomía.
- Nivel 1: realiza tareas criticas especificas de forma autónoma.
- Nivel 2: coordina dos o mas tareas realizadas de forma autónoma.
- Nivel 3: implica un nivel de conducción limitado.
- Nivel 4: el vehículo controla de forma total y autónoma todas las operaciones.

El nivel de autonomía de estos vehículos permite mejorar la seguridad, fiabilidad y eficiencia mediante el manejo de ciertas operaciones sin requerir de la intervención humana. Con respecto a la industria automotriz, actualmente a nivel mundial se estiman 1.20 millones de muertes por año a causa de accidentes de transito donde mas de un 90 % de las mismas son causadas por

errores humanos. Estos números podrían ser reducidos gracias a los vehículos autónomos, los cuales no sufren de distracciones como los humanos.

Por otro lado, existe un gran interés actual sobre estos tipos de vehículos: una encuesta realizada en Estados Unidos, Reino Unido y Australia indica que mas de un 50 % de las personas tiene una opinión positiva sobre los mismos. Otra encuesta realizada en los mismos países, demuestra mas de un 30 % de personas con gran preocupación de conducir un vehículo de nivel 4 de autonomía. A pesar de los primeros datos obtenidos de la primera encuesta, todavía se tiene una gran preocupación por su desempeño con respecto a la seguridad y fiabilidad.

El estudio de vehículos autónomos es una área activa de investigación y desarrollo la cual posee múltiples aplicaciones. Las primeras implementaciones de estos sistemas realizan las tareas de forma independiente, limitando sus capacidades o poder de computo al hardware utilizado en su confección. Para poder implementar vehículos autónomos que logren ser manejados de forma autónoma, es de gran ayuda sobrepasar estas limitaciones y utilizar paradigmas que permitan realizar tareas o cálculos de forma remota o coordinar con otros pares, posiblemente a través del Internet. Estos paradigmas son los mencionados anteriormente: Cloud Robotics y IoT[21].

3.5.4. Dispositivos de vuelo

3.5.4.1. Definición

Un Dron puede definirse como un vehículo aéreo no tripulado, controlado mediante un sistema de comunicación, vía satélite, radio control, Bluetooth y redes WiFi, cuyo movimiento es controlado por una emisora o estación de control que dirige la aceleración o desaceleración de los motores/hélices, los cuales proporcionan sustentación vertical y rigen el movimiento según las preferencias del usuario [22].

El origen de los dispositivos de vuelo se identifica con el ámbito militar, siendo extremadamente útil para misiones de espionaje, y de reconocimiento y observación desde el aire.

Los avances científicos y técnicos han permitido en los últimos años masificación de los Drones y la utilización de estos por parte de civiles, como consecuencia de la reducción de costes de estos vehículos aéreos.

En la actualidad ha aumentado considerablemente en el mercado el desarrollo de los VANT (vehículos aéreos no tripulados).

Existe una gran variedad de diseño de los VANT, de diferentes formas, tamaños, configuraciones y características.

Los Drones cuentan con un sistema de control de vuelo que les permite

realizar los desplazamientos y maniobras necesarias para dirigirse y mantenerse en el aire. Estas acciones son comandadas por una persona o por un plan de vuelo establecido, lo que posibilita realizar vuelos en forma autónoma.

3.5.4.2. Clasificación

Estos dispositivos pueden clasificarse en base a una gran variedad de aspectos: por tamaño, tipo de motor, origen del diseño, forma de despegue, entre otras. En particular, se clasificará teniendo en cuenta dos aspectos:

1. Dependiendo del origen de uso
    - Militar: conocidos también como drones de combate. Utilizados para misiones militares.
    - Civil: sin aplicación militar. Para uso de entretenimiento, filmografía, cartografía, etc.
2. Dependiendo del número de hélices
    - Tricópteros de 3 hélices.
    - Quadricópteros de 4 hélices.
    - Hexacópteros de 6 hélices.
    - Octocópteros de 8 hélices.

Figura 3.7: Tipos de drones

3.5.4.3. Componentes

- Chasis: Es el esqueleto del Dron, en el que van a ir incluidos el resto de los componentes. Es la estructura central, que determinará el tamaño y el resto de las características del Dron. Generalmente compuestos por materiales de fibra de carbono, fibra de vidrio y plástico, para reducir el peso y aumentar la resistencia.

Figura 3.8: Chasis de dron.

- Motores: mantienen al dispositivo en el aire mediante el giro de las hélices. Generalmente son motores eléctricos, mediante una bobina por la que circula la corriente eléctrica.

Figura 3.9: Motor de dron.

- Hélices: permiten elevar al dron en el aire, impulsadas mediante la activación de los motores. Pueden ser de dos o de tres aspas, siendo la primera opción la más común en el mercado. Están compuestas generalmente por fibra de carbono, plástico o nylon.

Figura 3.10: Hélices de dron.

- Baterías: existen una gran variedad de baterías que un dron puede utilizar, entre las que se encuentran:

  - Ni-Cd: no toleran carga rápido.
  - Ni-Mh: tienen poca vida útil y efecto de memoria.
  - Ion-Litio: tienen casi el doble de capacidad de las anteriores, se caracterizan por realizar la carga en forma veloz, no poseen efecto memoria y se descargan menos.
  - Li-Po, es el tipo de batería más moderno y utilizado, optimiza el espacio de fuselaje dedicado a las baterías.

Figura 3.11: Baterías de dron.

- Placa controladora de vuelo:

  - Giroscopio: mide los grados de alabeo y cabeceo respecto al horizonte.
  - Sensores de altitud y altura: presenta información tanto de la distancia al suelo (altura) como también de la distancia al nivel del mar (altitud).
  - Sensor de variación de altura: información obtenida debido al ascenso y descenso del dron mediante variaciones de presión.
  - Sensor de velocidad: mide la presión que ejerce el aire contra la parte frontal del dron al desplazarse.

- Sensores de posición: suelen utilizar GPS, además de la utilización del giroscopio y acelerómetros.

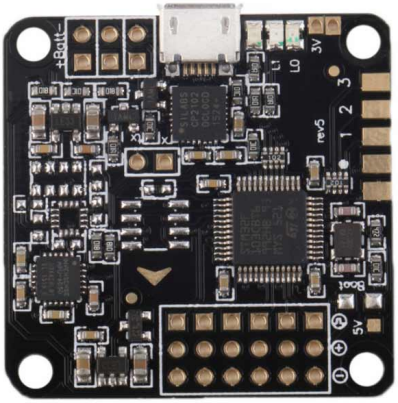

Figura 3.12: Placa controladora de dron.

- Estación de control:
  - Emisor/receptor de señal: comunica con el dron para el control del vuelo y recibe información de los sensores mediante señales de radio a través de una antena.
  - Elementos de control o mandos: permiten pilotar el dispositivo dando control sobre los motores y el resto de sistemas que involucran en el vuelo.
  - Elementos de visualización y gestión de datos: procesa datos de posicionamiento y telemetría mostrando también la información necesaria para el vuelo. El dispositivo de vuelo tiene un FPV integrado que permite al receptor de video de la estación de control visualizar las imágenes de la cámara del dron. Además, por medio de un software se puede lograr la ejecución del piloto automático, controlar la cámara, etc.

Figura 3.13: Estación de comando de dron.

- Cámara: generalmente incluida en el dron. Permiten visualizar a través de capturas o de video en tiempo real lo que estaría observando el dron. Se puede configurar el grado de visión de la misma.

Figura 3.14: Cámara de dron.

3.6. Visión por Computadora

3.6.1. Introducción

La visión por computadora es un disciplina científica que se basa en adquirir, procesar, analizar y comprender las imágenes captadas del mundo real.

Dicho en otras palabras, el proceso de visión por computadora toma una imagen como entrada y devuelve una serie de valores que luego pueden ser tratados por una computadora. El fin de esta interpretación de la imagen es obtener cierta información como por ejemplo, la localización, dimensiones y tipo de diferentes objetos, la presencia de personas, posibles caminos transitables, entre otros [23] [24].

Esta interpretación se logra gracias al trabajo en conjunto con diversas disciplinas como la geometría, estadística, física, entre otras. Es importante destacar la importancia del área de procesamiento de imágenes, la cual ayuda a mejorar las imágenes a interpretar por los algoritmos de visión. Actualmente existen muchas aplicaciones prácticas para la visión por computadora. Alguna de ellas son:

- Robótica móvil y vehículos autónomos: se utilizan cámaras y otros tipos de sensores, con los cuales se obtiene información de objetos, personas y/o escenarios.
- Manufactura: se utiliza la visión como medio para detectar fallos en productos dentro del control de calidad, localizar e identificar piezas, entre otras tareas.
- Imágenes aéreas y satelitales: la visión permite identificar, en este caso, diferentes tipos de cultivos, predecir el clima, etc.
- Análisis e interpretación de imágenes médicas: la visión se aplica para colaborar en la interpretación de imágenes como rayos-X, tomografías, resonancia magnética, ultrasonido, endoscopia, entre otros.

En este trabajo, se utilizará esta disciplina para procesar e interpretar imágenes obtenidas de un escenario con el fin de obtener la información de los objetos presentes en el mismo.

3.6.2. Conceptos fundamentales

Es importante entender ciertas definiciones y conceptos relacionados con el ámbito de la visión digital y procesamiento de imágenes.

3.6.2.1. Definiciones básicas

- Píxel: elemento mínimo de una imagen. Este elemento puede estar definido por una cantidad N de bits. Estos bits representan el color y luminosidad del píxel dependiendo del formato de la imagen.

- Imagen: digitalmente hablando, una imagen se trata como un arreglo bidimensional o matriz de píxeles. Esta imagen cuenta con una resolución, que es definida por la cantidad de píxeles que tiene a lo alto y a lo ancho; dicho de otra forma, la cantidad de columnas y filas que tiene la matriz. Por ejemplo, una imagen podría tener una resolución de 640x480; esto significa 640 columnas y 480 filas en la matriz de píxeles.

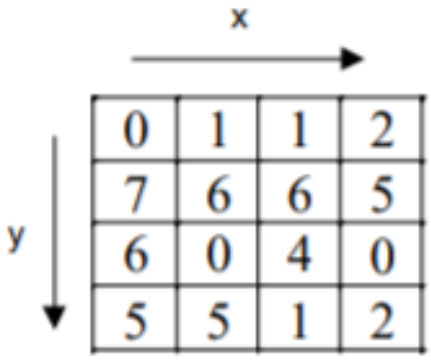

Figura 3.15: Matriz de píxeles.

- Color: está basado en la combinación de tres colores básicos, rojo, azul y verde(RGB), para poder representar cualquier otro color.
- Brillo: indica la iluminación de un área. Tono: indica la similitud de un área al color rojo, amarillo, verde o azul.
- Luminosidad: cantidad de brillo de un área respecto a otra área blanca en la imagen.
- Croma: coloración de un área respecto al brillo de un blanco de referencia.
- Field of View(FOV): campo de visión en grados que detecta una cámara.
- Sensor width: ancho del sensor de una cámara
- Focal length: indica la distancia a la que se va a realizar el enfoque de una cámara.

3.6.2.2. Relaciones entre píxeles

- Vecindad: un píxel p con coordenadas *(x, y)*, tiene 2 vecinos horizontales: *(x+1,y), (x-1,y)*, 2 vecinos verticales: *(x,y-1), (x,y+1)* y 4 vecinos diagonales *(x+1,y+1), (x+1,y-1), (x1,y-1), (x-1,y-1)*.

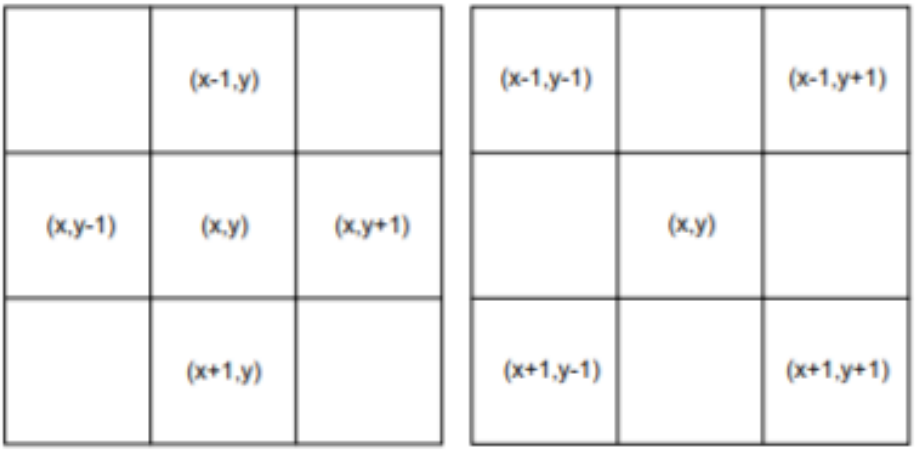

Figura 3.16: A la izquierda vecinos horizontales y verticales, a la derecha vecinos diagonales.

- Conectividad: utilizado para establecer límites de objetos en regiones dentro de una imagen. Dos píxeles están conectados si son vecinos entre sí y además cumplen con algún criterio de similitud en sus niveles de grises.

- Distancia: indica una medición de la separación entre dos puntos (píxeles) dentro de una imagen. Las funciones de distancia más comunes son: distancia euclidiana, distancia Manhattan y distancia de tablero de ajedrez. En las siguientes figuras se puede observar la distancia calculada desde el punto del centro hacia los demás en una matriz de 5x5, utilizando diferentes funciones.

  - Distancia euclidiana: dado dos puntos p y q con coordenadas (x, y) y (s, t) respectivamente, la fórmula está dada por la siguiente ecuación:

$$D(p,q) = \sqrt{(x-s)^2 + (y-t)^2}$$

$$\begin{array}{ccccc} \sqrt{8} & \sqrt{5} & 2 & \sqrt{5} & \sqrt{8} \\ \sqrt{5} & \sqrt{2} & 1 & \sqrt{2} & \sqrt{5} \\ 2 & 1 & 0 & 1 & 2 \\ \sqrt{5} & \sqrt{2} & 1 & \sqrt{2} & \sqrt{5} \\ \sqrt{8} & \sqrt{5} & 2 & \sqrt{5} & \sqrt{8} \end{array}$$

Figura 3.17: Distancia euclidiana.

- Distancia Manhattan: en este caso, se calcula la distancia teniendo en cuenta sólo los vecinos verticales y horizontales. Su ecuación:

$$D(p,q) = |x-s| + |y-t|$$

$$\begin{array}{ccccc} 4 & 3 & 2 & 3 & 4 \\ 3 & 2 & 1 & 2 & 3 \\ 2 & 1 & 0 & 1 & 2 \\ 3 & 2 & 1 & 2 & 3 \\ 4 & 3 & 2 & 3 & 4 \end{array}$$

Figura 3.18: Distancia Manhattan.

- Distancia tablero de ajedrez: similar a la distancia anterior, pero en este caso se toma como la misma distancia a los 8 vecinos del punto. Su ecuación:

$$D(p,q) = max(x-s, y-t)$$

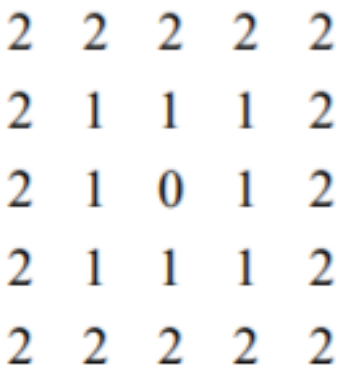

Figura 3.19: Distancia tablero de ajedrez.

3.6.2.3. Ruido en imágenes

Un fenómeno que afecta a las imágenes es el ruido generado por la cámara o el medio de transmisión. Este se manifiesta como píxeles aislados que toman un nivel de gris distinto al de sus vecinos. Existen varios tipos de ruido:

- Gaussiano: pequeñas variaciones en la imagen. Puede ser generado por distintas ganancias en la cámara, interferencias en la transmisión, entre otros. Este ruido modifica el valor de cada píxel sumando un error que se describe como una variable aleatoria gaussiana.
- Impulsional: este ruido es generado por una saturación del sensor, una pérdida de señal en un punto o al trabajar con objetos a altas temperaturas. Los píxeles que se consideran ruido son aquellos con valores muy altos o bajos.
- Multiplicativo: ruido que resulta de la multiplicación de dos señales (imágenes).

Figura 3.20: Distintos tipos de ruidos afectando a una imagen.

3.6.3. Procesamiento de las imágenes

Área utilizada para mejorar o facilitar la obtención de información de la imagen. Este proceso se basa en tomar una imagen como entrada y devolver una o varias imágenes modificadas como salida. Esta modificación o procesamiento se alcanza mediante la aplicación de diferentes técnicas que destacan ciertos atributos de la imagen. Algunos métodos o técnicas logran remover defectos y problemas de movimiento o desenfoque, mejorar contraste o color, agregar "colores falsos" a imágenes monocromáticas, entre otras. Por ejemplo, como se observa en la figura 3.21, se realiza un procesamiento de la imagen para poder identificar de forma más simple el contorno del objeto.

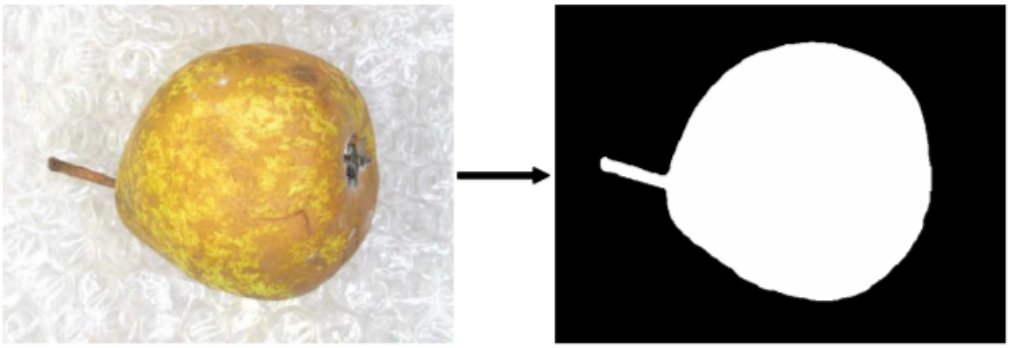

Figura 3.21: Ejemplo del procesamiento de una imagen para contrastar al objeto del fondo.

3.6.3.1. Filtrado

Una de las técnicas principales del área de procesamiento de imágenes es el filtrado. Consiste en aplicar una transformación a una imagen, para obtener otra con ciertos atributos modificados. Esta operación se basa en el concepto de vecindad; en la cual el valor de un píxel se modifica en base al valor de otros píxeles vecinos. Muchas veces se utiliza una máscara la cual se centra sobre el píxel a modificar y utiliza, para calcular su valor, los píxeles que se encuentren dentro de la misma.

Existen diferentes tipos de filtros que realzan o atenúan diferentes atributos de la imagen, algunos de los más importantes:

- Filtros de suavizamiento: su objetivo es eliminar o disminuir el ruido o detalles que no son de interés. Corresponde a un filtro pasa-bajos, los cuales eliminan o reducen las altas frecuencias. Tipos de filtros de suavizamiento:
  - Promedio: se calcula el promedio de los píxeles vecinos
  - Mediana: reemplaza el valor del píxel central por el de la mediana de los valores en la máscara.
  - Gaussiano: aproxima a una distribución gaussiana. Este filtro en general da mejores resultados que los otros y se argumenta que la vista humana hace un filtrado similar.
- Filtros de acentuamiento: la idea de estos filtros es intensificar los detalles y contrastes mientras reduce las bajas frecuencias. En caso de que la

imagen tenga ruido, este se intensifica, por lo tanto se recomienda eliminar el ruido antes de utilizar estos filtros. También son conocidos como filtros pasa-alta.

3.6.3.2. Mejoramiento de imagen para detección de contornos

Para detectar bordes y contornos, se pueden combinar diferentes técnicas de filtrado con distintos parámetros con el fin de obtener una imagen simple que ayude a detectar objetos, bordes, etc.

Figura 3.22: Imagen modificada para facilitar la detección de objetos y bordes.

Un algoritmo conocido para realizar esta tarea es el algoritmo de Canny o "Canny Edge". Este consiste en varias etapas: filtrar la imagen usando un filtro Gaussiano con el fin de suavizar y reducir ruido, luego se calcula el gradiente de la imagen y se aplica una supresión de no máximos [25].

Otra técnica que se puede utilizar para obtener una imagen que dibuje solo los contornos es suavizando la imagen y utilizando un umbral. Básicamente, a cada píxel se le asigna un valor de blanco o negro dependiendo de un umbral. Este umbral puede ser relativo a los píxeles cercanos o global [26].

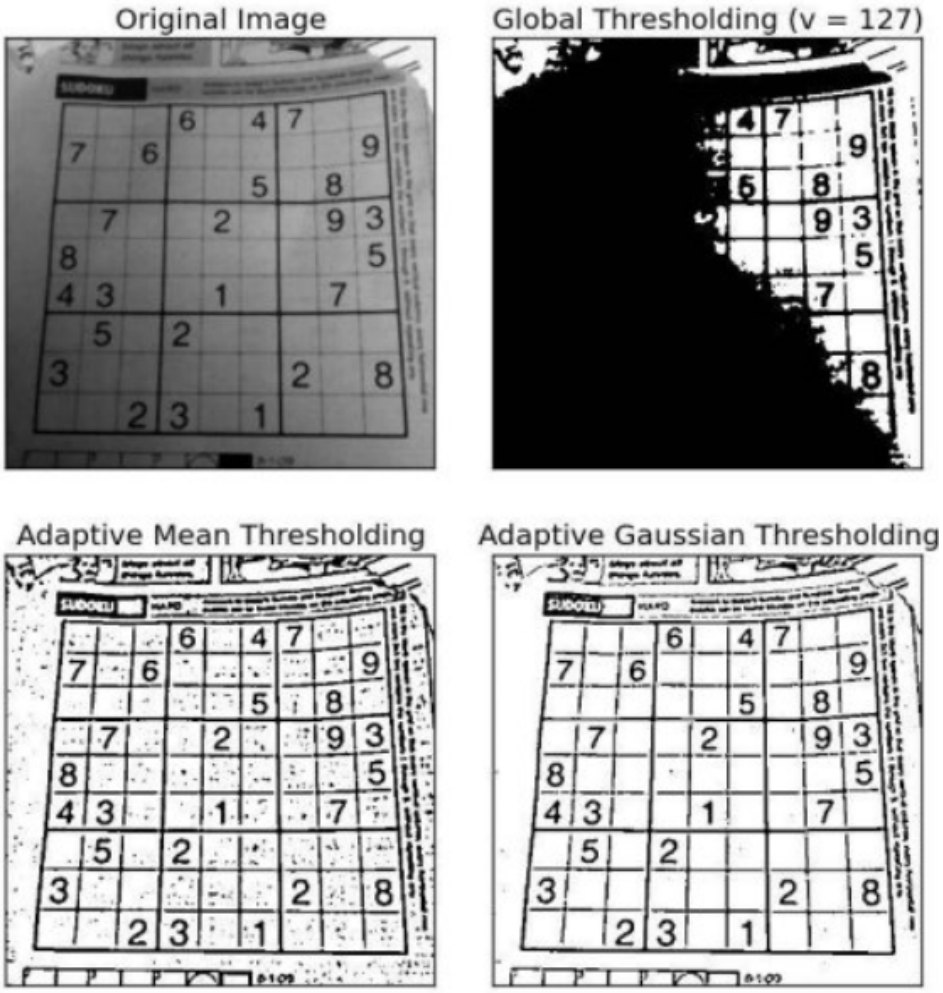

Figura 3.23: Resultado de diferentes técnicas de umbral.

3.7. Planificación de caminos

3.7.1. Introducción

En los últimos tiempos, la robótica ha sido una de las disciplinas que mayor desarrollo ha experimentado. En la actualidad, es común hallar robots que operan en diferentes ámbitos de aplicación, como producción en cadena, exploración de entornos inaccesibles por el ser humano, trabajos de elevada precisión, etc. Aunque estos robots son de gran ayuda, es indispensable la intervención humana para dirigir la tarea. El esfuerzo actual radica en propiciar a los robots de la autonomía necesaria para que puedan realizar sus movimien-

tos sin la interferencia de una persona. En el caso de la robótica móvil, tema fundamental de esta tesina, los robots deben desplazarse en su entorno en forma autónoma. Una de las metas es percibir este entorno mediante sensores que permitan identificar el espacio transitable, es decir, aquel camino que no colisiona con un obstáculo, y moverse en el mismo en forma precisa, utilizando datos de referencia para ajustar su comportamiento en todo momento, en forma autónoma [27].

Existen múltiples maneras de encontrar el camino correcto desde una configuración inicial hasta un destino:

El primer método a describir es el de "roadmap"; estos métodos se basan en la construcción de un grafo que contiene nodos, quienes representan algunas configuraciones físicas del robot, y los arcos son la posibilidad de conexión entre dos nodos. Luego de la construcción del grafo, para determinar el camino que llevará al robot desde su punto de inicio al destino, se deberá conectar estos puntos con el grafo, y hallar una secuencia de nodos dentro de él.

Dentro de la categoría de planificación "roadmap", se encuentra el método de los grafos de visibilidad, que consiste en generar el grafo identificando sus nodos con los vértices de los polígonos que representan los obstáculos; posteriormente se agregan los puntos de inicio y de destino. Por consiguiente, la solución que propone consiste en determinar una sucesión de vértices conexos dentro del grafo, en la que primero tiene lugar al punto inicial y por último el punto de la meta. Recorriendo el grafo, el robot será capaz de alcanzar cualquier vértice [28].

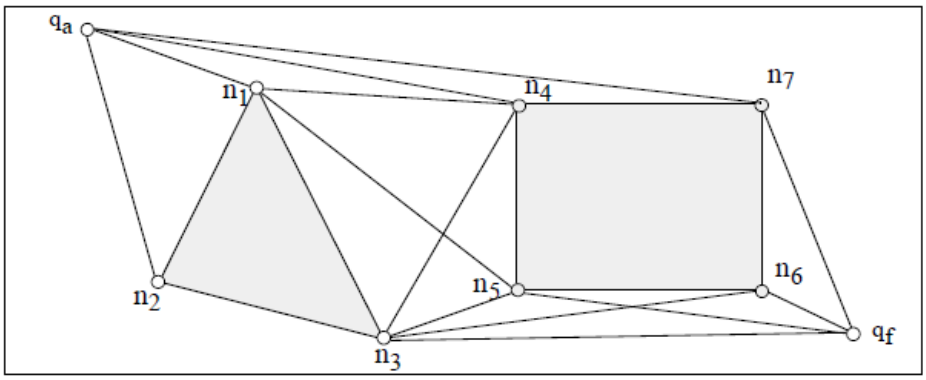

Figura 3.24: Método "roadmap", grafo de visibilidad.

Otra alternativa dentro del método "roadmap" es el diagrama de Voronoi, que construye el grafo de conectividad teniendo en cuenta la distancia entre el robot y los diferentes obstáculos. Un diagrama de Voronoi representa los

puntos que equidistan de los límites existentes en el espacio de configuraciones que están libres de colisión [29].

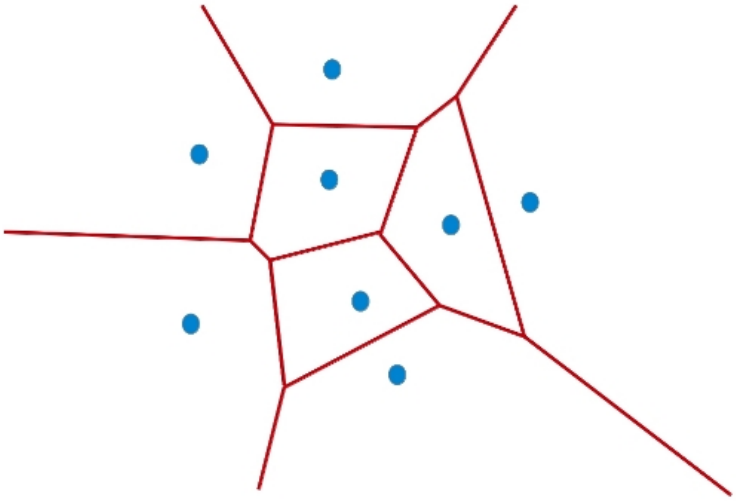

Figura 3.25: Método “roadmap”, de diagrama de Voronoi.

Por otra parte, para la determinación del camino se puede optar por los métodos de descomposición en celdas, que tratan de dividir el espacio de configuraciones libres de colisión en una colección de celdas. Estas celdas tienen la particularidad de que la conexión entre dos puntos en su interior es trivial. Luego de dividir el espacio de colisión en un conjunto de celdas, se prosigue a efectivizar la conectividad a un nivel superior, por ejemplo, mediante la confección de un grafo de adyacencia. En resumen, se le asigna un nodo a cada celda, y dichos nodos serán unidos o no en función de que las celdas que representan sean adyacentes [30].

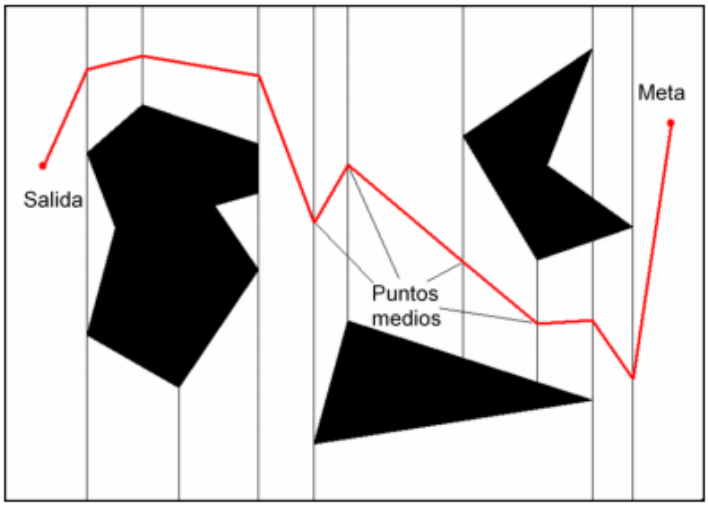

Figura 3.26: Método de descomposición en celdas.

Una tercera opción son las técnicas de campos de potencial, basadas en el diseño de funciones que pueden ser interpretadas como campos de potencial. Esta interpretación se fundamenta en que el robot se comporta como una partícula que se desliza en el campo de potencial definido. La partícula iniciará su marcha en forma espontánea en el punto de partida y seguirá la línea de máxima pendiente, evitando los obstáculos hasta llegar al objetivo [31].

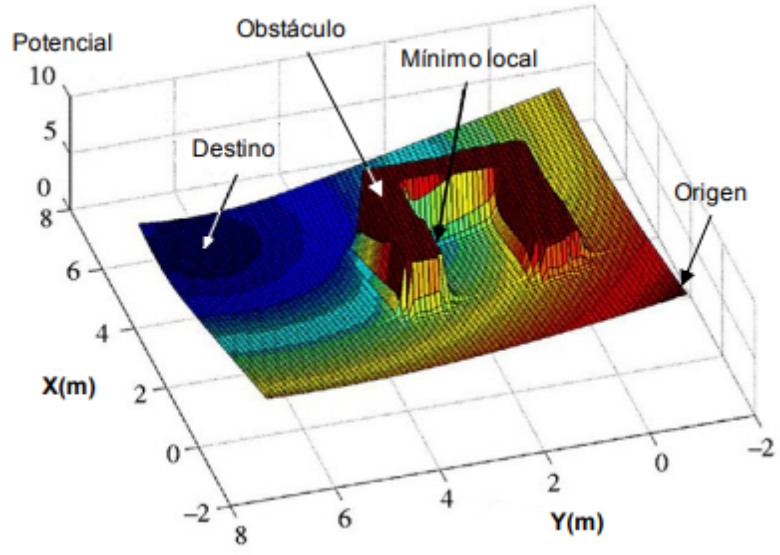

Figura 3.27: Método de campos de potencial.

Por último, se presenta otra categoría de determinación de caminos, implementada y utilizada para esta tesina, denominado métodos de generación aleatoria. Estos métodos surgieron como consecuencia del aumento en la complejidad del entorno, el número de grados de libertad y las restricciones que presentan los robots tanto en cuestiones cinemáticas como dinámicas.

Dichos factores provocaron que los métodos anteriormente mencionados presenten limitaciones en cuanto a la obtención del camino y el excesivo tiempo de cómputo. Entre los métodos de generación aleatoria se pueden destacar en primer medida los mapas probabilísticos ("Probabilistic Roadmaps") que utilizan grafos. Es un proceso iterativo que selecciona un punto aleatorio perteneciente al espacio de configuraciones y se agrega al grafo como un nuevo nodo. Posteriormente, se intenta efectuar la conexión entre el nodo y los demás, y si existe conectividad, se agrega al arco del grafo correspondiente [32].

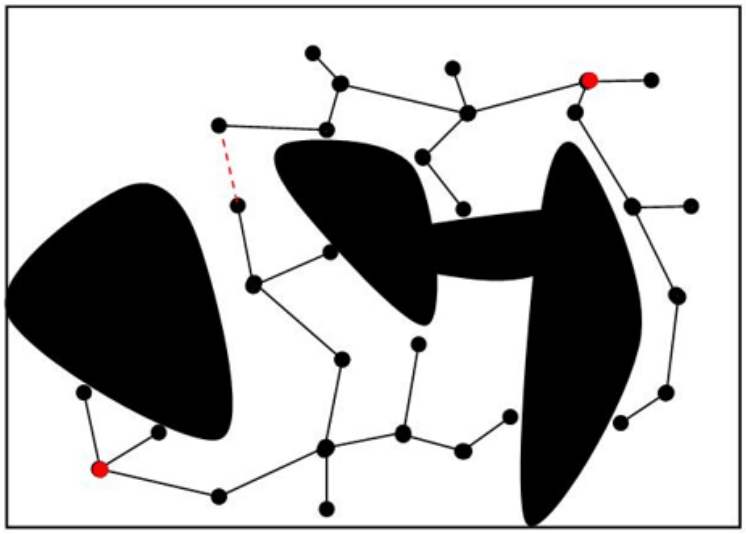

Figura 3.28: Ejemplo de mapa probabilístico.

De los métodos de generación aleatoria más novedosos se encuentra el algoritmo "Rapidly Exploring Random Trees", o RRT. Este algoritmo aleatorio es utilizado como uno de los preferidos planificadores autónomos, existiendo diferentes versiones como "RRT Star" ("RRT*") o "Informed RRT Star" ("Informed RRT*") [33].

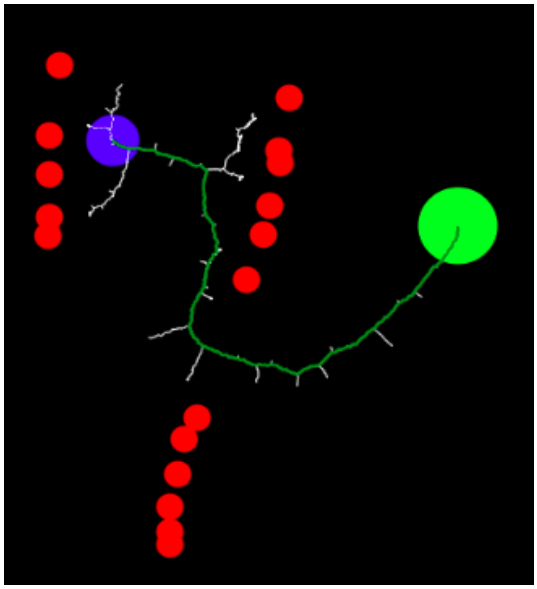

Figura 3.29: Ejemplo de planificación aleatoria.

3.7.2. Rapidly Exploring Random Tree

En 1998 Steven M. LaValle publicó el primer artículo en el que describe uno de los algoritmos de planificación más utilizados en el ámbito de la determinación de trayectorias, denominado Rapidly Exploring Random Tree (RRT). Este algoritmo posee una gran variedad de versiones, cada una presenta soluciones y mejoras, como así también desventajas del algoritmo RRT clásico; en este apartado se explicará el funcionamiento del algoritmo de planificación aleatorio RRT básico.

Un algoritmo RRT se basa en la construcción de un árbol de configuraciones que aumenta desde un punto origen hacia un destino preestablecido. Este método tiene como objetivo la construcción de un árbol de exploración, que permita cubrir de manera uniforme todo el espacio de configuraciones libres de colisión.

El algoritmo recibe como entradas el punto de inicio y destino, la cantidad máxima de iteraciones permitidas, y la distancia entre los nodos (es decir, la longitud en píxeles de las ramas del árbol). Como resultado final, se obtendrá un árbol (conjunto de puntos), que representa el camino encontrado [34].

Para comprender la explicación posterior, se define:

- p: es una métrica que puede ser distancia euclídea u otra ponderación de proximidad.

- Qinit: configuración inicial (por ejemplo, las coordenadas del centro del robot y la orientación del mismo)
- Qend: configuración final a alcanzar.
- Qrand: configuración aleatoria que genera el algoritmo dentro del espacio de configuraciones.
- Qnear: configuración más próxima a Qrand, en el sentido que define p, entre las existentes del árbol.
- Qnew: configuración que se va a incorporar al árbol.

Se ejecuta un algoritmo iterativo, donde en cada iteración busca un punto aleatorio en el espacio libre de obstáculos, y su nodo más cercano del árbol (*Qnear*). Obtenidos estos dos puntos, se calcula una línea recta que une a ambos y determina un tercer punto (*Qnew*) a una distancia Δq de *Qnear*, en dirección a *Qrand*. Si entre *Qnear* y *Qnew* no existen colisiones con los obstáculos, se introduce *Qnew* al árbol, asignando como padre *Qnear*.

El nodo padre es aquel que va unido en dirección hacia *Qinit*. *Qinit* será el único nodo del árbol que no posee padre, todos los demás tendrán exactamente un solo padre, pero pueden contener más de un hijo.

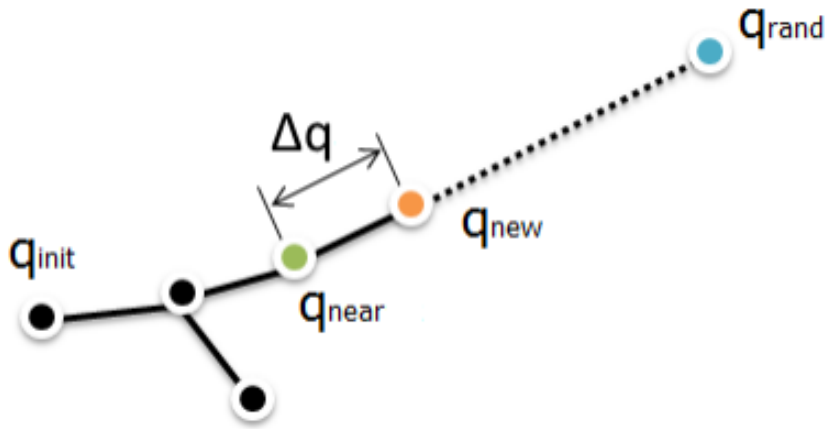

Figura 3.30: Determinación del nuevo nodo en algoritmo RRT.

El algoritmo RRT repetirá estas acciones hasta encontrar un nodo *Qnew* a una distancia mínima de la configuración *Qend*. Esta distancia mínima puede ser una constante definida como propia del algoritmo o puede formar parte de los parámetros iniciales del mismo. Si apareciese este evento, se añade *Qend* al árbol y finaliza la ejecución del algoritmo.

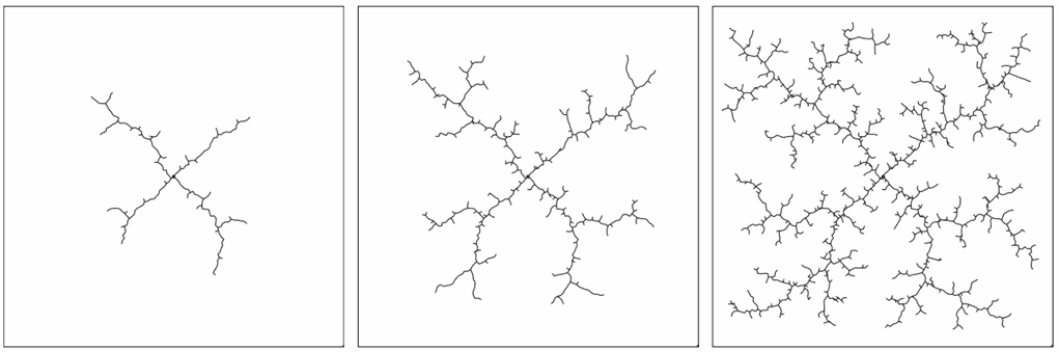

Figura 3.31: Evolución del árbol de configuraciones RRT.

3.7.3. RRT*

Este algoritmo denominado RRT Star (o RRT*), presenta una mejora al RRT básico, propiciando la determinación de un camino en forma óptima. El algoritmo del apartado anterior, finaliza cuando se determina que el nuevo nodo a agregar se encuentra cercano al nodo final. Por el contrario, el algoritmo RRT* una vez encontrado el camino, seguirá iterando hasta encontrar aquel árbol o ruta óptima. Una planificación óptima es aquella que considera el coste de la trayectoria y trabaja para minimizarlo, es decir, son algoritmos que pretenden reducir la distancia o el tiempo que le puede llevar al robot a transitar el camino. Si bien se presentan ventajas, debido a la concepción de rutas más cortas, computacionalmente suelen ser más costosos, a causa que por definición, necesitará una gran cantidad de iteraciones luego de encontrar la primer ruta, con el fin de mejorar el camino. La justificación de la utilización de estos algoritmos óptimos, radica en que se puede llegar a computar la trayectoria más acotada en un pequeño período de tiempo, por ejemplo paralelizando la solución, lo que permite al robot remediar este tiempo extra ante la realización de un camino más corto. Estos algoritmos no son convenientes en sistemas de tiempo real, siendo útil en ese caso un algoritmo RRT básico [35].

Figura 3.32: Ejemplo RRT*.

3.7.4. Informed RRT*

Este algoritmo fue presentado por primera vez en 2014, basado en el algoritmo RRT*, que propone un método para minimizar su tiempo de convergencia. La idea del algoritmo radica en encontrar el camino inicial utilizando como técnica el RRT* simple y una vez encontrado calcula la elipse equivalente de dicho camino, muestreando solamente puntos dentro de la misma [36].

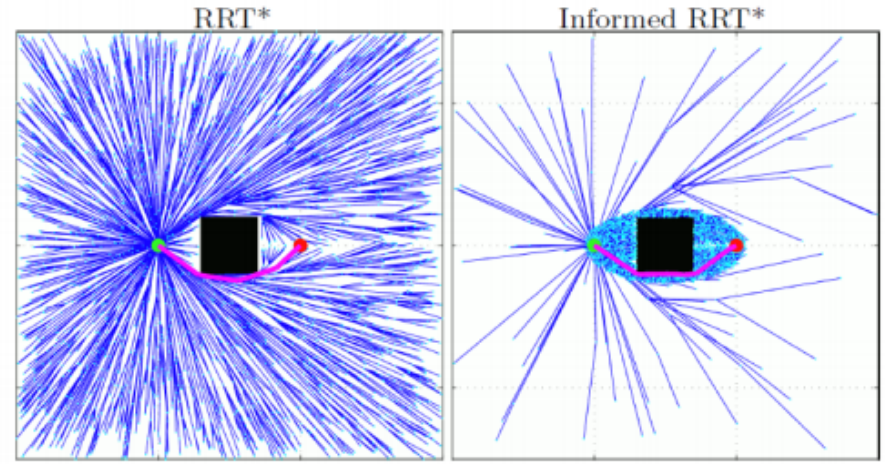

Figura 3.33: Comparación entre RRT* e Informed RRT*.

Una elipse es el lugar geométrico de todos los puntos pertenecientes a un plano tal que la suma de todos sus puntos a dos puntos fijos es constante. En

base a esto, se garantiza que sólo aquellos puntos dentro de la elipse equivalente tienen la capacidad de reducir la longitud total de la trayectoria.

Se refiere a elipse equivalente de la trayectoria a aquella elipse que contiene a la trayectoria, ajustándose al máximo a la misma.

La elipse equivalente contendrá diferentes parámetros, como su centroide, su distancia mínima entre el punto inicial y destino, la orientación de la elipse respecto a los ejes y su matriz de orientación.

Luego del cálculo de la elipse equivalente, se muestran puntos sólo dentro de la misma. Este muestreo consiste primeramente en calcular un punto al azar en polares centrado en (0,0), de forma tal que su radio esté en el rango [0,1] y su ángulo entre [0, 2*π]. Posteriormente, dicho punto en polares se deforma según las dimensiones de la elipse equivalente. Como último paso, el punto se gira multiplicando por la matriz de rotación y es desplazado hacia el centroide. Como resultado se tiene un punto muestreado al azar incorporado en la elipse equivalente, sólo resta verificar que dicho punto no colisiona con ningún obstáculo, si esto sucede se debería comenzar nuevamente.

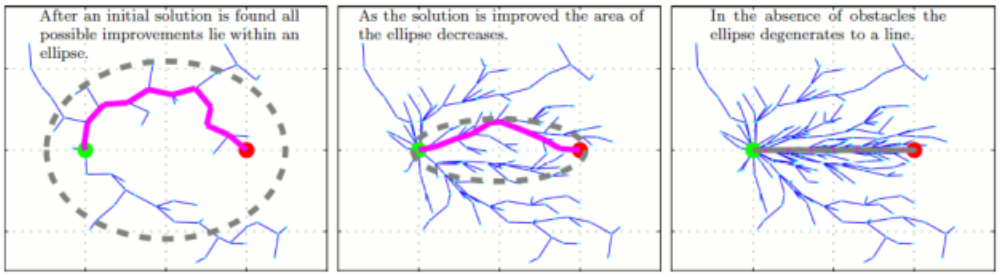

Figura 3.34: Elipse equivalente.

3.8. Lenguajes y tecnologías

En esta sección serán presentadas los diferentes lenguajes, tecnologías y herramientas utilizadas para la realización de esta tesina.

3.8.1. JavaScript

JavaScript es un lenguaje de programación orientado a objetos, basado en prototipos, dinámico y débilmente tipado. Generalmente utilizado en el ámbito WEB, principalmente del lado del cliente, permitiendo obtener mejoras en cuanto al diseño de las interfaces y funcionalidad de páginas WEB. Además, puede ser utilizado del lado del servidor (Server-side JavaScript) [37].

3.8.2. NodeJS

Es un entorno JavaScript del lado del servidor basado en eventos. Es una librería y entorno de ejecución de entrada/salida asíncrona, ejecutada sobre el intérprete de JavaScript. NodeJS permite generar un sistema escalable y consistente que soporta conexiones intermitentes. Propone un modelo con un único hilo de ejecución, utilizando E/S asíncronas que pueden ejecutarse en forma concurrente hasta cientos de miles sin condicionar el rendimiento general. Este diseño es eficaz en aplicaciones altamente concurrentes [38].

3.8.3. Python

Python es un lenguaje de programación, con la filosofía de hacer hincapié en favorecer el código a través de la sintaxis. Es un lenguaje multiparadigma, debido a que soporta programación imperativa, programación orientada a objetos y programación funcional. Es un lenguaje interpretado, utiliza tipado dinámico y es multiplataforma [39].

3.8.4. Lenguaje C

C es un lenguaje de programación orientado a la implementación de sistemas operativos en gran medida, pero también es utilizado para el desarrollo de aplicaciones. Es un lenguaje popular debido a la eficiencia de su código. Este lenguaje tiene tipos de datos estáticos, es débilmente tipificado y proporciona estructuras que permiten crear código a bajo nivel, por lo que se dice que es un lenguaje de mediano nivel [40].

3.8.5. Node-RED

Es una herramienta de programación visual desarrollada en NodeJS, que permite programar algoritmos basados en flujos, reduciendo la cantidad de código escrito. Provee de un editor de flujo basado en navegador WEB y una amplia paleta de nodos con diversas funcionalidades que permiten, por ejemplo, ejecutar código Python o comunicarse con una placa controladora de motores [41].

3.8.6. OpenCV

OpenCV es una biblioteca o librería de visión artificial, es multiplataforma y contiene más de 500 funciones que abarcan una gran rama de áreas en el

proceso de visión, como calibración de cámaras, visión estereoscópica, robótica, reconocimiento de objetos, etc [42].

3.8.7. FFMPEG

Es una colección de software libre que permite grabar, convertir y hacer streaming de audio y video. Permite además, codificar y decodificar audio y video en diferentes formatos [43].

3.8.8. Parrot SDK

Con el fin de controlar el Dron de forma autónoma, fue utilizado el SDK que provee la empresa Parrot. Un kit de desarrollo software (SDK), es un conjunto de herramientas que apoyan y ayudan a la programación de aplicaciones para un entorno tecnológico particular. Haciendo uso de este SDK, fue posible conectarse con el Dron y pilotarlo programáticamente, a través del entorno Node-RED, utilizando el lenguaje C y JavaScript [44].

3.8.9. Flask

Es un micro framework web para el lenguaje Python con licencia BSD. Tiene como objetivo facilitar el desarrollo de páginas web estáticas sin proveer acceso a base de datos ni validaciones de entrada [45].

Capítulo 4

Aporte de la tesina

Este trabajo desarrolla un sistema prototipo que aporta soluciones a diferentes problemáticas encontradas relacionadas con el diseño y ensamblaje de robots, planificación de caminos, procesamiento de imágenes, despliegue de drones y sistemas Multi-Robots en tiempo real conectados a la nube. Contribuye a la comunidad científica y de la industria con herramientas basadas en el paradigma Cloud Robotics para la navegación autónoma de vehículos, con el fin de colaborar con la realización de otras aplicaciones que combinen drones y robots.

A continuación, una lista donde se describen, en líneas generales, posibles aportes realizados por la presente tesina:

- Soluciones para ciudades inteligentes, donde uno o más drones sobrevuelan la ciudad y capturan imágenes de las calles y sus vehículos. Modificando el procesamiento de imágenes y detección de objetos, se podrían detectar los caudales de tráfico y en función de la congestión presentes en las calles, se calcula una ruta alternativa óptima a tomar.

- Diseño de una infraestructura de bajo consumo energético que permite manejar un dispositivo de vuelo con captura de video de forma programática a través de internet. Por ejemplo, esto puede ser utilizado en aplicaciones agrícolas para sobrevolar un campo y obtener imágenes del mismo que luego podrían ser procesadas.

- Sistema desplegado en el Cloud capaz de controlar múltiples robots a través de internet de forma segura. Por ejemplo, esto permite, modificando el diseño del Auto Robot, coordinar múltiples instancias del mismo para que trabajen colaborativamente en un depósito realizando alguna tarea de logística.

Capítulo 5

Trabajo experimental

5.1. Introducción

En esta sección se realizarán diferentes demostraciones con el propósito de explicar en forma práctica el objetivo de la presente tesina y el contexto del trabajo realizado. Se presentan diferentes escenarios, cada uno con un nivel de complejidad que incrementa a lo largo de los ejemplos. Se muestra, en cada caso, una aproximación del modo en que debería ser resuelto por las diferentes partes del sistema, es decir, cómo debería trabajar tanto el mecanismo de procesamiento de imágenes, el algoritmo de planificación de caminos y el Auto Robot.

A continuación se detallan aspectos en común a tener en cuenta en todos los experimentos:

- Procesamiento de imágenes: en cada una de las pruebas se deberán detectar las figuras correspondientes al Auto Robot (rectángulo con un círculo dentro), obstáculos (triángulos) y el destino (círculo). En cada prueba varía el número de triángulos y la presencia del vehículo o de la meta (es necesario que tanto el Auto Robot como el destino estén en el entorno y se detecten para poder continuar con la posterior planificación).

- Planificación de caminos: próximo a la detección, deberán ser utilizadas las propiedades de cada uno de los objetos y será necesario encontrar la ruta que conecte el inicio con el destino, evitando los obstáculos. En caso que no sea posible encontrar el camino, se deberá informar la situación. El camino a encontrar debe ser el óptimo, entendiendo como camino óptimo a la trayectoria más corta desde el inicio a la meta.

- Desplazamiento: el Auto Robot deberá desplazarse ajustando su orientación en función de la trayectoria localizada.

5.1.1. Primer experimento

La primer prueba a realizar consiste únicamente en demostrar que ante la ausencia tanto del Auto Robot como del destino, será imposible detectar la trayectoria a realizar, por lo que finalizará la ejecución del sistema. En la figura 5.1 se presenta el escenario propuesto.

Figura 5.1: Experimento N°1: escenario original

Se deberá detectar el Auto robot, no así el destino ni obstáculos (figura 5.2). Como consecuencia de esto el sistema deberá finalizar informando el error.

Figura 5.2: Experimento N°1: resultado esperado de la detección

5.1.2. Segundo experimento

Como fue expresado en la introducción de esta sección, se pretende aumentar la complejidad de las pruebas a medida que avanzan los ejemplos.

En este segundo experimento, como se ve en la figura 5.3, en el escenario se encuentra únicamente el Auto Robot y el destino, sin obstáculos. Además, el vehículo se encuentra orientado directamente hacia la meta, por lo que la trayectoria a localizar deberá ser una recta que conecte ambos objetos.

Figura 5.3: Experimento N°2: escenario original

En primera medida es necesario detectar el Auto Robot (rectángulo de la parte izquierda) y el destino (círculo en la parte derecha), sin localizar obstáculo alguno. A su vez, se requiere percibir que el auto está aproximadamente a 0 grados. La figura 5.4 muestra en forma aproximada el modo en que debería efectuarse la detección.

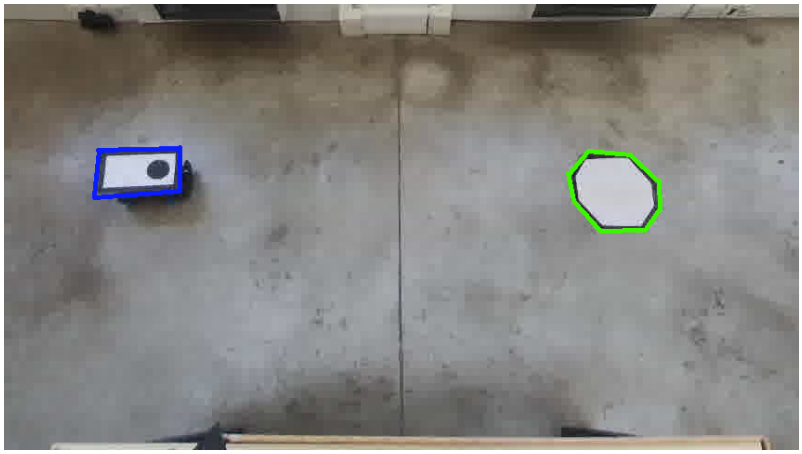

Figura 5.4: Experimento N°2: resultado esperado de la detección

Una vez detectadas las figuras, es necesario computar la ruta. En este caso al no tener obstáculos, la trayectoria a realizar deberá ser una recta desde el centro del Auto Robot hacia el centro del destino, como muestra la figura 5.5.

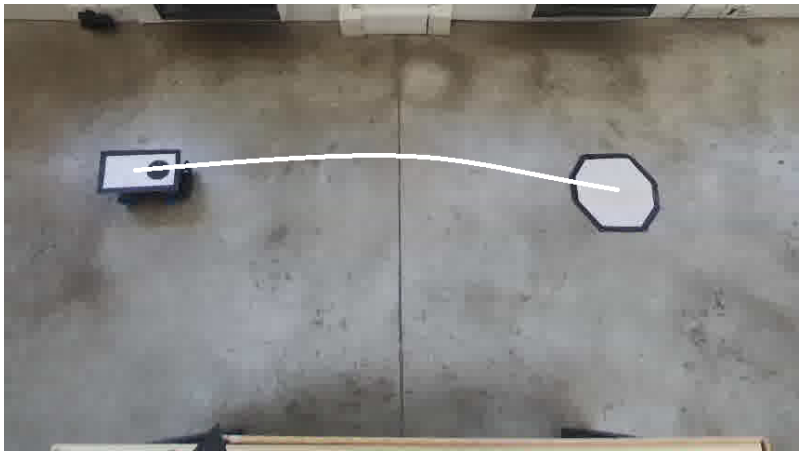

Figura 5.5: Experimento N°2: trayectoria a realizar

Por último, el vehículo deberá desplazarse hasta ingresar en el destino (ejemplo en figura 5.6).

Figura 5.6: Experimento N°2: finalización del trayecto

5.1.3. Tercer experimento

En este experimento se encuentra un obstáculo que interfiere en el desplazamiento del vehículo, por lo que se deberá realizar el esfuerzo necesario para modificar la dirección del Auto Robot, esquivar el obstáculo, y retomar hacia el destino. Como se explicó en la introducción de esta sección, es requisito encontrar la ruta más corta; persiguiendo este objetivo, la ruta óptima en esta prueba (figura 5.7) se encuentra esquivando el obstáculo por el margen izquierdo del escenario.

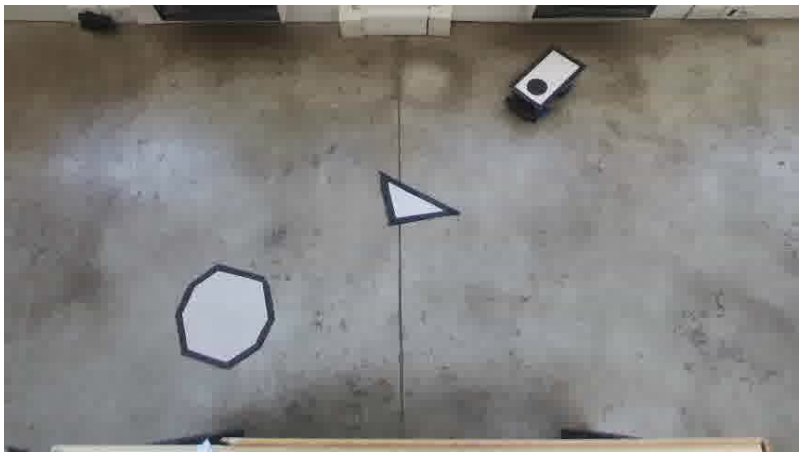

Figura 5.7: Experimento N°3: escenario original

Se deberá detectar tanto el Auto Robot, la meta y un único obstáculo, como muestra la figura 5.8

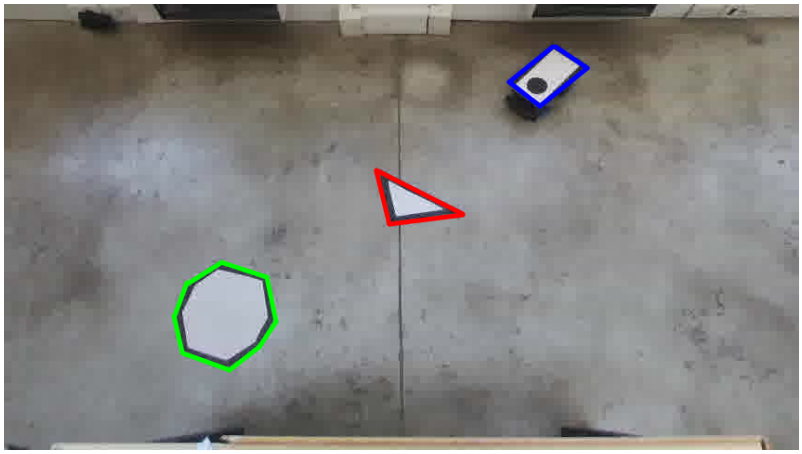

Figura 5.8: Experimento N°3: resultado esperado de la detección

Debido a que existe un obstáculo que atenta contra el libre desplazamiento del Auto Robot, la ruta a realizar deberá generar una curva que comience en el centro del robot y suavemente se tendrá que abrir hacia la derecha y, una vez esquivado el triángulo, deberá retomar en sentido opuesto hacia el destino. Ejemplo en la figura 5.9.

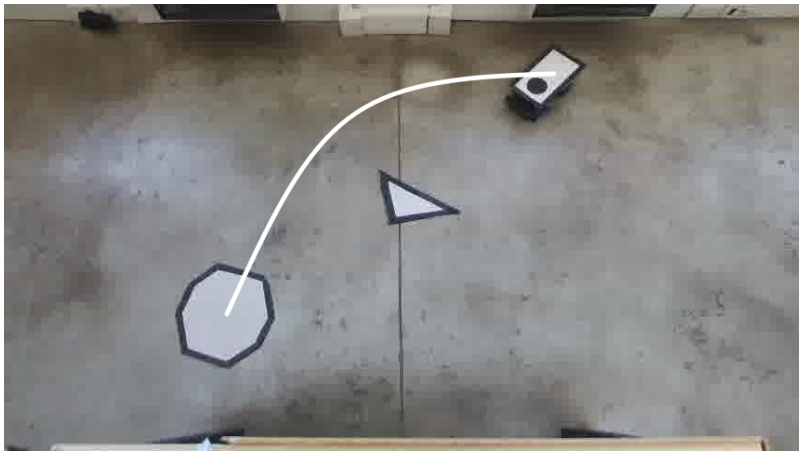

Figura 5.9: Experimento N°3: trayectoria a realizar

Se deberá dirigir al vehículo hacia la meta, siguiendo como referencia la trayectoria anterior (ejemplo en figura 5.10).

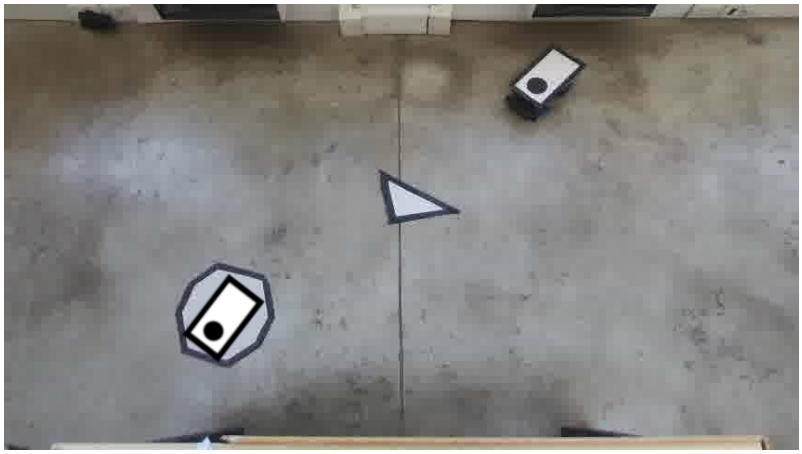

Figura 5.10: Experimento N°3: finalización del trayecto

5.1.4. Cuarto experimento

Es deseable complejizar el escenario, incluyendo una barrera de 4 obstáculos, con el propósito que se planifique una curva abierta, para dificultar aún más el traslado del Auto Robot. Se puede apreciar el escenario en la figura 5.11.

Figura 5.11: Experimento N°4: escenario original

Como siempre, se deberán detectar todos los objetos dentro del entorno; en este caso particular, son 4 los triángulos a encontrar (figura 5.12).

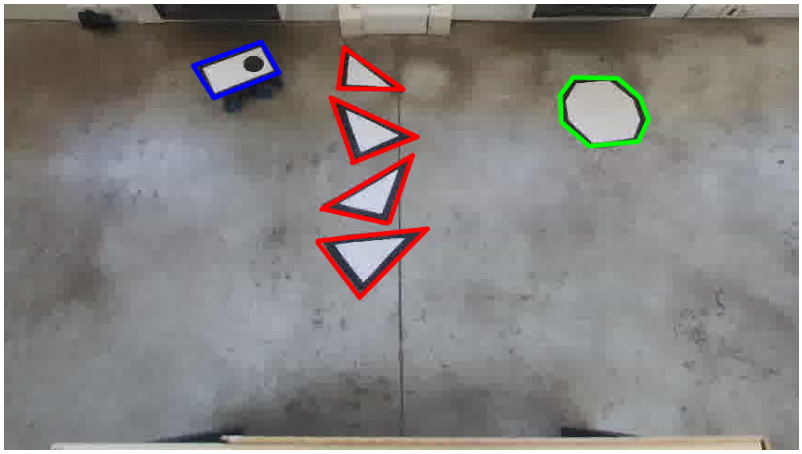

Figura 5.12: Experimento N°4: resultado esperado de la detección

En esta situación, el planificador deberá generar una curva que permita al Auto Robot esquivar la barrera por la parte baja del escenario (figura 5.13).

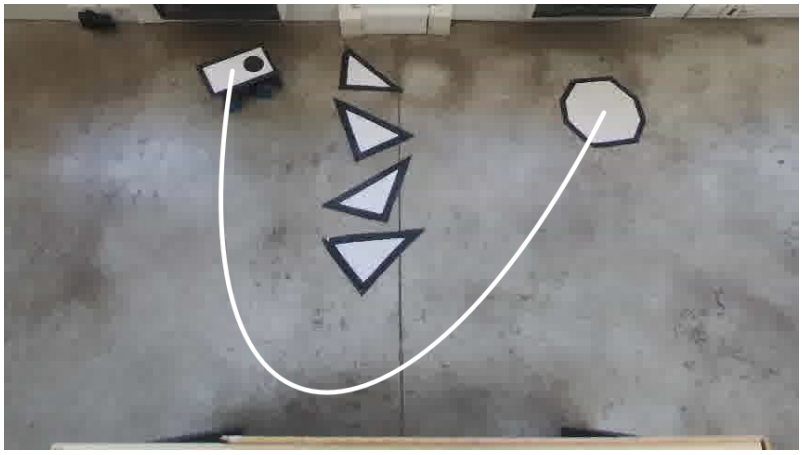

Figura 5.13: Experimento N°4: trayectoria a realizar

El vehículo se deberá desplazar tomando como referencia la curva, evitando colisionar con la barrera propuesta (figura 5.14).

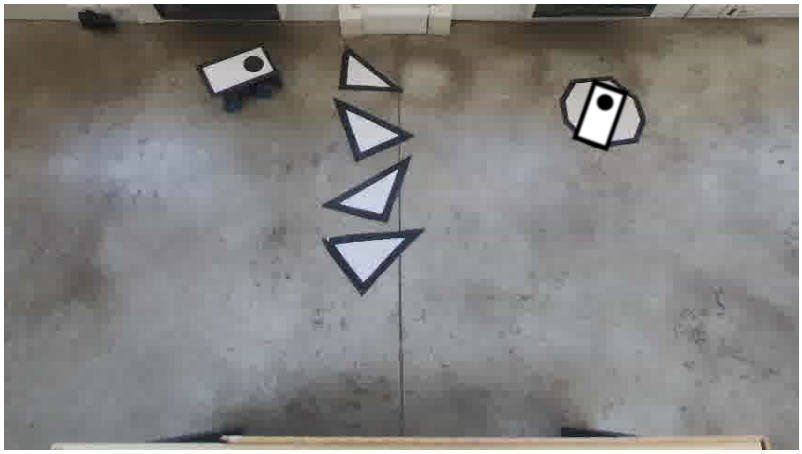

Figura 5.14: Experimento N°4: finalización del trayecto

5.1.5. Quinto experimento

En este experimento se desea rodear de obstáculos al destino, teniendo la responsabilidad el planificador de encontrar la ruta que permita el ingreso del vehículo sin colisionar con los mismos (figura 5.15).

Figura 5.15: Experimento N°5: escenario original

En este ejemplo se desea detectar 4 obstáculos que rodean al destino. En la figura 5.16 es posible visualizar una aproximación a cómo debería realizarse la detección.

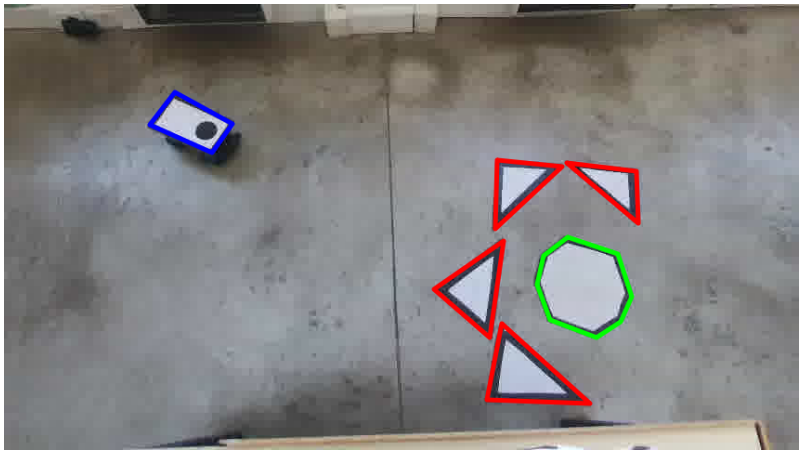

Figura 5.16: Experimento N°5: resultado esperado de la detección

El planificador debe encontrar la trayectoria que posibilite la entrada del Auto Robot al entorno de obstáculos (figura 5.17).

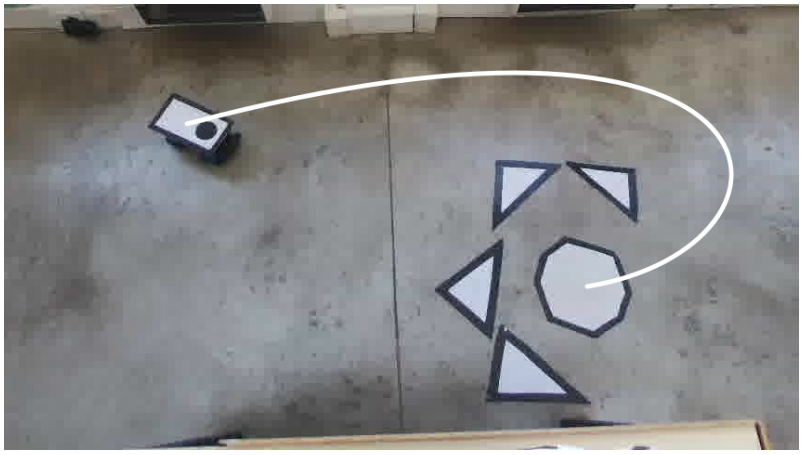

Figura 5.17: Experimento N°5: trayectoria a realizar

Como muestra la figura 5.18, el vehículo deberá ingresar al círculo de forma satisfactoria.

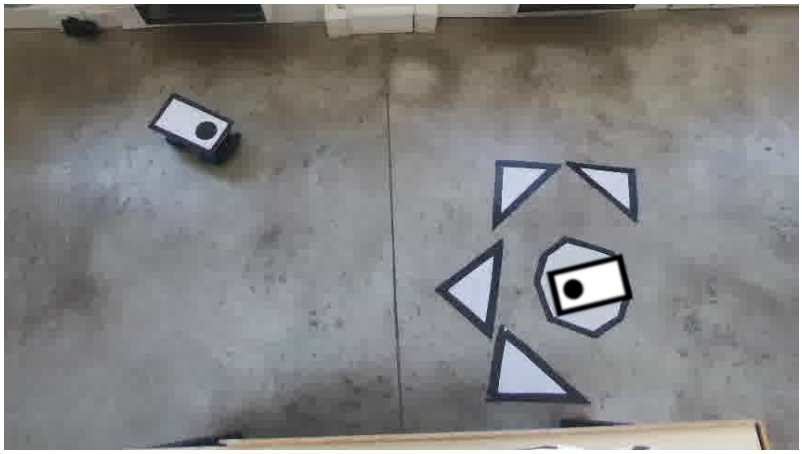

Figura 5.18: Experimento N°5: finalización del trayecto

5.1.6. Sexto experimento

La última prueba propone aumentar la complejidad del escenario, incluyendo obstáculos distribuidos de forma tal que se requiera direccionar múltiples veces en diversos sentidos. En la figura 5.19 se observa el experimento propuesto.

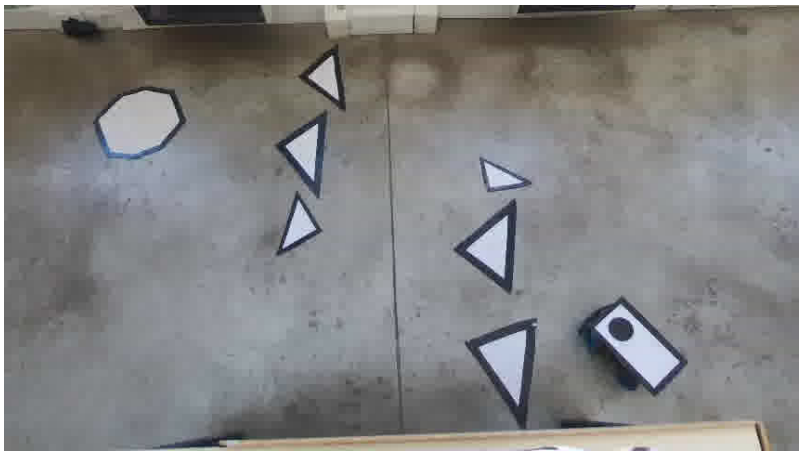

Figura 5.19: Experimento N°6: escenario original

Se deberá detectar tanto el Auto Robot, como el destino y 6 diferentes obstáculos (figura 5.20).

Figura 5.20: Experimento N°6: resultado esperado de la detección

Se requiere planificar la trayectoria de manera que no colisione con ningún obstáculo, como muestra la figura 5.21.

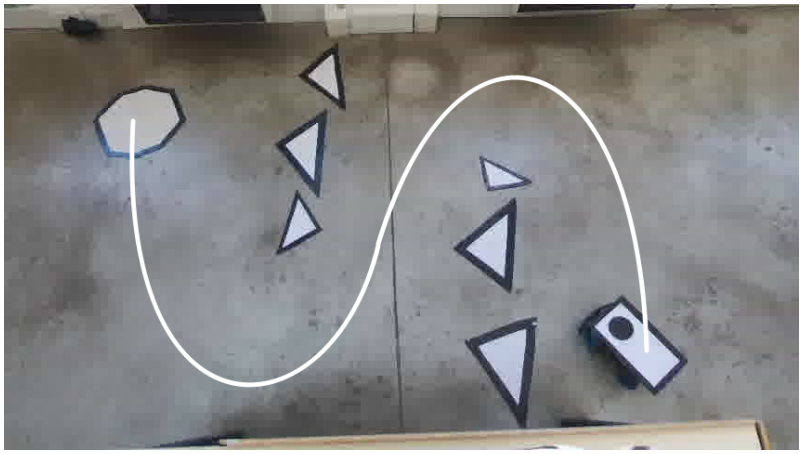

Figura 5.21: Experimento N°6: trayectoria a realizar

Se deberá dirigir al vehículo, direccionando las ruedas en el sentido que corresponda, hasta llegar al destino (figura 5.22).

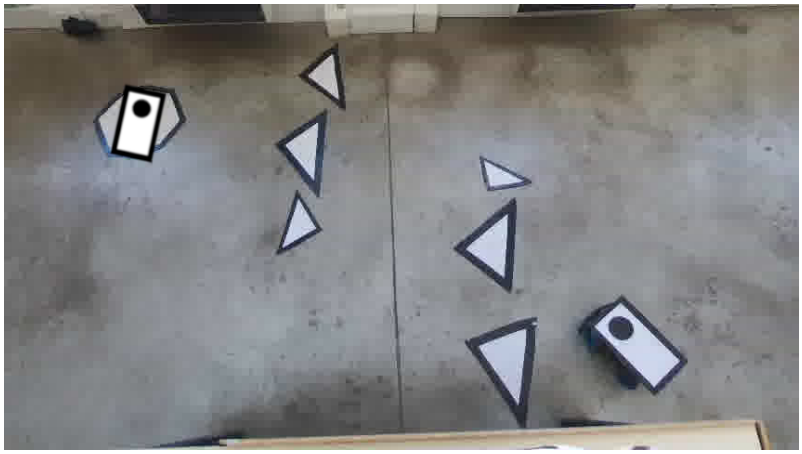

Figura 5.22: Experimento N°6: finalización del trayecto

Capítulo 6

Desarrollo e implementación

6.1. Introducción

En esta sección se detalla el funcionamiento del sistema de la presente tesina, donde son analizadas cada una de las partes y actores que conforman el mismo. Además, se detallan las diferentes problemáticas y desafíos abordados junto con las soluciones propuestas y los desarrollos realizados para el cumplimiento del objetivo.

El trabajo planteado cuenta, en primera medida, con un sistema Multi-Robots, conformado por un dron y un vehículo de cuatro ruedas programable. Por otro lado, se encuentra presente un servidor (instancia EC2) contratado en el Cloud público de AWS, donde reside casi toda la lógica computacional, implementaciones y desarrollos de las funcionalidades más complejas del sistema.

El escenario de este trabajo consiste en un vehículo, destino y posibles obstáculos; todos identificados en base a una figura geométrica.

Los objetivos que persigue el servidor son: por un lado, coordinar el envío y recepción de los mensajes para sincronizar el sistema Multi-Robots, y por otro lado, decodificar las imágenes provistas por el dron, procesar las capturas para la detección de los componentes, planificar el camino que conecte al Auto Robot con el destino evitando los obstáculos y calcular y enviar el movimiento a realizar por el vehículo.

Este procesamiento se realiza en tiempo real, por lo que se requieren diseñar, implementar y desarrollar algoritmos rápidos, eficientes en el uso de recursos y efectivos, debido a que una frustrada detección de los objetos, una tardía o errónea planificación del camino, o un cálculo de movimiento incorrecto, puede incurrir en el fracaso del sistema.

El usuario es la persona que inicia con la ejecución, a través de la correspon-

diente interfaz WEB, desplegada desde la instancia EC2. Luego, la instancia (encargada de interactuar con los robots) envía un mensaje al vehículo, para indicar el comienzo del sistema, por lo que el Auto Robot responde emitiendo diferentes características propias requeridas para la realización de las funciones generales. Cuando la instancia obtiene la respuesta, transmite un mensaje al dron para dar comienzo con su navegación. El dron despega y vuela hacia el centro del escenario, enviando en todo momento un video en tiempo real del mismo. Una vez que se encuentra posicionado sobre el escenario, emite un mensaje a la instancia para notificar este evento.

La instancia entonces, cuenta con las características del vehículo, y a su vez, recibe un video en tiempo real de lo que ocurre en el escenario; el Auto Robot se encuentra preparado para comenzar con el desplazamiento y el dron sobrevuela el entornos.

El servidor realiza tres funciones en paralelo: la primera consiste en la decodificación de las imágenes emanadas por el dron. Luego, cada imagen decodificada es enviada a una segunda función encargada de actualizar la interfaz WEB; en la interfaz será posible observar el streaming de video como así también el resultado de la detección y planificación, junto con una terminal que permite visualizar la interacción del sistema Multi-Robots. La tercer función consiste en procesar las imágenes para la detección los componentes, y su posterior planificación y cálculo del movimiento a realizar por el vehículo.

El procesamiento de imágenes consiste en encontrar tanto al vehículo, como el destino y los obstáculos; obteniendo las propiedades de cada uno de ellos (coordenadas de centro, radio, etc). Estos objetos están representados por figuras geométricas para facilitar su detección.

Luego de la detección de los componentes, se procede a encontrar el camino que permita conectar al vehículo con el destino, evitando los obstáculos. El cálculo de la trayectoria cuenta con dos fases que utilizan el algoritmo de planificación aleatoria en tiempo real RRT: la primer fase ocurre antes de comenzar con el desplazamiento del vehículo, es decir que no requiere una planificación en forma rápida. Para ello, se realiza el esfuerzo necesario para encontrar la ruta óptima, como se explicará en el apartado correspondiente. Obtenida la trayectoria óptima, se activa la segunda fase de planificación; en esta fase es usará la ruta óptima anteriormente mencionada como referencia, modificando la misma siempre y cuando sea necesario (por ejemplo a causa de la presencia de un nuevo obstáculo en el camino, o un desvío del Auto Robot no previsto).

Por último, se procede a calcular el movimiento a realizar por el vehículo, para ello se utilizan datos físicos del Auto Robot (enviados al principio del sistema, en el intercambio de mensajes iniciales) y se calcula la maniobra que

el robot debe ejecutar. Este movimiento consiste en un sentido de dirección (hacia delante o atrás) y el grado en el que el vehículo debe orientar sus ruedas. Finalizado el cálculo de la maniobra, se envía al vehículo para que el mismo la ejecute.

Cuando se determina que el auto robot alcanzó la meta, la instancia envía una notificación de finalización tanto al vehículo como al dron; este último deberá volver a su posición inicial, por lo que retorna hacia el punto de partida, desciende y posteriormente concluye con streaming de video.

Como ha sido mencionado, todas estas acciones son realizadas en tiempo real, delegando todo el procesamiento en una instancia EC2 en la nube y coordinado y comunicando a los integrantes del sistema Multi-Robots a través del envío de mensajes, lo que posibilita generar robots económicos y de bajo consumo energético, pero que pueden realizar funciones complejas, como es el caso del presente trabajo, en donde el vehículo se desplaza desde un inicio a un destino, evitando los obstáculos, equipándose con el hardware y software mínimo.

6.2. Auto Robot

6.2.1. Introducción

El primer paso para la realización del presente trabajo fue la confección de un prototipo que simula un vehículo tradicional. En primera medida se realizó el armado de la estructura del vehículo. Luego, se investigó sobre las diferentes placas microcontroladoras que permitan la manipulación de los sensores, como también posibiliten la conexión a la nube a través de una red WI-FI y posea un sistema de archivos que permite utilizar certificados de seguridad al transmitir y recibir mensajes a través del protocolo MQTT. Posteriormente, fue cuestión de debate el sistema de alimentación de energía tanto hacia las placas, como también a los actuadores y motores. Por último, se precisaron desarrollar los algoritmos necesarios con el fin de ejecutar los movimientos recibidos desde la nube.

6.2.2. Características y componentes

En este subapartado se detallan las diferentes características y componentes pertinentes al Auto Robot. Además, se expondrá las decisiones y problemáticas junto con el modo de resolución de las mismas, surgidas a lo largo de la evolución de la tesina.

6.2.2.1. Ensamblaje

Fue confeccionado un Auto Robot que simula un vehículo tradicional. El prototipo utilizado debe permitir gestionar los motores y dar la posibilidad de direccionar la estructura de forma precisa.

En un principio fue utilizado un chasis que consta de 2 capas de acrílico y 4 ruedas de plástico, junto con 4 motores (Auto Robot 4WD, figura 6.1).

Surgieron complicaciones en la utilización de este prototipo debido a que el suministro de energía hacia los cuatro motores no era igual, lo que repercutió en un desbalance en las velocidades de los mismos, provocando comportamientos irregulares en el desplazamiento del vehículo. Por otro lado, la orientación era otorgada a través de los motores sin un sistema de dirección, dificultando el direccionamiento del auto.

Figura 6.1: Auto Robot 4WD.

El prototipo mencionado fue reemplazado por una estructura que garantiza el desplazamiento mediante un mecanismo de tracción trasera y propicia el manejo de la orientación a través un sistema de dirección que modifica la posición de las ruedas delanteras. Este nuevo chasis consta de 1 capa metálica, pisos de acrílico, tuercas y tornillos de metal, un paragolpes de goma y 4 ruedas de plástico (figura 6.2); en la parte trasera del chasis se encuentra un motor de corriente continua que mueve un eje conectado a las dos ruedas (figura 6.4, posibilitando el desplazamiento del auto, mientras que en el otro extremo se dirige al robot orientando las ruedas delanteras a través un servomotor, desplazando un brazo metálico, que conecta ambas partes.

Figura 6.2: Tuercas, tornillos, switch, servomotor y otros componentes.

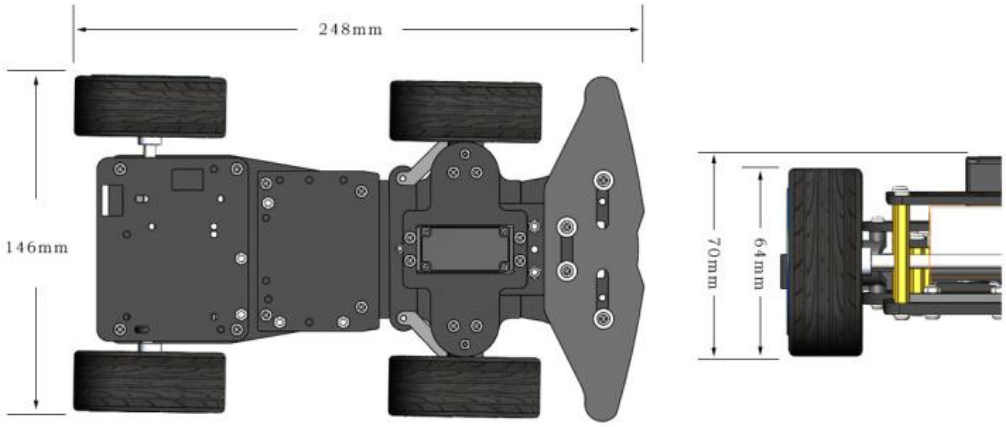

Figura 6.3: Auto Robot 2WD y sus medidas.

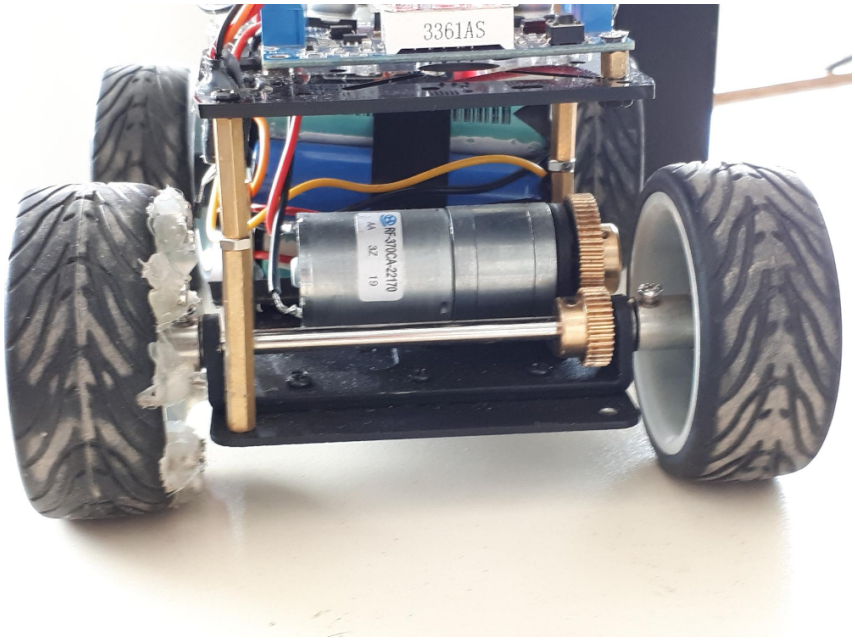

Figura 6.4: Auto Robot 2WD.

Se utilizó un servomotor modelo MG996R; es de propósito general y torque medio. Permite rotar aproximadamente en 180° (90° en cada dirección). Sus dimensiones son de 40.2*20.2*43.2 mm (figura 6.5).

Figura 6.5: Servomotor MG996R.

6.2.2.2. Placas de desarrollo

Luego de la confección del Auto Robot, se investigó sobre las posibles placas de desarrollo que permitan controlar tanto el vehículo junto los diferentes actuadores incorporados, como también transmitir información desde y hacia el Cloud, utilizando el protocolo de comunicación MQTT en forma segura (a través certificados y llaves de seguridad). Fueron investigadas las dos placas de desarrollo más comunes en robótica: Arduino Uno y Raspberry Pi 3.

- Arduino Uno: es una placa basada en un microcontrolador Atmega328. Cuenta con 14 pines de entrada/salida digital, 6 entradas análogas, un resonador cerámico de 16 MHz, un conector para USB tipo hembra, un Jack para fuente de Poder, un conector ICSP y un botón reset. No posee conexión a internet vía WI-FI por defecto. No provee un sistema de archivos.

- Raspberry Pi 3: es una placa basada en un procesador ARMv8 de cuatro núcleos, posee 17 GPIO con funciones SPI, I^2C y UART, 1 GB de SDRAM, puerto ethernet 10/100 MB, Bluetooth 4.1 y Wifi 802.11n. Permite la utilización de cualquier sistema operativo.

Entre las dos alternativas propuestas, fue elegida en forma contundente la utilización de la placa de desarrollo Raspberry Pi 3, principalmente debido a la posibilidad de conexión a internet a través de redes wireless y la propiedad de incorporación de un sistema de archivos, permitiendo la inclusión de certificados de seguridad para la transmisión vía MQTT.

Se optó por la utilización del sistema operativo Raspbian (figura 6.6); es una distribución del sistema operativo GNU/Linux basado en Debian Strech. Este sistema operativo es el utilizado por defecto en las placas Raspberry Pi.

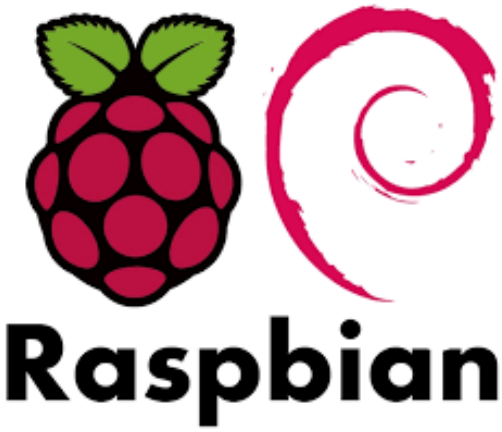

Figura 6.6: Sistema operativo Raspbian OS.

6.2.2.3. Alimentación y gestión de los actuadores

Se incorporó una placa controladora de motores RaspiRobot Board v3 (figura 6.7), la cual se encuentra conectada a los pines de la placa Raspberry Pi 3 permitiendo suministrar energía de forma constante a todas las placas, a los actuadores y al motor, facilitando su control y propiciando la gestión de la velocidad del mismo. Esta placa es un módulo de expansión compatible con Raspberry Pi 3, que brinda la interfaz de conexión entre la placa de desarrollo y los motores; incluye una fuente de alimentación conmutada para el suministro de energía, tanto para la placa Raspberry Pi como a los actuadores.

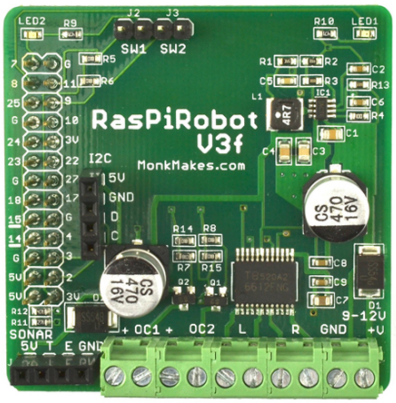

Figura 6.7: Placa controladora RaspiRobot Board V3.

Con el fin de suministrar la energía, en un principio se optó por confeccionar

un paquete de seis baterías de litio 18650 distribuidas en forma paralela (figura 6.11), conectadas a un regulador de voltaje que regulaba la entrada de 4V en una salida salida de 5V, alimentando la placa RaspiRobot Board V3, que como se explicó anteriormente, suministra la energía a los diferentes actuadores.

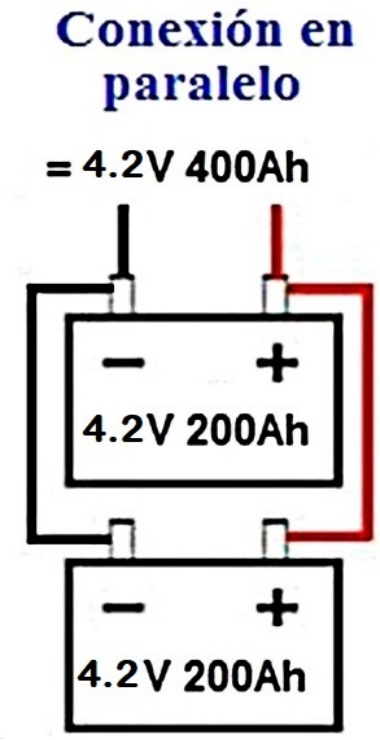

Figura 6.8: Ejemplo de conexión de baterías en paralelo.

Este sistema de alimentación no prosperó, debido a que se limitaba la velocidad de los motores, produciendo un bajo rendimiento al momento de la puesta en funcionamiento del robot, a causa del bajo suministro de energía. La solución a este problema fue realizar una distribución del paquete de baterías en serie-paralelo (ejemplo en la figura 6.9), obteniendo de esta forma mayor nivel energético de salida, y la incorporación de un regulador de voltaje step-down (figura 6.10), que regula la entrada de 12V en una salida de 9V, lo que permitió aumentar la velocidad de los motores, sin condicionar el rendimiento general.

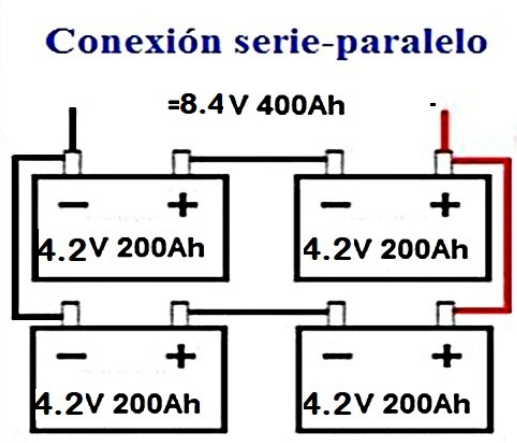

Figura 6.9: Ejemplo de conexión de baterías en serie-paralelo.

Figura 6.10: Fuente DC Step-Down.

Fue confeccionado un cargador de baterías (figura 6.13), conformado por una placa madre de una netbook EXO modelo 355 y un regulador de carga, en donde se pueden conectar las baterías a través de una ficha, facilitando la conexión de las mismas.

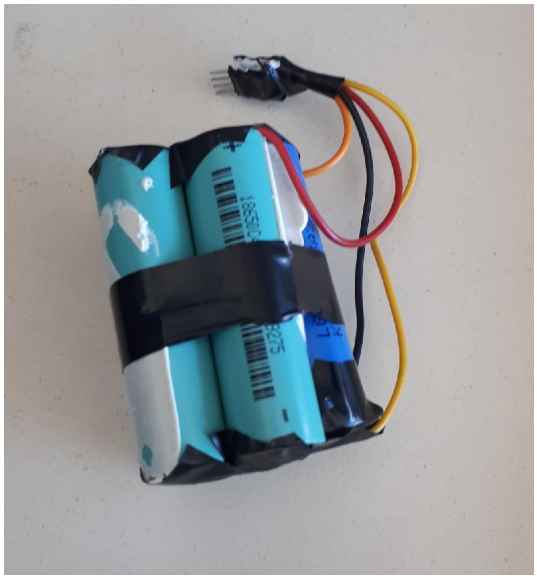

Figura 6.11: Paquete baterías de litio 18650.

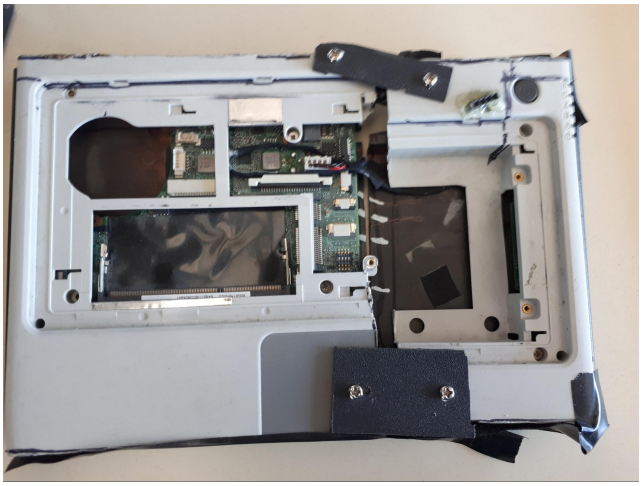

Figura 6.12: Cargador de baterías.

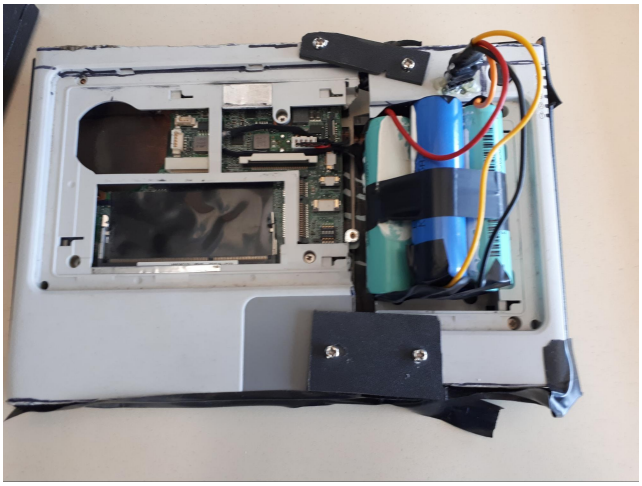

Figura 6.13: Cargador con baterías conectadas.

6.2.3. Desarrollo

6.2.3.1. Funcionamiento

El Auto Robot recibe desde el Cloud cada movimiento que debe realizar en todo momento. Un movimiento se conforma por un un ángulo al que deberá direccionar sus ruedas ([-180°, 180°]) y el sentido en el que debe desplazarse (hacia delante o atrás).

Para cumplir con las directivas comandadas, el vehículo deberá contar con las capacidades que permitan alterar su comportamiento, ya sea modificando la dirección de las ruedas, como así también el sentido de su desplazamiento.

6.2.3.1.1 Modificación de la dirección

Con el fin de orientar la estructura, el Auto Robot cuenta con un Servomotor MG996R, encargado de direccionar las ruedas a través de un sistema compuesto por dos brazos metálicos que conecta ambas partes. En la figura 6.14 se puede visualizar el sistema de direccionamiento.

Este sistema restringe la orientación de las ruedas a un máximo de 30 grados hacia los extremos. Luego de realizar diferentes pruebas se determinó que los valores límites aceptados por el servomotor son 6.5, 10 y 13.5; correspondientes a los grados 30 (extremo izquierdo), 0 (centro) y -30 (extremo derecho) respectivamente. En base a estos datos, se diseñó la ecuación siguiente, siendo “deg” el grado al que deberá direccionar las ruedas y “R” la rotación que tendrá que realizar el servomotor:

$$R = 10 - 0{,}125 * deg - 0{,}0002777778 * deg^2$$

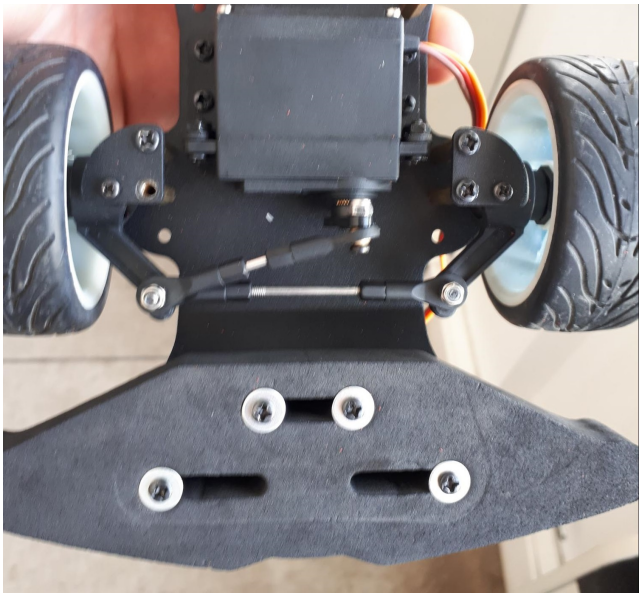

Figura 6.14: Sistema de direccionamiento.

6.2.3.1.2 Algoritmos desarrollados

La inteligencia del robot es provista a través de algoritmos desarrollados en JavaScript, haciendo uso de la aplicación Node-RED. Estos algoritmos están insertados en nodos que residen en flujos particulares, facilitando la instanciación y modificación del comportamiento global.

Por un lado se encuentra un flujo principal (figura 6.15), que contiene nodos MQTT para recibir y emanar datos desde y hacia el Cloud; y nodos para inicializar variables utilizadas en forma general por todos los flujos. Cuando se recibe un mensaje proveniente de la nube vía MQTT, son activados dos subflujos: “Steeringment” y “Movement”.

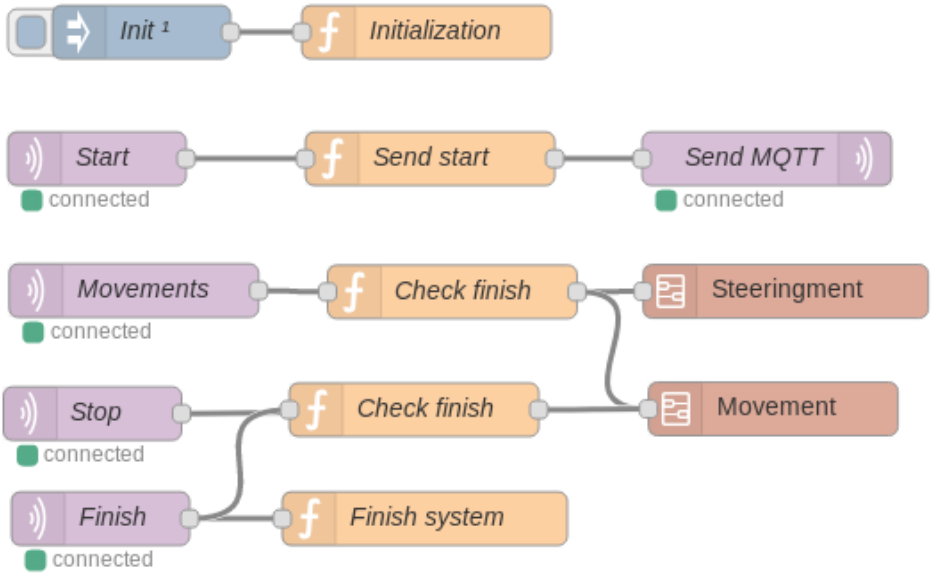

Figura 6.15: Flujo principal del robot.

El subflujo de direccionamiento ("Steeringment", figura 6.16), cuenta con un nodo "Convert degree to servo" que realizará la conversión del grado a dirigir recibido, al correspondiente número propio del servomotor (utilizando el cálculo explicado en el apartado anterior). Luego, es enviado el resultado al nodo "MG996R"; en este nodo se encuentra configurado el número de pin de la Raspberry Pi 3 donde fue conectado el servomotor, por lo que todo número enviado al mismo, repercutirá en modificaciones en el sensor y por consiguiente en la dirección de las ruedas.

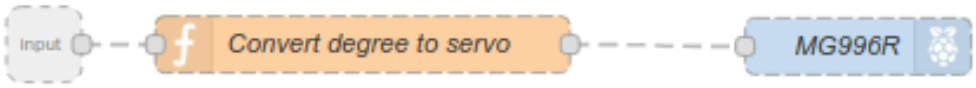

Figura 6.16: Flujo "Steeringment".

En relación al flujo de desplazamiento ("Movement", figura 6.17), contiene un nodo llamado "Update movement", utilizado para modificar el sentido en el que se está desplazando el vehículo, en base a lo comandado por la instancia. Esta modificación consiste en activar/desactivar los nodos "Right motor pin

1" y "Right motor pin 2", lo que permite desplazarse hacia delante, en reversa, o simplemente frenar con el movimiento. Para ello, se encuentra un nodo intermedio "Filter activation", que consiste en un una estructura de control "case", recibiendo el valor entero 1 cuando se desea desplazar hacia delante, 2 cuando se requiere invertir el sentido del desplazamiento y 0 para parar el motor. A su vez, se encuentra el nodo "Speed" que permite modificar la velocidad del motor del vehículo.

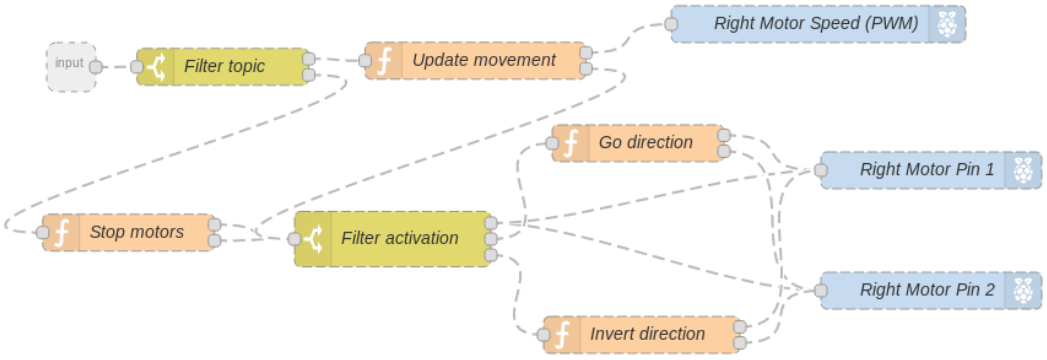

Figura 6.17: Flujo "Movement".

6.2.3.2. Mensajes y tópicos

El nexo de comunicación entre el vehículo y el Cloud es realizada a través del protocolo MQTT en el puerto 8883, utilizando certificados de seguridad provistos por el servicio IoT de AWS.

El Auto Robot se subscribe al tópico "RobotStart", "RobotStop" y "RobotFinish" en donde recibe una notificación que se corresponde al inicio, al freno del auto y finalización del sistema respectivamente. A su vez, se suscribe en el tópico "RobotMove" por el cual recibe los movimientos a realizar. El Auto Robot publica en el tópico "RobotReady" confirmando que está preparado para comenzar; donde envía el grado de apertura máximo al que puede dirigirse.

6.2.4. Conclusiones

Fue confeccionado un prototipo económico y de bajos recursos que a través de la algoritmia desarrollada ejecuta los movimientos recibidos desde el Cloud, a través de MQTT, ajustando su comportamiento mediante sus componentes integrados. El prototipo se abstrae de la lógica necesaria para determinar la

percepción del escenario, de la planificación del camino y el cómputo de los movimientos a realizar, siguiendo únicamente las órdenes provenientes desde el Cloud, resultando en un robot equipado con un hardware y software mínimo, lo que incide en un menor consumo energético; y haciendo uso de la aplicación Node-RED, permite alcanzar un desarrollo simple y de fácil mantenimiento, pero que a su vez satisface las metas en forma efectiva.

Fueron superados y solucionados diferentes problemas que surgieron a lo largo del trabajo, referentes en gran medida a aspectos físicos, subsanados en algunas situaciones con actuadores y cálculos posteriores (como el caso del direccionamiento), o por desarrollos propios de componentes externos (la nueva administración de las baterías y la confección de un cargador para las mismas).

El Auto Robot puede alcanzar un destino establecido, evitando los obstáculos presentes en el escenario como también los límites del mismo, utilizando como únicos requisitos conexión a internet, y un mecanismo de desplazamiento (motores) y dirección (servomotor), lo que abstrae al sistema Multi-Robots del tipo de robot a utilizar, permitiendo el uso del mismo por parte de cualquier robot móvil básico.

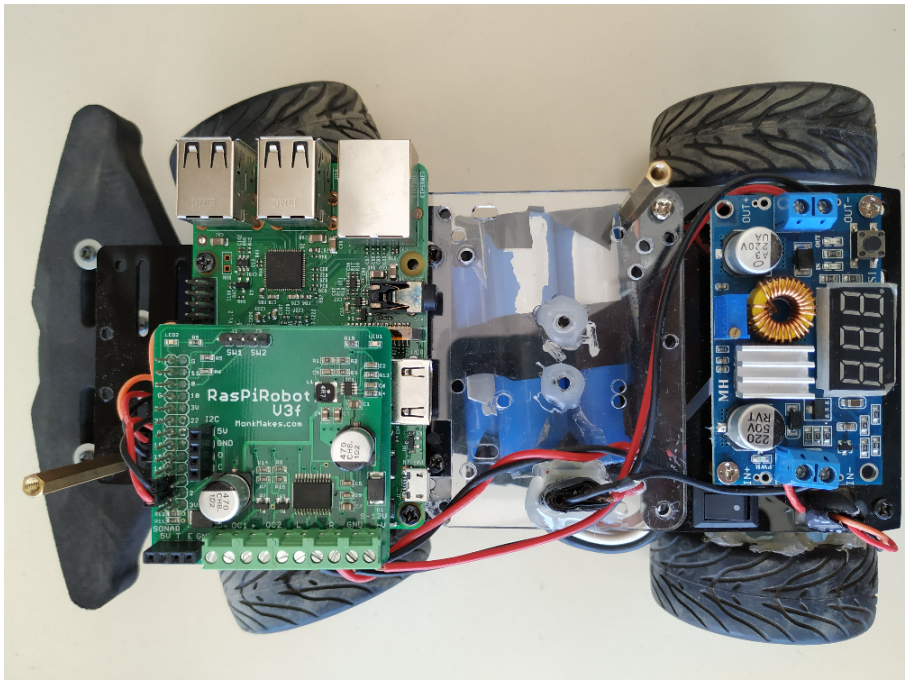

Figura 6.18: Auto Robot final.

6.3. Cuadricóptero

6.3.1. Introducción

El sistema propuesto requiere de un método que capte imágenes del escenario y las envíe al Cloud, debido a la necesidad de obtener información completa del entorno para su posterior procesamiento. Es necesario entonces, la utilización de un dispositivo que capture imágenes aéreas y emane las mismas en tiempo real al servidor de internet.

El propósito de este apartado en primera medida consiste en plantear el debate de los diferentes dispositivos a cumplir con esta funcionalidad, detallando tanto ventajas y problemáticas de los mismos, junto con las experimentaciones realizadas.

Posteriormente, se explica el diseño y la implementación realizada para efectuar la comunicación con el dispositivo elegido.

Por último, se expone el modo de conexión al Cloud para la realización del envío del streaming de video.

6.3.2. ¿Por qué un dron?

Inicialmente, el sistema estaba compuesto por un Auto Robot que contenía una cámara local para captar imágenes del escenario y sensores ultrasónicos para sustentar la percepción. Este mecanismo de captación de información del entorno no prosperó, debido a que no permitía visualizar el escenario por completo, como consecuencia del ángulo de visión horizontal que propone la alternativa; por ende, fue necesario abocarse en una opción que efectivamente capte el escenario en su completitud.

Primeramente fue planteada una solución que consiste en la inclusión de una cámara aérea fija, que enviaba las capturas al Cloud, donde se procesan para localizar e identificar los objetos dentro del escenario.

Luego de diferentes experimentos, surgieron problemáticas respecto a la posición de la cámara, ya que al estar fijada, limita la visión del escenario debido a la imposibilidad de modificar la altura de la misma en forma dinámica, además de impedir la portabilidad del sistema.

La solución final radica en la incorporación de un dron, que permite tener un ángulo de visión útil a través de la cámara incorporada, con la posibilidad de pilotar automáticamente el dispositivo a fin de obtener una mejor precisión en las capturas de las imágenes. Fue necesario investigar y experimentar con diferentes dispositivos de vuelos, debido a que cada uno presenta características, habilidades y facilidades diferentes.

En apartados posteriores se describen las pruebas, diferencias y conclusiones entre los tipos de drones experimentados.

6.3.3. Propósito y funcionalidad

La finalidad del dron consiste en realizar una captación de imágenes del escenario en tiempo real. Para alcanzar este objetivo, una vez que el sistema inicia, el dron debe elevarse a una altura preestablecida (1,5 metros) y desplazarse hacia el escenario. La cámara tiene una visión orientada hacia el suelo (-90°) y envía al Cloud un el streaming de video correspondiente. Cuando el sistema finaliza, el dron debe retornar a su posición inicial, finalizando con el video.

6.3.4. Elección del dron

A continuación, se mencionan las características de los drones investigados y se describe la experimentación realizada que compara, a fines prácticos, ambos candidatos.

- Minidrone Cargo: pequeño dron con conexión Bluetooth que está diseñado para pilotar en espacios reducidos. Cuenta con una cámara orientada 90° hacia el suelo permitiendo capturar imágenes de 640x480 pixeles y diversos sensores, como ultrasónico para calcular distancia al suelo, giroscopio para el cálculo de la orientación, entre otros. Alcanza un rango de uso de 20 metros. Disponible en dos modelos: Mars y Travis [46].

Figura 6.19: Parrot Minidrone Cargo Mars.

- Bebop 1: dron de tamaño mediano con conexión WiFi y una cámara frontal que permite capturar imágenes y video en un rango de 180° en

todas las direcciones mediante el lente “Fish-Eye” y el software propio desarrollado por la empresa. Cuenta con diversos sensores, entre los que se destacan: giroscopio para calcular su orientación, GPS, ultrasónico para calcular la distancia del dispositivo al suelo y cámara para calcular velocidad horizontal, entre otros. También implementa un sistema de streaming de video encodeado en formato H.264 y con resolución 856x640 pixeles de manera eficiente [47].

Figura 6.20: Parrot Bebop.

Dada la posibilidad de utilizar dos drones, fue necesario realizar pruebas para determinar la mejor opción a integrar al sistema Multi-Robots.

En primera medida se investigó el desempeño del Minidrone Cargo. Para ello fue utilizada una librería desarrollada en NodeJS que brinda la posibilidad de conectarse con el dron, pilotar el mismo, capturar y descargar imágenes a través de su cámara.

Fueron realizadas pruebas de movimiento y captación de imágenes; para ello se precisó realizar una experimentación que consiste en elevar al dron, pilotarlo hasta una distancia preestablecida, efectuar capturas y retornar al punto de inicio.

El resultado del experimento no fue el deseado, debido a que el Minidrone cargo tuvo una mala estabilidad de vuelo generando errores en los movimientos comandados, lo que repercute en la incapacidad de alcanzar el punto de fin establecido para tomar una captura consistente del escenario. Por otro lado, la obtención de imágenes al dispositivo intermediario solo era posible cuando el dron se encontraba estacionado, impidiendo el envío de la imagen en vuelo; cuestión requerida para la realización del trabajo debido a que es necesario enviar un video en tiempo real de las imágenes al Cloud.

Luego, se experimentó con el dron Bebop 1. En este caso se utilizó el SDK que brinda Parrot para sistemas UNIX, en lenguaje C.

Siguiendo el mismo objetivo mencionado anteriormente, los resultados fueron satisfactorios: el dron se desplazó en forma estable hacia el escenario, respetando las directivas programadas con antelación, y además, fue posible obtener el streaming de video en forma rápida y efectiva.

Como conclusión, el dispositivo de vuelo que mejor se adapta al sistema propuesto y el utilizado en la presente tesina es el drone "Parrot Bebop 1", ya que permite realizar un vuelo preestablecido de forma estable, posee un ángulo de visión de la cámara útil para el sistema propuesto y a su vez posibilita el envío en tiempo real de las imágenes del escenario en forma eficiente, encodeando las mismas en H.264 para una transmisión de mayor velocidad.

6.3.5. Desarrollo

En este apartado se detalla el diseño realizado en cuanto a la conectividad del dron al Cloud, los desarrollos propuestos respecto al mecanismo de control y coordinación y el modo de intercambio de información con el dispositivo de vuelo. Además, se explican los flujos Node-RED desarrollados junto con los mensajes y tópicos enviados desde y hacia la instancia.

6.3.5.1. Conexión del dron al Cloud

Una particularidad del dron elegido es la imposibilidad de realizar una conexión directa a internet. Esto surge debido a que su interfaz WiFi funciona como punto de acceso o "Access Point" (AP), lo que obliga al usuario a utilizar un dispositivo capaz de conectarse a una red WiFi.

Para solucionar este problema, se utiliza una Raspberry Pi 3 B (RPI). Esta placa cuenta con dos interfaces de red integradas: una WiFi y otra Ethernet; la interfaz de WiFi conecta al AP del dron y la conexión a internet se efectúa a través de la correspondiente interfaz Ethernet.

En la figura 6.21 se puede observar la placa de desarrollo conectada a Ethernet, alimentada directamente a través de una fuente de 5V.

Figura 6.21: Raspberry Pi 3 B usada para comunicar el dron con el Cloud.

6.3.5.2. Control y comunicación con el dron

Una vez lograda la conexión entre la nube y el dron, fue necesario gestionar la comunicación entre los dos puntos.

El cuadricóptero genera un streaming de video hacia la RPi, quien redirige las imágenes al Cloud.

Con el fin de gestionar el control y acciones del dron fue desarrollado un programa en C y el respectivo flujo en Node-RED. Ambos programas se comunican a través de un socket TCP.

Entre los comandos más importantes se encuentran:

- takeOff: inicio de vuelo del dron (despegue).
- land: aterriza el dron.
- up:cant: indica al dron que se eleve por cierto tiempo hacia arriba.
- forward:cant: indica que se desplace por cierto tiempo hacia adelante.
- startStreaming: comienza con la trasmisión de imágenes en tiempo real.
- stopStreaming: finaliza con la trasmisión de imágenes.
- exit: finaliza la conexión con el dron y el programa en C.

La interacción desde Node-RED y la nube se realiza a través del protocolo MQTT. En cambio, la conexión entre la aplicación Node-RED y el programa en C es efectuada a través de un socket TCP. Se muestra de forma gráfica en la figura 6.22.

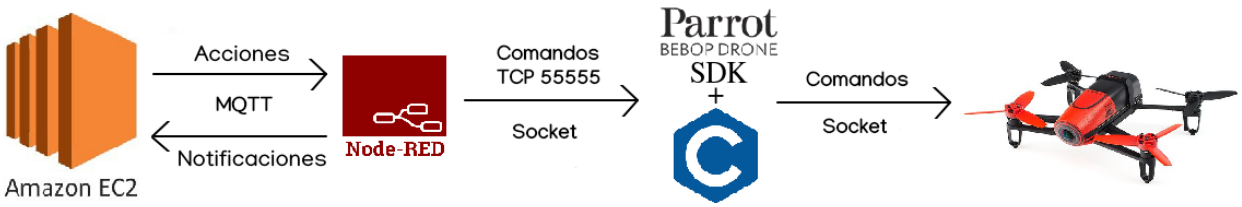

Figura 6.22: Conexión del dron, RPi y la nube.

6.3.5.2.1 Conexión con el dron

Se utilizó la SDK oficial de Parrot de Unix para desarrollar el software que interactúa con el dron. Este programa escrito en lenguaje C, traduce los comandos enviados desde la aplicación Node-RED y los envía hacia el dron mediante directivas de la librería oficial.

A continuación, se describe el funcionamiento del software desarrollado.

El programa comienza con la función "droneConnection(void)", utilizada para conectar al dron dentro de la red. En caso de desconexión en cualquier punto de la ejecución, el programa intenta conectarse nuevamente hasta lograrlo de forma automática.

Efectuada la conexión, se ejecuta la función "droneInitialization(void)" que envía al dron una serie de comandos referentes a configuraciones de pilotaje y acciones iniciales a realizar. En el anexo 1 se explican las más importantes.

Luego de la inicialización es llamada la función "socketCommandReceive(void)"; dentro de esta función se crea un socket en el puerto TCP 55555 para posibilitar la recepción de comandos desde el flujo Node-RED. Lograda la creación del socket, el programa queda a la espera de algún comando en el puerto anteriormente mencionado. Ante la recepción, se ejecuta la función "executeCmd(...)", que recibe un comando y se lo envía al dron (Anexo 2).

Por último, al recibir el comando "exit" desde Node-RED, el programa finaliza la funcion "socketCommandRecive()" y llama a "droneDisconnection()", finalizando la conexión y terminando el proceso de forma segura.

6.3.5.2.2 Flujo en Node-RED

El flujo desarrollado (figura 6.23) funciona como nexo de comunicación entre el dron y el Cloud. Recibe acciones desde la nube referentes a la gestión

del cuadricóptero. Además, notifica a la misma la finalización de las acciones mencionadas. Por otro lado, se comunica a través de sockets TCP con el programa desarrollado en C. Esta comunicación permite para enviar comandos al dispositivo de vuelo, como se detalló en el apartado anterior.

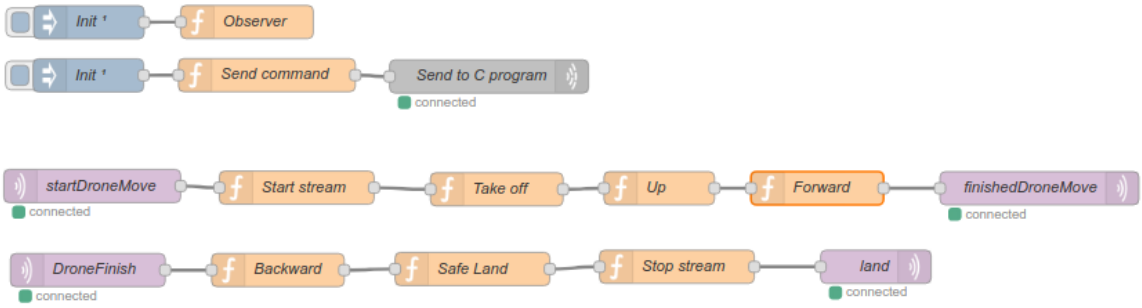

Figura 6.23: Flujo de comunicación con del dron.

Los nodos de color violeta indican envío o recepción de mensajes MQTT. Por otro lado, los nodos de color naranja son funciones que activan comandos los cuales son enviados al programa en C usando un socket TCP a través del nodo color gris.

6.3.5.2.3 Reenvío de streaming de video al Cloud

El video en tiempo real es generado automáticamente por el dron luego de haber recibido un comando de comienzo de trasmisión. El streaming se envía al dispositivo conectado al AP que emitió el comando (en el presente trabajo, la Raspberry Pi 3), en un formato RAW, encodeado en H.264 y con una resolución de 856x480 pixeles. La transmisión es realizada a 24 FPS. El protocolo que utiliza el dron para esta transmisión es Real-Time Transfer Protocol (RTP) el cual transporta sobre UDP en el puerto 55004 (figura 6.25). La Raspberry redirige estos paquetes UDP recibidos del streaming hacia el Cloud utilizando una política de ruteo configurada en el firewall del sistema operativo, como se muestra en la figura 6.24. Esta indica que todos los paquetes con puerto destino 55004 sean enviados al destino y puerto configurado.

```
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     udp  --  anywhere             ec2-34-230-200-176.compute-1.amazonaws.com  udp dpt:55004
ACCEPT     udp  --  anywhere             localhost           udp dpt:55004
```

Figura 6.24: Resultado de “iptables –list”.

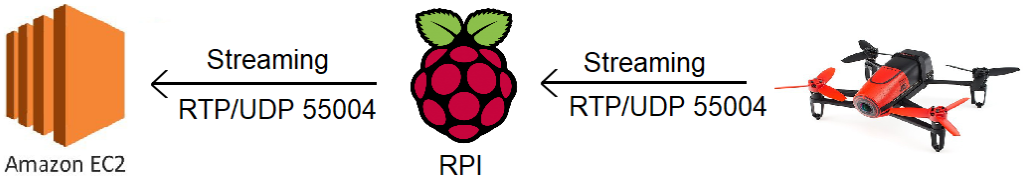

Figura 6.25: Camino que realiza el streaming de video.

6.3.6. Conclusiones

Se investigaron y subsanaron problemáticas referentes a la percepción del escenario en su completitud para su posterior procesamiento, concluyendo en la utilización de un dron como mejor alternativa.

Se investigó y realizaron diferentes experimentos con distintos tipos de drones, donde fueron comparadas características de los mismos y el resultado de las pruebas, siendo útil para la realización de la presente tesina el dron "Parrot Bebop 1". Basado en el SDK oficial, se desarrolló un programa que permite la conexión del dron y el pilotaje del mismo.

Además, se diseñó e implementó un mecanismo de comunicación entre el dron, la RPi y el Cloud, detallando los aspectos lógicos y técnicos que proporcionan la interacción y el intercambio de información entre estos actores. Por último se explicó el mecanismo de envío de las imágenes en tiempo real proporcionadas por el dron.

Como conclusión final, se incorporó al sistema Multi-Robots un dispositivo de vuelo, capaz de desplazarse en forma autónoma hacia el escenario, quien se responsabiliza de la captación completa del entorno y el posterior envío en tiempo real al Cloud.

6.4. Servidor en la nube

6.4.1. Introducción

Para la presente tesina fue contratada una instancia del servicio EC2 de AWS, se encuentra alojada en los Estados Unidos. Como se mencionó en la sección de introducción, la instancia adquirida es del tipo T3.small, la misma posee el sistema operativo Ubuntu 18.04 LTS; se instaló el lenguaje de programación Python 3.7 y las librerías necesarias. Además, se incorporó el programa FFMPEG utilizado para realizar la decodificación de las imágenes (como se explicará en la sección de video en tiempo real).

La instancia EC2 será la encargada de computar la lógica del trabajo, cumpliendo con los aspectos funcionales del sistema, lo que permite desprender la responsabilidad del procesamiento a los robots, interactuando con los mismos mediante el envío de los mensajes correspondientes a través del protocolo MQTT.

Dentro de la instancia, se desarrollaron e implementaron algoritmos que permiten detectar figuras geométricas en un escenario real, a través de imágenes provenientes de una cámara aérea, previa decodificación de las mismas. Además, ejecuta algoritmos de planificación de caminos que encuentra el trayecto a realizar basándose en coordenadas de inicio, destino y obstáculos, sin colisionar con estos últimos; utilizando la matemática necesaria para transformar las coordenadas del trayecto resultante en directivas que el Auto Robot pueda comprender y realizar. Asimismo, brinda una interfaz WEB para realizar simulaciones de planificaciones de caminos eligiendo entre diferentes algoritmos de planificación y permite comenzar la ejecución del sistema. La misma está formada por tres apartados: una pantalla de notificación que informa las diferentes acciones que se están llevando a cabo, los resultados del procesamiento de la imagen junto con el trazo de la ruta a realizar y un video del escenario en tiempo real.

El sistema deberá decodificar las imágenes, procesar las mismas para detectar los objetos del escenario, planificar la ruta a recorrer y transformar estos resultados en movimientos a realizar por el vehículo, en tiempo real.

Esto condiciona a generar un sistema que procese en forma eficiente las acciones previamente mencionadas; teniendo un adecuado uso de recursos y la lógica pertinente que permita obtener los resultados de forma efectiva en un tiempo considerablemente bajo.

6.4.2. Sistema desarrollado

Fue implementado un sistema en el lenguaje Python 3.7 que provee funcionalidades para decodificación de capturas en formato H.264, procesamiento de imágenes para detectar figuras geométricas, planificación de caminos utilizando diversos planificadores y transformación de los resultados de la planificación en movimientos que el robot debe ejecutar; además, cuenta con una interfaz WEB que posee un apartado para realizar simulaciones de caminos y otra sección para controlar la ejecución general del sistema, en forma online.

Se planteó un diseño jerárquico del sistema: el sistema mayor (figura 6.26), está compuesto por un módulo principal llamado “main” que es el enlace hacia todas las funcionalidades y subsistemas del mismo, enviando los parámetros correspondientes al módulo “master”, encargado de la activación de la acción requerida. Además, se dispone de tres módulos: “model”, “helper” y “excep-

tions" para la declaración de clases, métodos de ayuda y excepciones personalizadas, respectivamente; utilizados en los diferentes módulos y paquetes del sistema.

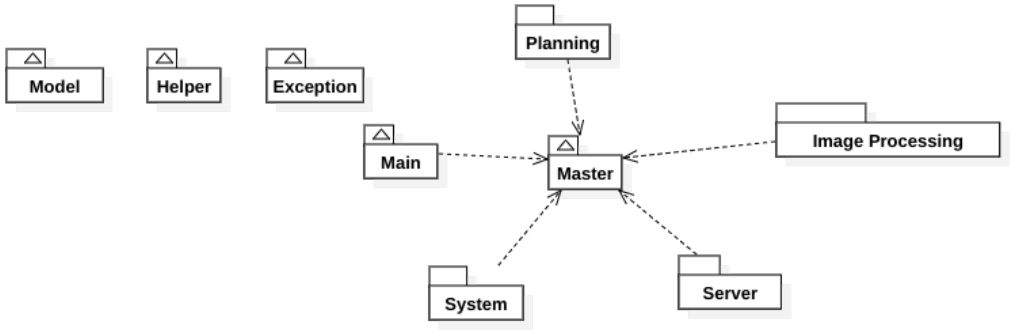

Figura 6.26: Diagrama de clases general del sistema mayor.

Previo a la ejecución del sistema, se deben especificar diferentes aspectos en un archivo de configuración ".ini", como dirección y puerto del servidor WEB, el tipo de planificador y un archivo de configuración especial para este sistema.

```
1   ;config.ini
2
3   [DEFAULT]
4   MODE = SERVER
5
6   [SERVER]
7   ADDRESS = 0.0.0.0
8   PORT = 1880
9   PLANNER = RRT
10  CONFIG = config/systemConfig.yaml
```

Figura 6.27: Archivo de configuración "config.ini"

6.4.2.1. Sistema servidor

Este sistema es el encargado del despliegue de la interfaz WEB y de controlar y accionar ante la interacción de los usuarios sobre la misma. Además, contiene la lógica desarrollada respecto al cómputo en tiempo real de las funcionalidades del trabajo, paralelizando las actividades de la mejor forma, con

el fin de aprovechar los recursos de la instancia y de administrar el uso de la CPU según sea necesario.

En primera medida se debe establecer un archivo de configuración (figura 6.28), donde se especifica el servidor, certificados y puerto de la conexión MQTT del servicio de IoT de AWS; es necesario indicar la ruta de donde proviene el streaming, en el caso del presente trabajo es un archivo de texto con extensión ".sdp". Por último, se debe configurar la secuencia de mensajes tanto de publicación como suscripción a los diferentes actores. Para esto, se establecen 3 diferentes estados:

- Starting: este estado ocurre previo a la ejecución del sistema. En la presente tesina, en un principio se publica en el tópico RobotStart (hacia el robot) y se espera en el tópico RobotReady, recibiendo por ejemplo, como mensaje "maxSteer".

- Running: el estado running indica que el sistema está en ejecución, es decir, se están decodificando las capturas, procesando las mismas, planificando el camino y transformando los resultados a movimientos.

- Finishing: cuando el sistema finaliza, es decir, se determina que el vehículo alcanzó el destino, da comienzo a este estado, y se respetarán la secuencia de publicaciones y suscripciones correspondientes.

Como puede observarse, para cada estado se presenta una cláusula "actions", donde declara aquellas publicaciones y suscripciones a realizar por cada uno de ellos. A su vez la cláusula "log" es utilizada para determinar los mensajes a informar en el correspondiente apartado en la interfaz WEB.

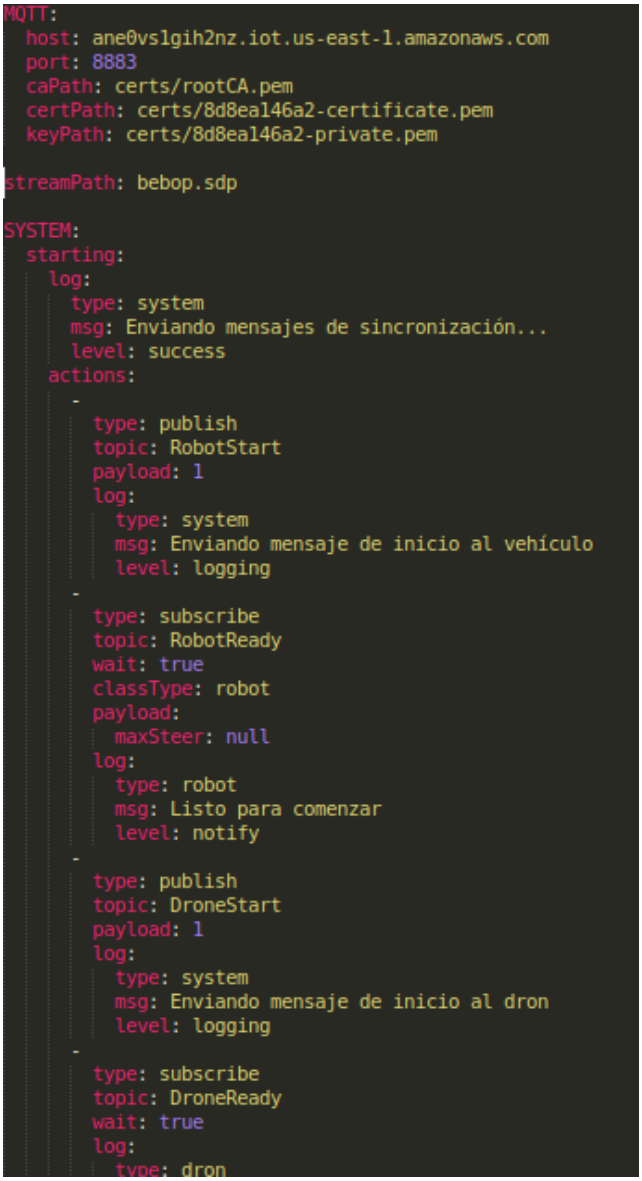

Figura 6.28: Archivo de configuración del sistema servidor

El sistema fue paralelizado con 4 procesos, todos ellos se comunican a través de colas de datos compartidas provistas por la librería "multiprocessing" de Python.

El primer proceso generado es el encargado de realizar la decodificación del streaming enviado por el dron. Esta tarea requiere alta capacidad de cómputo, ya a que será esta acción la que limite la velocidad de procesamiento del sistema (es necesario obtener la imagen decodificada para poder procesar y

planificar); este proceso se denomina Streamer. Por otro lado, existe un proceso Planner, encargado de la detección de las figuras, la planificación de caminos y la transformación a movimientos. El proceso Planner requerirá una imagen mediante la correspondiente cola al Streamer; obtenida la imagen, detecta las figuras geométricas, planifica la ruta y calcula el movimiento a realizar por el vehículo. A su vez, este proceso posee un hilo encargado de dibujar los contornos y la trayectoria a realizar en una nueva imagen que será servida al tercer proceso (servidor WEB).

El servidor brinda una interfaz WEB donde recibe y envía datos desde/hacia el usuario a través de websockets.

Por último, se despliega un proceso encargado de sincronizar a estos últimos denominado SystemCoordinator.

La figura 6.29 muestra el esquema anteriormente mencionado, junto con las comunicaciones llevadas a cabo por parte de los procesos.

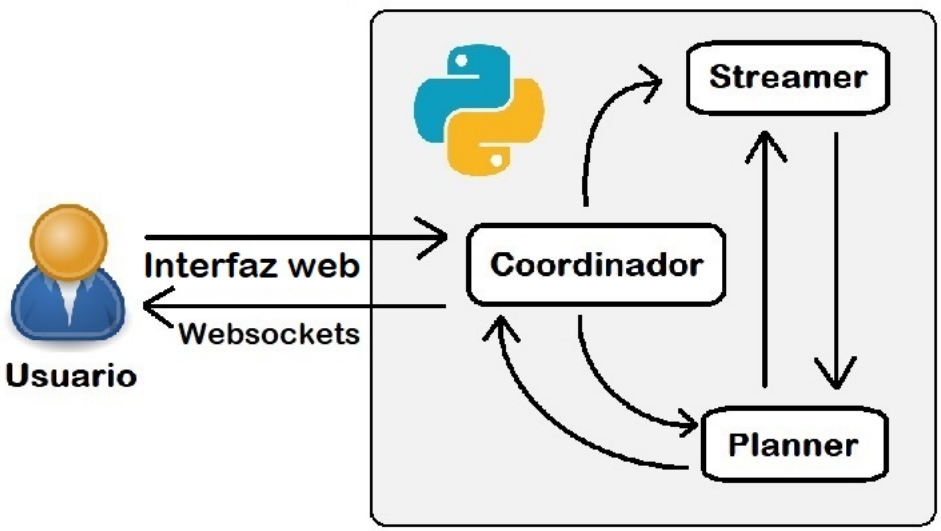

Figura 6.29: Comunicación entre procesos.

6.4.2.2. Subsistema de procesamiento de imágenes

Con respecto al procesamiento de las imágenes, se posee un módulo principal denominado "processor" que contiene métodos para ejecutar funciones relacionadas con tratamiento de imágenes, como compresión y descompresión, detección de componentes, obtención del tamaño de las imágenes, etc. Este módulo utiliza una instancia de la clase "ComponentsDetector" que es la encargada de encontrar las figuras geométricas correspondientes al Auto Robot, el destino y los obstáculos. La clase detector de componentes posee una instan-

cia de la clase “ShapeLabeler”, que dado un contorno (componente), retorna una cadena que identifica el tipo de figura geométrica. A su vez, “ComponentsDetector”, utiliza la clase “Filter” para encontrar los contornos en la imagen a través de diferentes técnicas de filtrado y máscaras aplicadas a la captura. Por otro lado, una vez encontrada la figura geométrica, la misma se representa en un objeto “Contour” que posee los puntos del contorno, las coordenadas del centro, su radio y sus puntos extremos, entre otras características identificatorias. La figura 6.30 muestra gráficamente el subsistema de procesamiento de imagen.

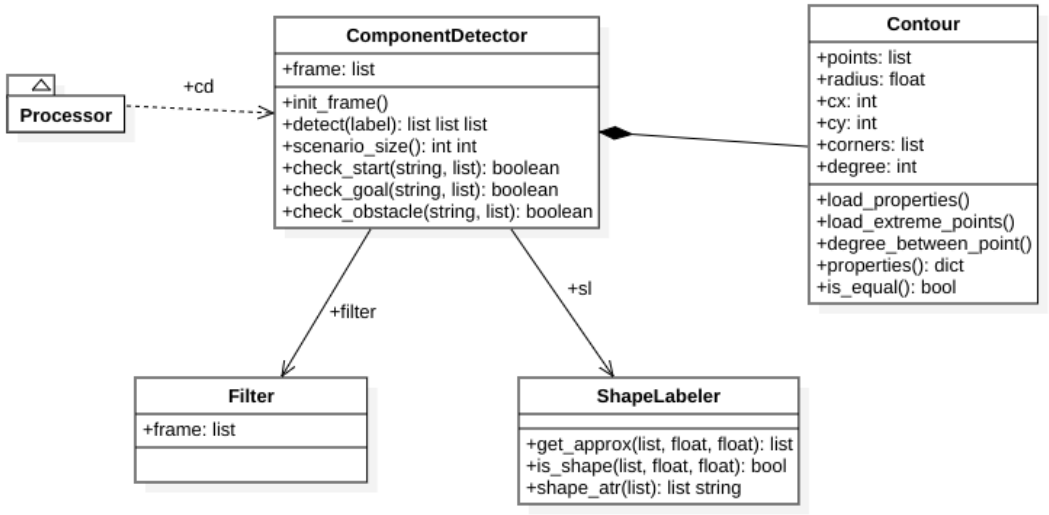

Figura 6.30: Diagrama del subsistema de procesamiento de imágenes.

6.4.2.3. Subsistema de planificación de caminos

En relación con la planificación de caminos, este subsistema consiste en un módulo encargado de la instanciación y ejecución del planificador seleccionado (“Planner”); permitiendo agregar diversos planificadores haciendo uso del patrón “Strategy”. Actualmente se programaron tres diferentes algoritmos de planificación, nombrados como “RRT”, “RRTStar” e “InformedRRTStar”, detallados en secciones posteriores.

Luego de la realización de la planificación, el subsistema genera una instancia de la clase “Path”, especificando todos los atributos que requiere el mismo (coordenadas de la planificación, coordenadas de la optimización, curva generada, etc). Por último, se calcula el movimiento a realizar por parte del Auto Robot.

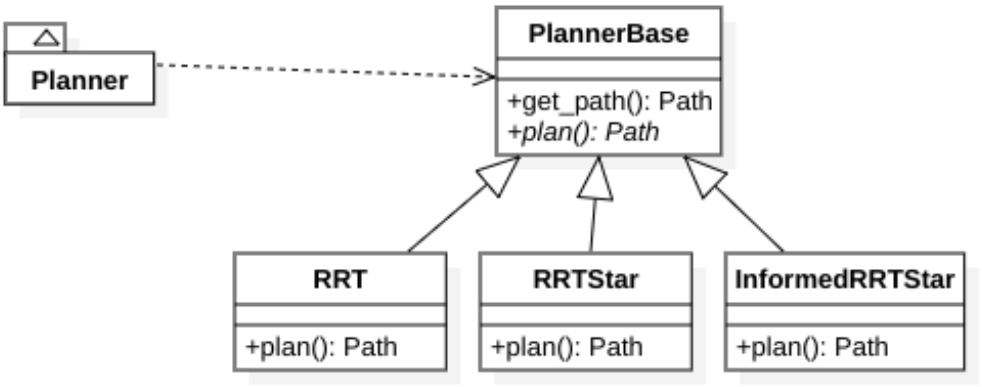

Figura 6.31: Diagrama del subsistema de planificación de caminos.

6.4.3. Video en tiempo real

El dron envía un streaming de video encodeado en H.264, por lo que se requiere la utilización de una tecnología que permita decodificar estas imágenes. Fue utilizado el programa "FFMPEG" para realizar la acción de decodificación; donde se debe especificar el archivo ".sdp" (figura 6.32) y los parámetros correspondientes que permitan mejorar el proceso de decodificación.

```
c=IN IP4 127.0.0.1
m=video 55004 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 packetization-mode=1
```

Figura 6.32: Archivo ".sdp".

6.4.4. Conclusiones

Se diseñó, desarrolló e implementó un sistema que contiene la algoritmia necesaria para detección de figuras geométricas dentro de imágenes, determinación y transformación de trayectorias y cálculo de movimientos a realizar, junto con la lógica que permite la coordinación y transmisión de datos e información en forma bidireccional con los robots, mediante un mecanismo sincrónico a través configuraciones especificadas en los archivos correspondientes. Por otro lado permite efectuar una transmisión de video hacia el Cloud en tiempo real

y desplegar una interfaz WEB que permite realizar experimentaciones mediante un simulador de planificación de caminos y gestionar la ejecución del sistema visualizando en todo momento las acciones realizadas y los resultados obtenidos.

Dentro de la instancia están incorporadas todas las funciones necesarias para el cumplimiento del sistema, delegando el poder de cómputo al Cloud, lo que genera que tanto el Auto Robot como el dron se responsabilicen de lo mínimo indispensable. Además, permite abstraer el sistema general, dando la posibilidad de reemplazar los actores del sistema Multi-Robots, sin condicionar el funcionamiento del mismo ni alterar el rendimiento.

6.5. Procesamiento de imágenes

En la presente tesina es deseable obtener información de las imágenes obtenidas por la cámara aérea, para esto se utilizaron técnicas de procesamiento e interpretación de imágenes. Fueron desarrollaros algoritmos en el lenguaje Python que trabajan principalmente en combinación con la librería OpenCV, y otras herramientas que permiten captar la información requerida y transformarla en datos requeridos para realización de diversas funciones del sistema.

El objetivo de este procesamiento detectar los elementos del escenario, es decir, encontrar en la imagen tanto el Auto Robot, como el destino y los obstáculos; y obtener las propiedades de los mismos (coordenadas del centro, radio, orientación, etc).

Los elementos nombrados anteriormente son identificados con figuras geométricas para simplificar el proceso de detección de los mismos. El destino es representado con un círculo, los obstáculos con un triángulo y el vehículo con un cuadrado o rectángulo con un círculo interior.

6.5.1. Funcionamiento

Se procede a describir el funcionamiento del procesamiento de una imagen:

- Al recibir una imagen, se aplica un primer algoritmo de procesamiento, que mejora la imagen y la transforma, detectando los contornos e identificando las figuras según sea posible.
- Los elementos detectados son almacenados y se vuelve al paso anterior con otra configuración de mejoramiento de imagen. En caso de haber encontrado y guardado todos los elementos fundamentales (es decir, se encontró tanto el Auto Robot como el círculo), finaliza el proceso.

- Por otro lado, si se determina que aún no se ha podido encontrar el vehículo o el destino, se procede a aplicar nuevos algoritmos de procesamiento, pero esta vez utilizando una serie de mejoramientos de imágenes mucho más demandantes en cuestiones de computo. Estos filtros se utilizan para obtener mejores resultados en la detección implicando a su vez un mayor tiempo de ejecución. Una vez encontrados los elementos necesarios en el escenario, finaliza este proceso.

- En caso de no haber encontrado todas las figuras a esta altura, se captura otra imagen del streaming de video y se repite todo el proceso, descartando las figuras previamente identificadas.

6.5.2. Mejoramiento de la imagen

Una vez obtenido la imagen del streaming de video (figura 6.33), la misma se convierte escala de grises (figura 6.34) y se procede a aplicar una serie de filtros con tal de facilitar la detección de las figuras.

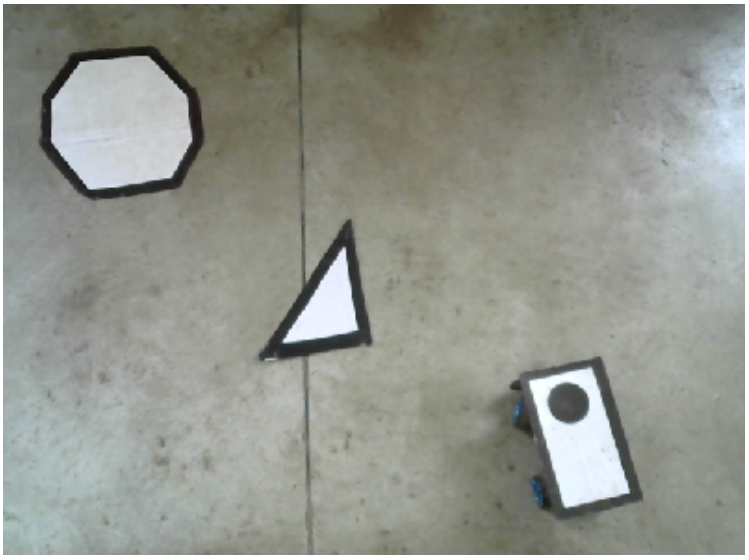

Figura 6.33: Imagen original

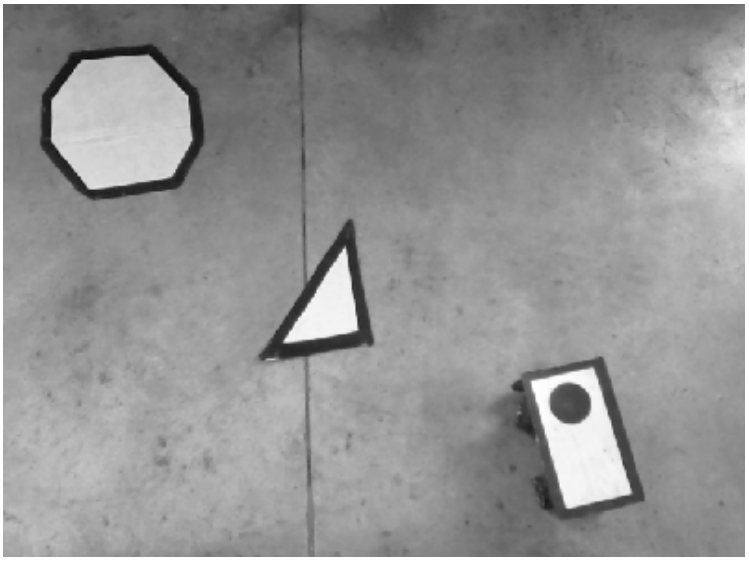

Figura 6.34: Imagen en escala de grises

Fueron implementados diferentes combinaciones de filtros los cuales son utilizados por el algoritmo de detección, que intenta encontrar las figuras con distintas imágenes mejoradas. Esto tiene como objetivo ampliar las posibles variaciones de luz que afectan al escenario logrando una detección mucho más eficaz en distintos ambientes.

Los filtros principales que se utilizaron fueron: "GaussianBlur" como suavizado de imagen, "AdaptativeThreshold" y "CannyEdge" para resaltar contornos. Estos algoritmos son provistos por la librería OpenCV.

Como se mencionó en el apartado de funcionamiento, se implementaron dos grupos de filtros: unos algoritmos simples y de rápida transformación y otros que precisan un gasto computacional elevado y por consiguiente requieren mayor tiempo de ejecución que los anteriores. La razón de esta división está dada debido a que en un sistema en tiempo real, es indispensable obtener respuestas velozmente, por lo que es imprescindible que la etapa de procesamiento de imágenes sea lo más eficiente posible.

Este primer grupo de filtros simples logra resolver el mejoramiento de imagen en menos de 10 milisegundos. Además, es importante remarcar que retorna imágenes con pocos contornos (entre 50 y 500 dependiendo de la imagen capturada) lo que implica menos información para procesar.

En caso de no lograr encontrar la información requerida, se implementó un grupo de filtros que precisa un mayor tiempo de cómputo para transformar la imagen y que además devuelven capturas con gran cantidad de contornos (aproximadamente 5000). El objetivo de estos filtros es encontrar aquellas figuras que los filtros más simples no detectaron.

Para los algoritmos simples se utilizaron tres diferentes combinaciones de filtros la cuales aplican en primera instancia un suavizado (figura 6.35) con máscaras de tamaño tres, siete y cinco aplicando un resaltado de contornos "CannyEdge" al final; en el caso del último suavizado se agrega además el filtro "AdaptativeThreshold".

Se presentan estos filtros en las figuras 6.36, 6.37, las cuales utilizan los algoritmos "Canny Edge" para la primer figura, y en la segunda son aplicados primeramente el algoritmo "Canny Edge" y sobre esa transformación se aplica el filtro "AdaptativeThreshold".

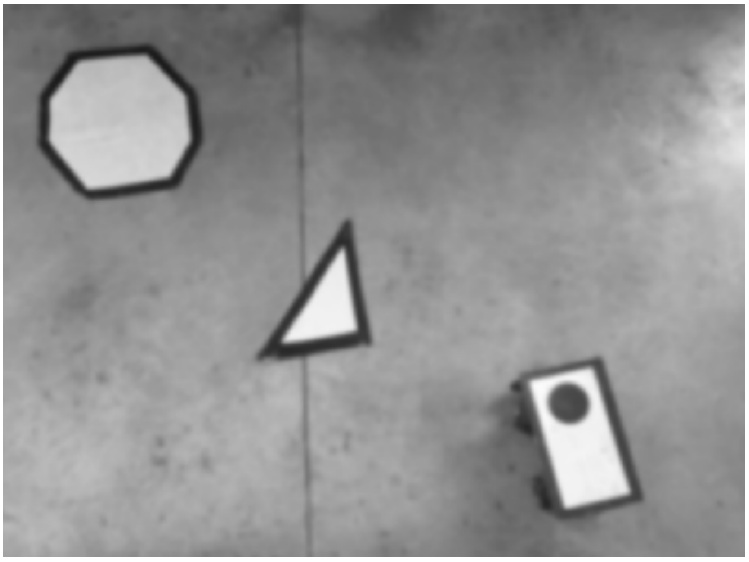

Figura 6.35: Imagen con suavizado Gaussiano.

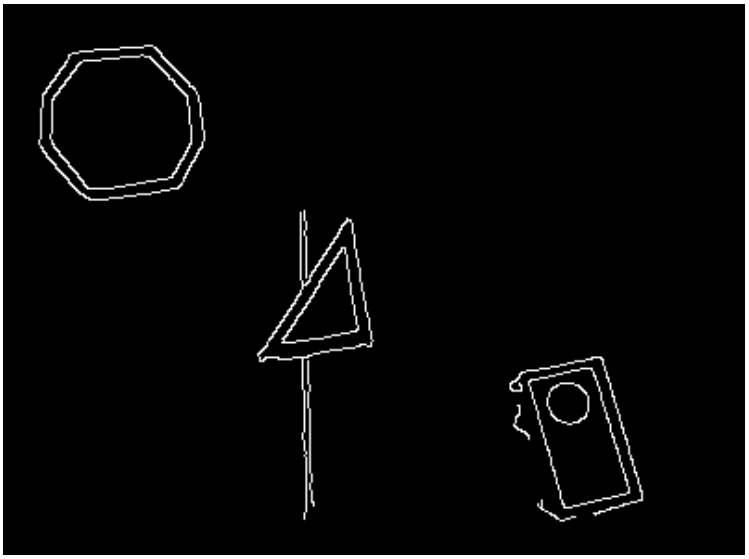

Figura 6.36: Filtro simple con “Canny Edge”

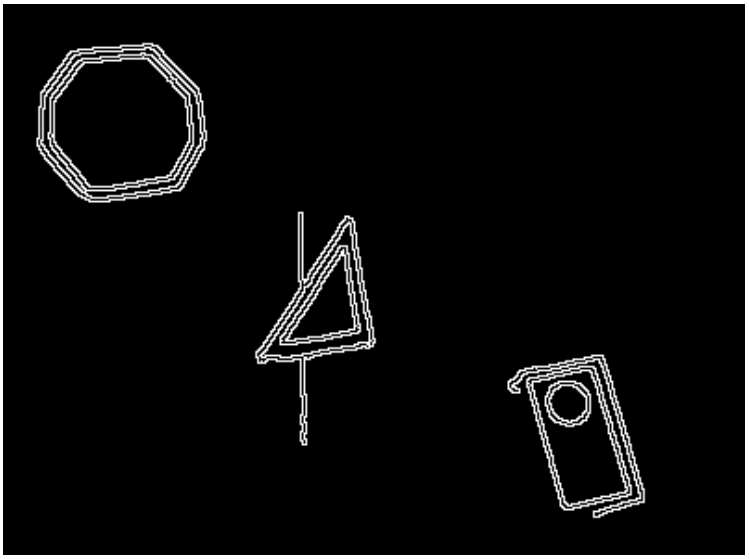

Figura 6.37: Filtro simple con “Canny Edge” y “AdaptativeThreshold”

Por último, para los filtros más pesados se utilizaron principalmente combinaciones de suavizados gaussianos y umbrales adaptativos utilizando diferentes máscaras (ejemplo en la figura 6.38).

Toda esta funcionalidad está contenida en una clase “Filter”. Esta se describe junto con dos combinaciones de filtros en el anexo 3.

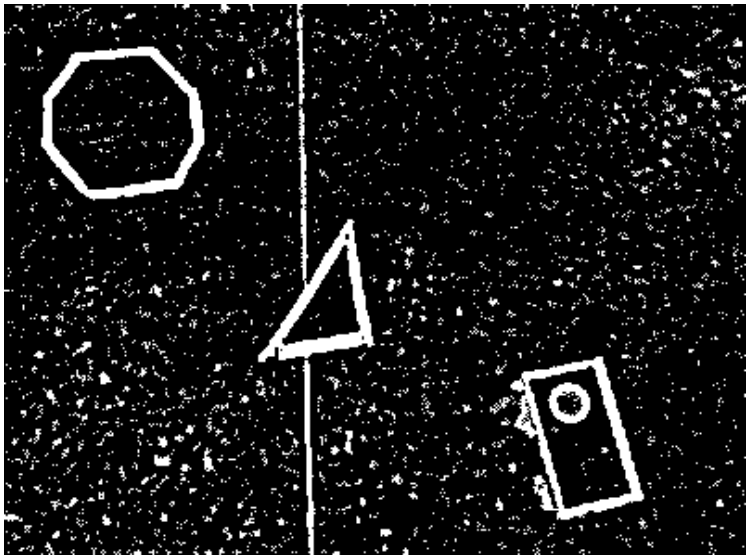

Figura 6.38: Imagen con filtro “Threshold”

6.5.3. Detección de contornos

Una vez aplicado alguna de las combinaciones de los filtros, se procede a detectar los contornos. Para esto, se ejecuta a la función de OpenCV “cv2.findContours()” la cual retorna un arreglo con los contornos que encontró y otro arreglo diferente que especifica la relación que tiene los mismos entre sí. Los parámetros de esta función son: la imagen mejorada a procesar, la forma en que se presentan y describen las relaciones y la forma de detección de contornos. En este trabajo son relevantes los dos primeros parámetros. En el caso del segundo argumento, se especifica obtener estas relaciones en forma de árbol con la directiva “cv2.RETR_TREE” ya que esta describe las relaciones de padre-hijo y hermanos necesarias para el cálculo del ángulo del robot. Estas relaciones se pueden entender de la siguiente manera: un contorno dentro de otro es hijo del segundo, varios contornos dentro de otro son hermanos entre sí.

En la figura 6.39 se puede observar un ejemplo de jerarquía entre contornos

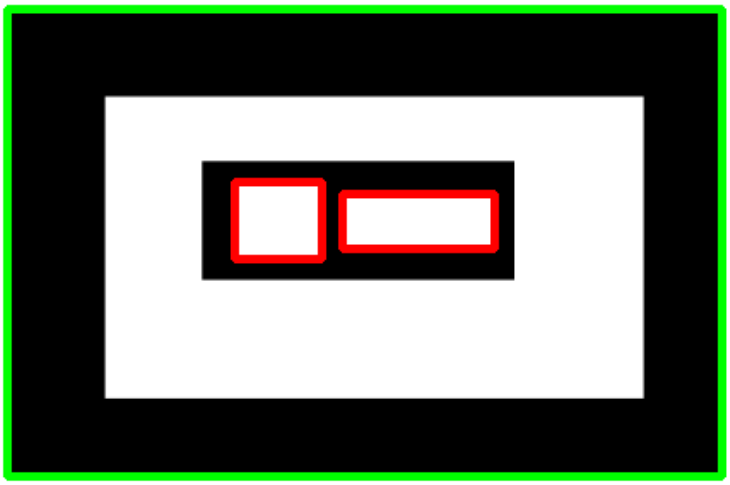

Figura 6.39: Ejemplo de jerarquía entre contornos

6.5.4. Identificación de figuras

En esta etapa, se itera sobre todos los contornos detectados verificando que cumplan ciertas condiciones para poder evaluar el tipo de figura que representan, calculando de antemano ciertos valores. Pasada la verificación, se identifica la figura en base a la cantidad de lados que tiene el contorno. En caso de tener tres lados, se considera un triángulo, cuatro lados es un cuadrado o rectángulo y más de 5 es identificado el componente como un círculo. Este comportamiento se modela en la clase "ShapeLabeler" la cual se describe en el anexo 4.

Luego, dependiendo del tipo de figura detectado se procede de distintas maneras:

- Triángulo: en caso de identificar un triángulo, se considera haber detectado un obstáculo. Este se agrega a una lista de obstáculos verificando que no haya sido agregado anteriormente.

- Cuadrado/Rectángulo: si se detectó un cuadrado o rectángulo, se verifica que éste represente al robot. Para esto se intenta detectar un círculo pequeño dentro de este contorno (haciendo uso de la jerarquía de contornos). Esta verificación se explica en el anexo 5.

  En la figura 6.40 se muestra la figura utilizada para representar al robot. El lado donde se ubica el círculo indica la parte delantera del robot.

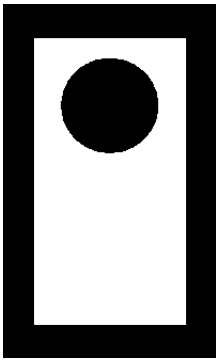

Figura 6.40: Figura de forma rectangular con un círculo interior en un extremo. Representa al robot.

- Círculo: si el contorno detectado se identifica como círculo, este se considera como destino a utilizar. Además, se verifica que el centro de este difiera del centro del circulo interior del robot para evitar confundir los componentes.

En la siguiente figura, se observan los contornos detectados e identificados, resaltados en diferentes colores según corresponda a cada elemento del escenario.

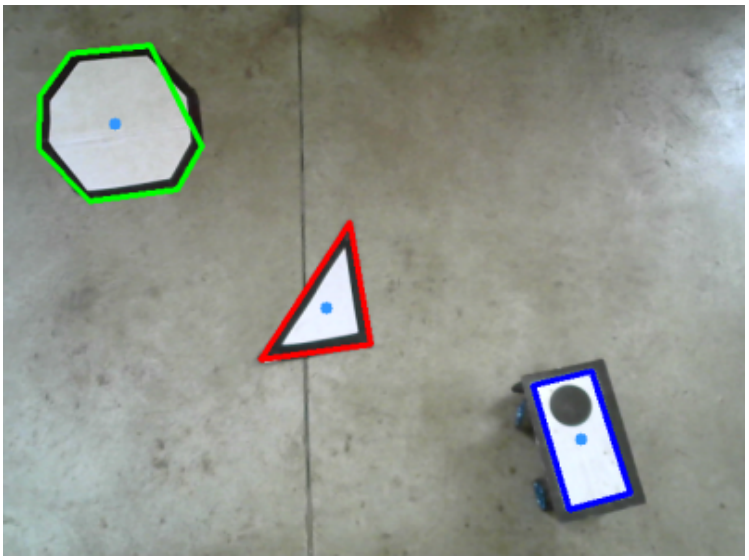

Figura 6.41: Detección e identificación de contornos.

6.5.5. Cálculo de ángulo del robot

En el momento que se detecta e identifica al robot, es calculada su posición angular utilizando una función matemática, que dado dos puntos determina el ángulo generado entre estos. Para eso, se utiliza las coordenadas céntricas del cuadrado detectado y las del círculo interior del mismo. El ángulo resultante sirve para conocer la orientación del robot. En la figura 6.42 se muestra, de forma gráfica, la detección del rectángulo, círculo interior y los puntos utilizados para el cálculo del ángulo además del ángulo en el que se encuentra el robot.

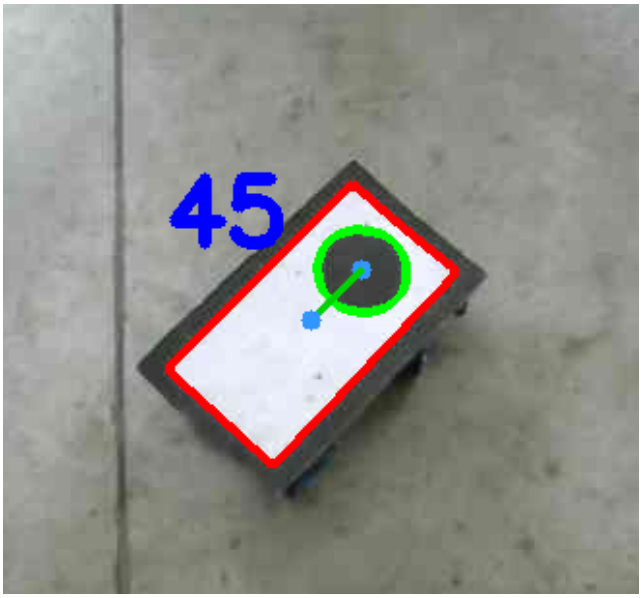

Figura 6.42: Representación de los datos utilizados para el cálculo de grado.

6.5.6. Conclusiones

Se desarrollaron algoritmos de mejoramiento de imagen capaces de presentar diferentes alternativas con el fin de poder detectar la mayor cantidad de contornos por imagen. Por otro lado, se implementó una solución para identificar figuras geométricas las cuales representan elementos reales del escenario, permitiendo conocer la posición y dimensión del robot, obstáculos y destino. Además, se logró obtener la orientación en que se encuentra el robot en una imagen a partir del cálculo del ángulo entre elementos geométricos de la figura que lo representa.

En conclusión, se desarrolló un subsistema capaz de obtener información del escenario e interpretarla con el fin de generar datos consumidos por otros

módulos del sistema.

6.6. Planificación de caminos y cálculo de movimientos

6.6.1. Introducción

La planificación de caminos es un tema fundamental en la presente tesina. La determinación de trayectorias consiste en encontrar aquella ruta que permita al Auto Robot desplazarse desde su posición inicial hasta llegar al destino, sorteando los obstáculos presentes en el escenario. Para lograr este objetivo, es necesario utilizar un algoritmo que detecte la ruta a realizar y un método de transformación y conversión que transforme dicha ruta en movimientos entendibles por el Auto Robot.

Fue implementado el algoritmo RRT para detección de caminos; como se explicará, requiere de algunos parámetros físicos del robot, como también precisa la obtención de diferentes datos de la imagen. Una vez encontrada la ruta que conecta la configuración inicial con la meta, se procede a un paso de optimización de la misma, que consiste en recorrer el conjunto de nodos obtenidos, aplicando una función matemática que modifica el camino removiendo del conjunto aquellos puntos intermedios que no intersectan con los obstáculos. Conseguida la optimización de la ruta, se aplica a la misma un método de suavizado, que permite unir los diferentes puntos del camino para generar una trayectoria más fácil de transitar.

Por último, se aplica la matemática necesaria para transformar la ruta obtenida en movimientos de referencia que el robot pueda ejecutar, para esto se precisa calcular la orientación y el sentido de desplazamiento a alterar.

El sistema de planificación desarrollado permite en primera instancia, encontrar el camino óptimo que conecta el inicio con la meta definida. Por otro lado, se permite también ajustar la trayectoria utilizando como referencia una ruta previa, obteniendo de esta forma una planificación de caminos eficiente y rápida; requisito fundamental para la realización del presente trabajo, debido a la necesidad de calcular en tiempo real la trayectoria a realizar.

En las siguientes subsecciones, se presentarán diversas simulaciones a modo de ejemplo. En cada una de ellas, el círculo azul representa al vehículo, el círculo verde al destino y los círculos rojos a los obstáculos.

6.6.2. Algoritmo RRT

Como algoritmo de planificación de caminos se escogió la utilización del método de generación aleatoria, específicamente el algoritmo RRT, desarrollado en el lenguaje Python 3.7. En este apartado se explicará la implementación realizada del método RRT básico.

El algoritmo RRT es instanciado de la siguiente manera:

```
1 rrt = RRT(init, goal, obstacles, area_x, area_y, expand_dist,
      goal_dist, max_time)
```

Como entradas tiene el inicio (es decir, el Auto Robot), con las propiedades de coordenadas céntricas (cx, cy) y radio en pixeles. Asimismo, se envía las propiedades del destino y una lista con las propiedades de los obstáculos. Estos datos son obtenidos al realizar la detección de los objetos explicado en la sección anterior. A su vez, recibe el área del escenario [0, columnas] y [0, filas]. Como se explicó en el subapartado RRT de planificación de caminos en la sección de introducción del presente trabajo, el algoritmo RRT precisa que se especifique la distancia entre los nodos en pixeles, el argumento "expand_dist" representa a esta variable. El parámetro "goal_dist" determina la distancia mínima entre el último nodo del árbol RRT y el destino, para comprobar si se llegó a la meta. Por último, el argumento "max_time" identifica el tiempo máximo que se tiene hasta encontrar el camino.

Posteriormente, comienza la ejecución del algoritmo con la llamada al método:

```
1 path = rrt.plan()
```

El método plan retornará una lista con los nodos involucrados en el camino que representan el árbol, como así también todas las iteraciones que fueron necesarias realizar para encontrar el destino, con el fin de poder graficar la búsqueda en su completitud. En el anexo 6 se muestra el código completo, en este apartado serán detalladas y explicadas las diferentes partes del mismo.

En primera medida, se inicializan tres diferentes variables:

- goal_flag: utilizada para comprobar si se ha encontrado el destino.
- iterations: lista con todos los nodos consultados en la búsqueda.
- node_list: lista con aquellos los nodos que forman el camino. Como se puede observar en el código siguiente, el primer nodo hace referencia a las coordenadas de inicio.

```
goal_flag = False
iterations = []
self.node_list = [self.start]
```

Una vez inicializadas las variables se comienza con la iteración del algoritmo, para esto se mide el tiempo actual y se planificará mientras la diferencia entre el tiempo de comienzo y el actual no sea mayor al límite, como muestra la instrucción siguiente:

```
start_time = time.time()
while (time.time() - start) < self.max_time:
```

Por cada iteración, lo primero a realizar es la búsqueda aleatoria; para esto, se busca un número random en el área de trabajo de la siguiente forma:

```
if random.randint(0, 100) > self.goal_sample_rate:
    rnd = [random.uniform(self.rand_x[0], self.rand_x[1]),
    random.uniform(self.rand_y[0], self.rand_y[1])]
else:
    rnd = [self.goal.x, self.goal.y]
```

Obtenida la coordenada aleatoria, se procede a conseguir el índice del nodo en esa posición y por consiguiente, busca el nodo más cercano:

```
nind = self.get_nearest_list_index(self.node_list, rnd)
nearest_node = self.node_list[nind]
```

El paso siguiente es calcular el nuevo nodo, para esto, se realiza una copia del nodo más cercano en una variable que representa al nuevo nodo, se calcula el theta y se adhiere a las coordenadas x e y la distancia correspondiente, como muestra el siguiente código:

```
nearest_node = self.node_list[nind]
theta = math.atan2(rnd[1] - nearest_node.y, rnd[0] - nearest_node.
    x)

new_node = copy.deepcopy(nearest_node)
new_node.x += self.expand_dis * math.cos(theta)
new_node.y += self.expand_dis * math.sin(theta)
```

Encontrado el nuevo nodo, resta agregar a su padre. Se recuerda que un nodo del árbol RRT, puede a lo sumo tener un padre pero contener varios hijos:

```
new_node.parent = nind
```

Como se expresó anteriormente, es necesario guardar todos los nodos que fueron accedidos en la búsqueda del destino con el fin de poder graficar las iteraciones realizadas, para esto se ejecuta la siguiente instrucción:

```
iterations.append([nearest_node, new_node])
```

Luego de la creación del nuevo nodo, se procede a verificar si el mismo colisiona con algún obstáculo. Cabe aclarar que el rango de colisión está dado por el radio del obstáculo sumado al radio del Auto Robot (radOb + radRob). Si el nuevo nodo colisiona con un obstáculo, se prosigue a la próxima iteración:

```
if not self.collision_check(new_node):
    continue
```

En caso de que el nuevo nodo no colisione con ningún obstáculo, estamos ante un punto que es parte del camino, por lo que se debe agregar al árbol resultante:

```
self.node_list.append(new_node)
```

Por último, se deberá verificar si el nuevo nodo es en realidad el último, es decir, si está a una distancia mínima de la especificada en el momento de la instanciación del algoritmo RRT. Si esto se cumple, finaliza la iteración del algoritmo, actualizando la bandera de alcance de destino en verdadero:

```
dx = new_node.x - self.goal.x
dy = new_node.y - self.goal.y
d = math.sqrt(dx**2 + dy**2)
if d <= self.expand_dis:
    goal_flag = True
    break
```

La figura 6.43 y 6.44, muestra aquellos nodos accedidos en la ejecución del algoritmo y se puede observar en verde, la trayectoria final que conecta el inicio con el destino, evitando los obstáculos. En el primer ejemplo, fue encontrado el camino en aproximadamente 1,09 segundos, mientras que en el segundo ejemplo fueron necesarios 1,96 segundos, en un entorno de 1313x641

pixeles. Estos experimentos fueron realizados utilizando el simulador WEB desarrollado, explicado en el apartado de interfaz WEB.

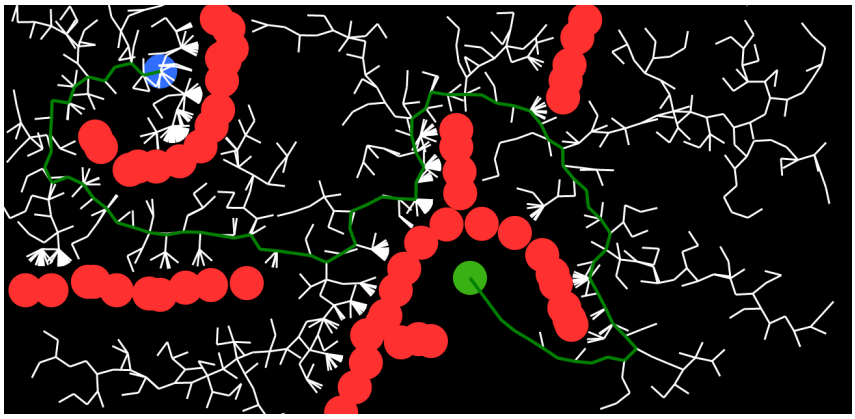

Figura 6.43: Ejemplo 1: de planificación RRT

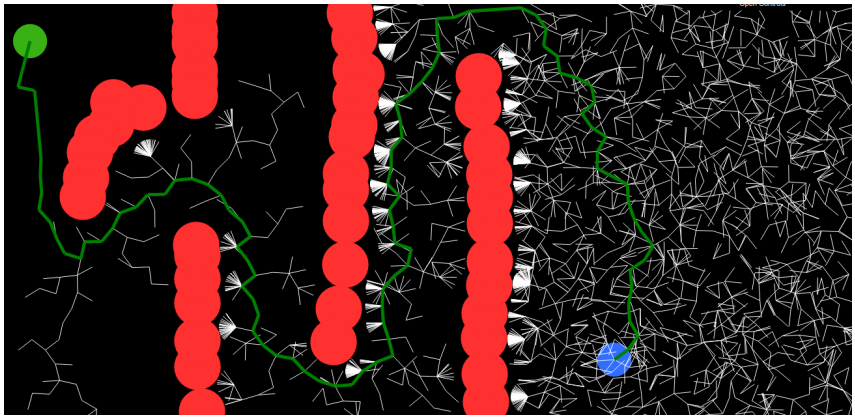

Figura 6.44: Ejemplo 2: de planificación RRT

6.6.3. Optimización del camino

El algoritmo RRT, como se mencionó, retorna un árbol que programáticamente hablando, es una lista de puntos (x, y). Como se puede apreciar en las figuras 6.43 y 6.44, el camino resultante cuenta con una gran variedad de desvíos innecesarios, lo que repercutirá en un mayor esfuerzo por parte del Auto Robot a la hora de desplazarse.

Para solucionar este problema, se propone un método de optimización del camino, que consiste en crear conexiones lo más longitudinales posibles entre

dos nodos extremos, removiendo como consecuencia los puntos que no forman parte de esa conectividad. La búsqueda del nodo más lejano es realizada generando una búsqueda dicotómica en la lista de coordenadas.

Con este objetivo, en primer lugar se indican diferentes variables:

- new_coords: la nueva lista de nodos resultante.
- lid: último índice de la nueva lista de nodos.
- i: índice actual de la lista de nodos.
- p: el nodo actual.

Inicializa la nueva lista de coordenadas, como así también el índice:

```
new_coords = set()
lid = 0
```

Comienza la iteración recorriendo en forma completa la lista de nodos, nombrada como "coords". Primeramente, es obtenido (el nodo en la posición lid de la lista de coordenadas) y se inserta al mismo en la nueva lista:

```
while lid < len(coords):
    p = coords[lid]
    new_coords.add(tuple(p))
```

Luego, se itera por todos los nodos de la lista en forma binaria, consultando si es posible realizar una conexión entre "p" (nodo actual) y "coords[aux]" (nodo a conectar). La conectividad se logra comprobando si existe intersección con un obstáculo en medio de una línea recta entre los dos nodos. Esta línea tiene un grosor igual al radio del Auto Robot. Si no hay colisión con ningún obstáculo, se almacena el índice del nodo a conectar. Para cualquier caso, se avanza al siguiente nodo de la lista:

```
lid+=1
first = lid
last = len(coords)
while first < last:
    med = (first + last) // 2
    line = geo.LineString((p, coords[med])).buffer(radius)
    cond = [o.intersects(line) for o in obstacles]
    if not True in cond:
        lid = med
        first = med + 1
    else:
        last = med-1
```

Una vez finalizada la iteración, la variable “lid” contiene el último índice del nodo al cual se puede conectar sin colisionar con los obstáculos, por consiguiente se retorna al comienzo de la iteración, agregando “p” a la nueva lista de coordenadas.

El resultado de esta operación consiste en una reducción de los nodos del árbol RRT, posibilitando, como muestra la figura 6.45 y 6.46, generar una trayectoria con una menor cantidad de desvíos, lo que facilita el traslado del Auto Robot.

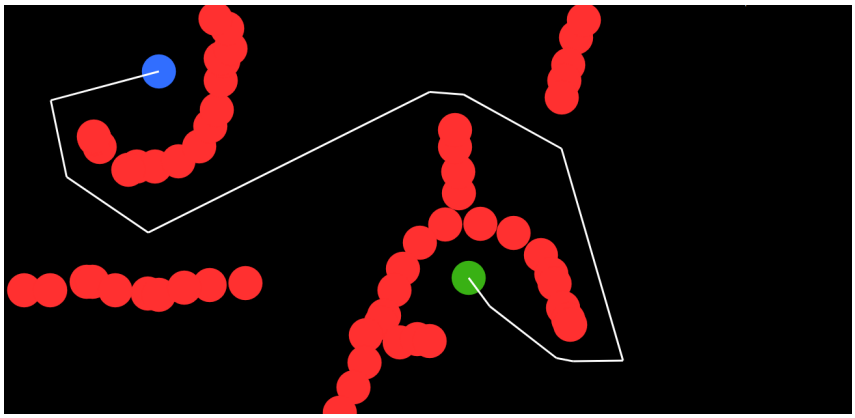

Figura 6.45: Ejemplo 1: camino optimizado

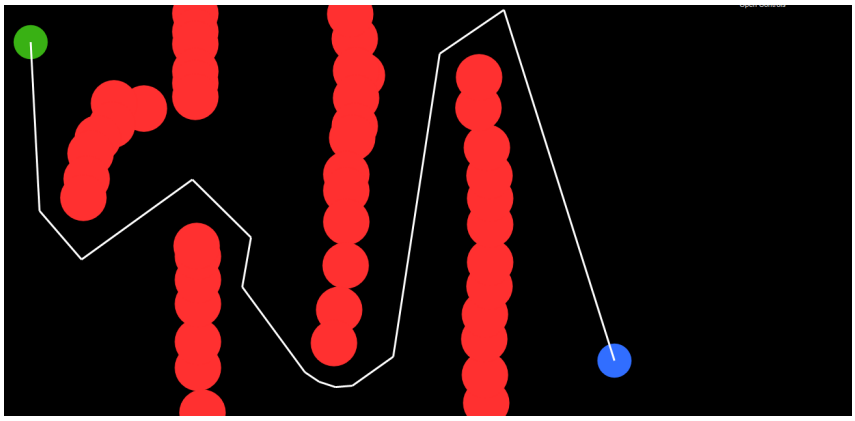

Figura 6.46: Ejemplo 2: camino optimizado

6.6.4. Suavizado del camino

Como puede apreciarse, si bien la ruta obtenida en el paso anterior contiene una menor cantidad de inflexiones, es evidente que el Auto Robot no podrá

cumplir con exactitud la trayectoria, como consecuencia de los desvíos bruscos generados en la unión de las diferentes partes del camino.

Para solventar este inconveniente, se precisa aplicar un nuevo método que consiste en suavizar el camino, de forma tal que permita al Auto Robot acomodarse en todo momento, y no tener que orientar una gran cantidad de grados a última instancia.

Con el fin de lograr el suavizado, fue utilizada una función matemática denominada interpolación cúbica (o del inglés "cubic spline"); dado un conjunto de puntos, retorna una nueva colección de puntos que forman parte de la nueva curva (figura 6.47). El algoritmo utilizado permite establecer la distancia en la que debe estar cada uno de los puntos de la curva.

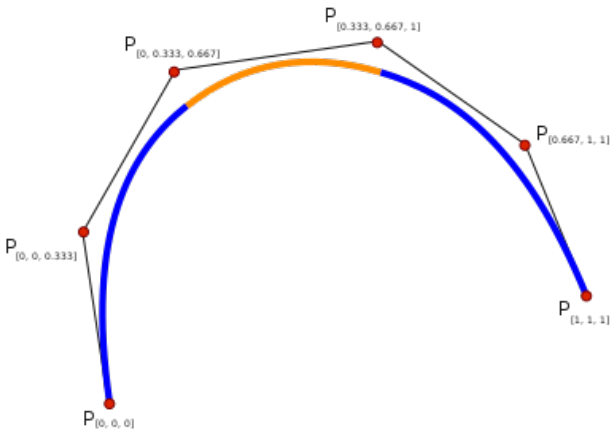

Figura 6.47: Ejemplo 1: camino optimizado

Para dar comienzo con el proceso de suavizado, se ejecuta un método de la clase "CubicSpline" de la siguiente forma:

```
cubic_spline(coords, ds=px_speed)
```

Los dos argumentos enviados hacen referencia a las coordenadas de los puntos resultantes luego de la optimización y a la velocidad en pixeles a la que se está desplazando el vehículo (diferencia entre la coordenada céntrica del vehículo y el centro del mismo en la imagen anterior)

Posteriormente, se separan los x e y del arreglo de coordenadas y se obtiene la función de interpolación cúbica (retornada por la clase "Spline2D"), como presenta a continuación:

```
x, y = list(zip(*coords))
sp = Spline2D(x, y)
```

Se genera una lista de puntos, en donde cada punto está separado por la distancia previamente definida (“ds”):

```
s = list(np.arange(0, sp.s[-1], ds)
```

Luego, se procede a inicializar dos variables referentes a las nuevas coordenadas del camino (“coords”) y a la nueva lista de grados (“dyaw”) para determinar efectivamente cada grado de la curva generada:

```
coords, dyaw = [], []
for i_s in s:
    coords.append(sp.calc_position(i_s))
    dyaw.append(yaw_to_degrees(sp.calc_yaw(i_s)))
```

Por último, retorna tanto las coordenadas resultantes como también la lista de grados.

Como fue expresado, el resultado final consiste en una trayectoria suavizada, que permite poder guiar al Auto Robot de forma más precisa (figura 6.48 y 6.49).

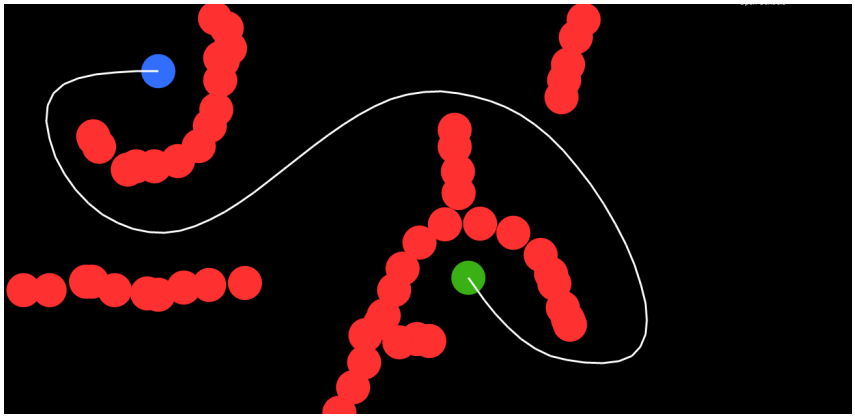

Figura 6.48: Ejemplo 1: trayectoria suavizada

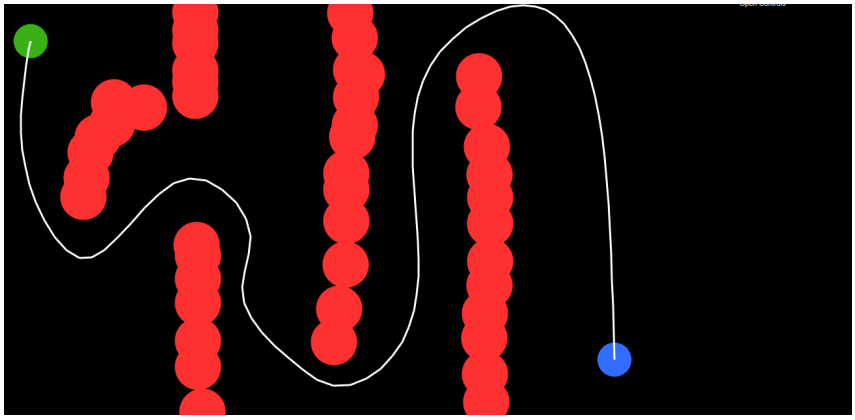

Figura 6.49: Ejemplo 2: trayectoria suavizada

6.6.5. Replanificación

En las figuras 6.43 y 6.44 fueron planteados dos ejemplos de diferentes planificaciones RRT, obteniendo la ruta a transitar en el primer caso en 1 segundo aproximadamente y 2 segundos para el otro. En un sistema en tiempo real como el de la presente tesina es necesario localizar la ruta en un menor tiempo.

Cuando comienza el sistema, el vehículo se encuentra estacionado a la espera del envío de un movimiento por parte del Cloud. En esta fase, no es necesario computar la trayectoria de forma rápida, por lo que se aprovecha para obtener la ruta óptima entre todas las posibles. Con el propósito de encontrar el camino óptimo, se recalcula la planificación RRT tantas veces sea posible en un tiempo finito (por ejemplo, 10 segundos). Cada vez que finaliza una planificación, se comprueba si el camino obtenido es más acotado que el camino óptimo en ese momento; de ser así, este último es actualizado.

Luego de obtener la ruta óptima, comienza el desplazamiento del vehículo, por lo que inicia la fase de replanificación de la trayectoria en tiempo real.

La replanificación consiste en tomar como base aquellos puntos generados en la optimización del camino óptimo, e iterar los mismos de a pares, consultando si existe una colisión entre estos dos puntos:

```
for p1, p2 in zip(last_plan, last_plan[1:]):
    line = geo.LineString((p1, p2)).buffer(c.robot.radius)
    cond = [o.intersects(line) for o in obstacles]
    if True in cond:
        ...
```

En caso que no haya un obstáculo, son almacenados tanto “p1” como “p2” en los puntos del nuevo camino. Por el contrario, si existe peligro de colisión entre los dos puntos, se procede a realizar la planificación RRT, pero en este caso entre dos nodos cercanos, esto quiere decir que la replanificación se realizará en un corto tiempo, sin afectar el rendimiento del sistema.

Se presenta el siguiente ejemplo: en el escenario está presente tanto el vehículo (círculo azul), el destino (círculo verde) y los obstáculos (círculos rojos), distribuidos estos últimos de forma tal que dificulte la detección del camino. En primera medida se realiza una primera planificación RRT (figura 6.50), con un tiempo aproximadamente de 5,93 segundos. Luego se procede a reaplanificar el camino, como puede observarse en la figura 6.51, el vehículo avanzo unos pocos pixeles, pero no existe ningún obstáculo que atenta contra el camino planificado anteriormente, por lo que no es necesario realizar ningún tipo de planificación, obteniendo el camino final en 0,02 segundos (20 milisegundos). El vehículo siguió avanzando y, como muestra la figura 6.52, se insertaron 2 obstáculos nuevos, por lo que se deberá replanificar el camino en esos tramos, obteniendo el camino final en un tiempo de 0,07 segundos (70 milisegundos), debido a que únicamente se replanificó aquellos tramos donde se introdujeron los obstáculos. En la figura 6.53 puede observarse el resultado de la última planificación.

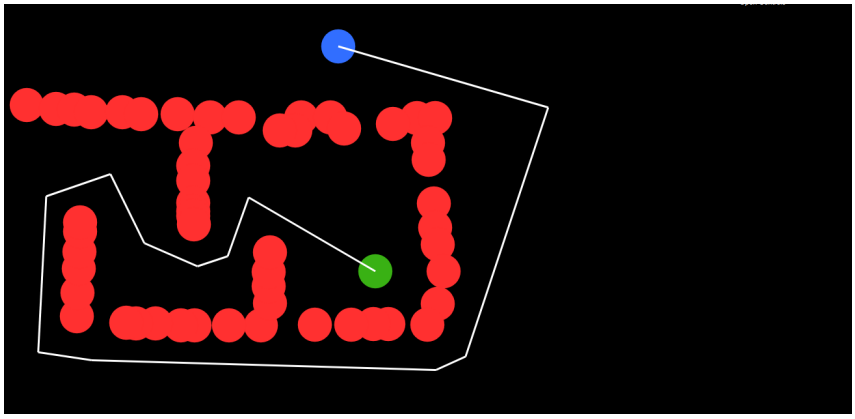

Figura 6.50: Ejemplo replanificación: planificación completa

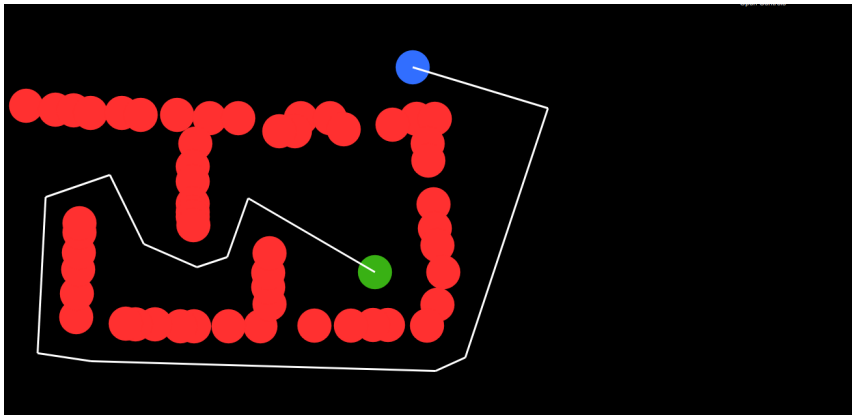

Figura 6.51: Ejemplo replanificación: primer replanificación

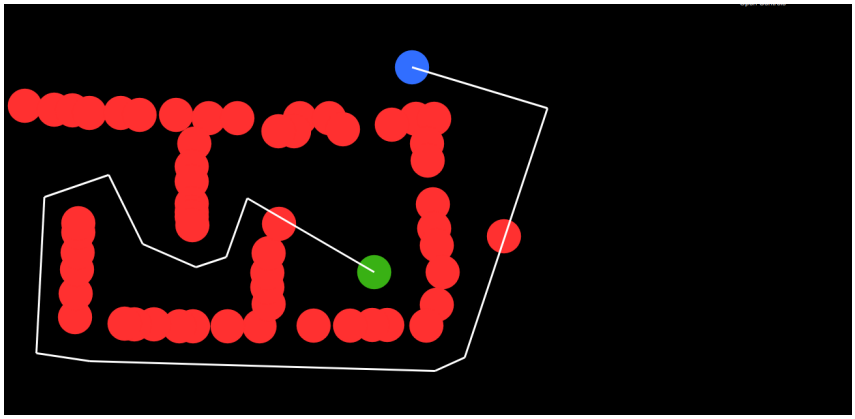

Figura 6.52: Ejemplo replanificación: segunda replanificación con nuevos obstáculos

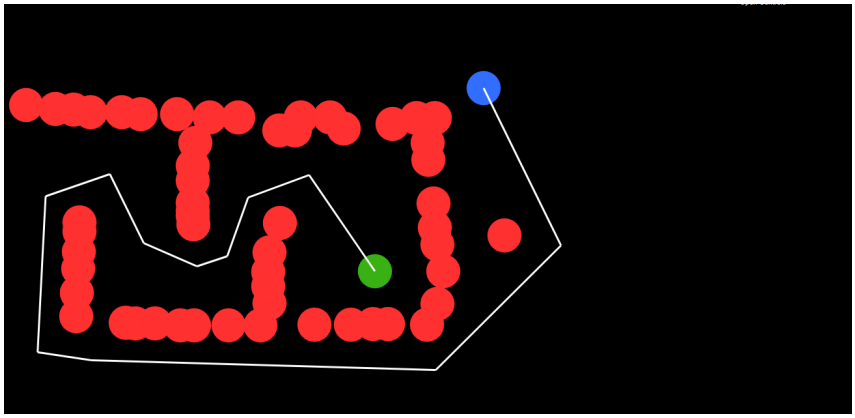

Figura 6.53: Ejemplo replanificación: resultado final

6.6.6. Cálculo de la maniobra

Luego de encontrar la ruta a realizar y su posterior transformación, se procede al cálculo del movimiento que el vehículo deberá ejecutar. El movimiento a realizar consta de un grado de giro ([-180°, 180°]) y un sentido de desplazamiento (1 hacia delante, -1 hacia atrás, 0 freno).

Para lograr este objetivo, se diseñó un método de percepción del área transitable. Este método permite obtener la información necesaria para detectar y actuar ante la posible colisión del vehículo con un obstáculo, o cuando el mismo se encuentra próximo a los límites del escenario.

Se planteó una técnica que consiste en incorporar de forma lógica un paralelogramo en los límites del vehículo. Este polígono abarca el área de movimiento del Auto Robot, por lo que se requiere conocer el ángulo de giro máximo que posee el vehículo (este dato es enviado por el Auto Robot en la interacción inicial previo al comienzo de la ejecución).

La figura 6.54 muestra en forma gráfica, un ejemplo del paralelogramo para el vehículo utilizado, el mismo se encuentra a 0 grados.

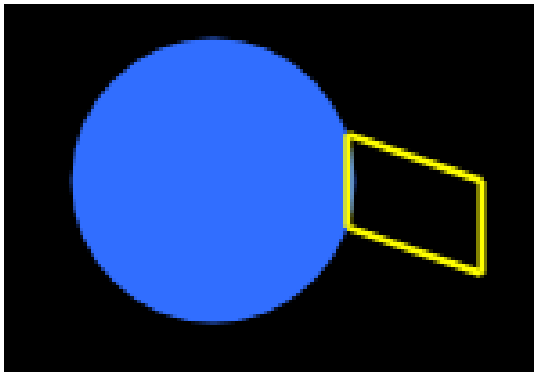

Figura 6.54: Ejemplo de un paralelogramo

La apertura del polígono está dada tanto por el ángulo de dirección máximo, como también por la diferencia angular entre la posición actual del vehículo y la del camino a alcanzar; mientras que la longitud del mismo la determina la velocidad en pixeles del vehículo. Esta velocidad se calcula en función de la diferencia entre la posición céntrica del Auto Robot en la imagen actual y la posición del mismo en la captura previa.

A modo de ejemplo se presentan dos escenarios diferentes: en ambos escenarios el vehículo está orientado en un ángulo de 180°, el ángulo de direccionamiento máximo es 30°y la velocidad es la misma en los dos casos. En la figura 6.55 se puede visualizar la formación de un polígono con escasa apertura, debido a que la diferencia angular entre la posición del vehículo (180°) y la del camino (160°) es mínima; mientras que en la figura 6.56, ocurre lo contrario, el vehículo está posicionado a 180°y debe dirigirse a 270°, generando entonces una apertura de visión máxima (30°) en la dirección que corresponda.

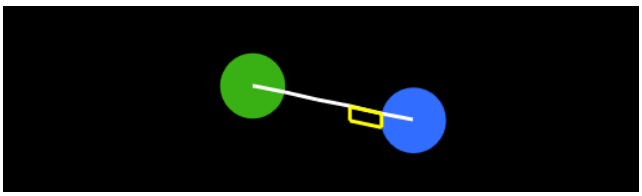

Figura 6.55: Paralelogramo con escasa apertura

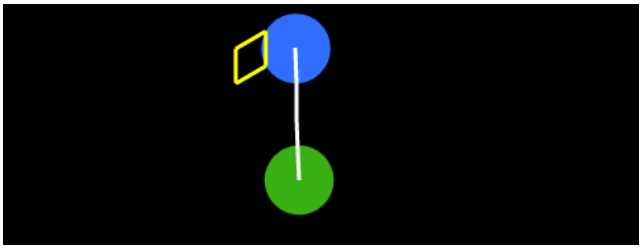

Figura 6.56: Paralelogramo con apertura máxima

Para calcular la maniobra a ejecutar por el vehículo, se precisan como se explicó, diferentes datos:

- point: centro, radio y grado del vehículo detectado.
- max_steer: ángulo máximo de giro del vehículo.
- dist: velocidad en pixeles que se dirige el vehículo
- degree_ref: la referencia al grado más cercano a alcanzar (obtenido en el método de suavizado).
- obstacles: los obstáculos.
- stage: límites del escenario.

Primeramente se debe calcular el grado al que deberá orientar las ruedas el Auto Robot. Este grado será la diferencia entre la posición actual ("current") del vehículo y el grado de referencia a alcanzar ("ref"), como muestra el cálculo siguiente:

$$\text{diff} = ((\text{ref - current}) + 180) \,\%\, 360 - 180$$

Luego, es necesario calcular el sentido de dirección. El primer objetivo de este proceso es obtener aquellos 4 puntos que forman el polígono de colisión del vehículo. Para esto, es necesario conocer el ángulo máximo de giro por parte del auto robot ("max_steer"). Para obtener los 4 puntos mencionados se utiliza la función "calculate_points_manuever"; como se puede observar, este método retorna el polígono representado (figura 6.54):

```
1 def calculate_points_manuever(degree_add=0):
2     p1 = point_at_distance_angle(center, point.radius, (degree_add
  +degree+oppening) %360)
3
```

```
    p2 = point_at_distance_angle(center, point.radius, (degree_add
    +degree-oppening) % 360)

    p3 = point_at_distance_angle(p1, dist, ((degree_add+degree-
    oppening) * orientation) % 360)

    p4 = point_at_distance_angle(p2, dist, ((degree_add+degree-
    oppening) * orientation) % 360)

    return geo.Polygon([p1, p3, p4, p2])
```

Obtenido el polígono, se precisa determinar si el mismo colisiona con un obstáculo, o queda fuera del escenario. Con el método "intersects" es posible conocer esta situación, siendo verdadero cuando ocurre alguna de estas dos situaciones, o falso en caso contrario:

```
def intersects(poly):
    ints = [poly.intersects(o) for o in obstacles]
    ints.append(not poly.within(stage))
    return True in ints
```

En caso que el método anterior retorne verdadero, quiere decir que si el robot sigue desplazándose en el mismo sentido, efectivamente colisionará con algún obstáculo o traspasará por fuera de los límites del escenario, por lo que se debe corroborar si puede desplazar en sentido opuesto. Con este fin, es llamado nuevamente el método "calculate_points_manuever", pero esta vez acumulando 180 grados, es decir el sentido opuesto. En las figuras 6.57 y 6.58 se presentan dos ejemplos, en ambos ejemplos el vehículo está orientado a 270°.

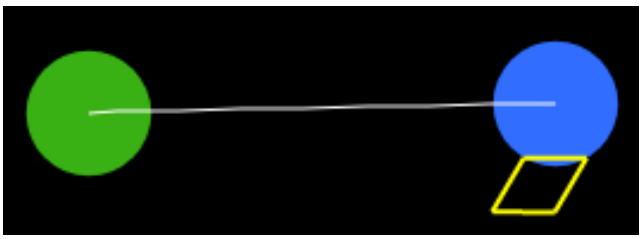

Figura 6.57: Ejemplo paralelogramo sin colisión

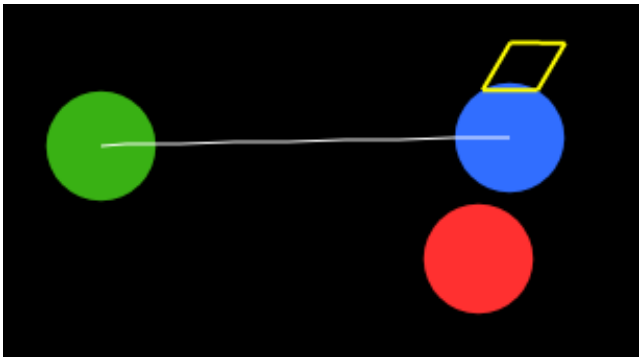

Figura 6.58: Ejemplo paralelogramo con colisión

Como se puede apreciar en la figura 6.58, se determinó por el método anterior que el vehículo está por colisionar con un obstáculo, por lo que invierte el sentido del desplazamiento del Auto Robot, lo que permite acomodar la estructura para poder seguir por la trayectoria planificada.

En caso que el método anterior informa que el paralelogramo sigue colisionando, se actualizará la dirección en 0 para que el vehículo frene sus motores, debido a la imposibilidad de movimiento sin colisionar o traspasar los límites del escenario. En caso contrario, es actualizada la dirección en el sentido opuesto, como así también el grado de direccionamiento de las ruedas, ya que la cambiar de sentido, también cambia la orientación.

6.6.7. Conclusiones

Se desarrolló e implementó un sistema que permite planificar y detectar un camino que conecta al Auto Robot con el destino, evitando los obstáculos. Se propuso e implementó un mecanismo de optimización de la trayectoria, que permite reducir el número de desvíos incluidos en la planificación RRT. Además, fue logrado realizar un suavizado de la ruta, lo que soluciona la problemática del Auto Robot en cuanto al impedimento de realizar giros bruscos.

Se diseñó y desarrolló un método de replanificación que permite ajustar caminos en tiempo real.

Por último, se desarrolló un algoritmo que permite conocer la maniobra a realizar por el vehículo, acomodándose cuando sea necesario con el fin de evitar colisiones con obstáculos y no exceder de los límites del escenario.

Todas estas funcionalidades se diseñaron y desarrollaron en forma eficiente, siendo útiles para sistemas en tiempo real como el propuesto en la presente tesina.

6.7. Interfaz WEB

Se desarrolló una interfaz WEB que permite al usuario interactuar con el sistema; por un lado, se muestra una sección que permite simular la determinación de caminos, con la propiedad de configurar diferentes aspectos, como por ejemplo el radio de los objetos, el grado al que está dirigido el vehículo, etc.

Por otro lado, se encuentra una sección denominada tiempo real o "live", utilizada para dar comienzo con la ejecución del sistema y visualizar en todo momento los eventos que ocurren en el transcurso del mismo.

La interfaz fue desarrollada con tecnologías de programación WEB, como "HTML", "JavaScript", "CSS". Por otro lado, la comunicación con el sistema fue realizado mediante sockets, haciendo uso de la librería "socket.io" (Javascript) y el microframework "flask" (Python).

6.7.1. Interfaz de simulación

Esta interfaz desarrollada permite realizar una simulación del camino hecha por el usuario, como también utilizando una captura. Como muestra la figura 6.59, se presenta la posibilidad de configurar una gran variedad de aspectos para lograr una simulación más realista. Por un lado, es posible especificar propiedades de los objetos, como por ejemplo su radio, el grado de orientación y su color (para distinguirlos en el simulador). A su vez, están presentes botones para agregar un inicio, un destino, y diversos obstáculos, con la facilidad de remover tanto el último obstáculo agregado como también todos los presentes.

Por otro lado, es posible escoger entre los algoritmos de planificación desarrollados (por el momento, entre RRT, RRT* e InformedRRT*), y también optar por ver la animación de cómo fue creado el árbol en busca del destino, o bien visualizar el resultado de la optimización o del suavizado. Contiene dos botones para realizar una planificación completa o replanificaciones de la misma.

Permite además, exportar e importar tanto la configuración establecida como también la planificación generada,

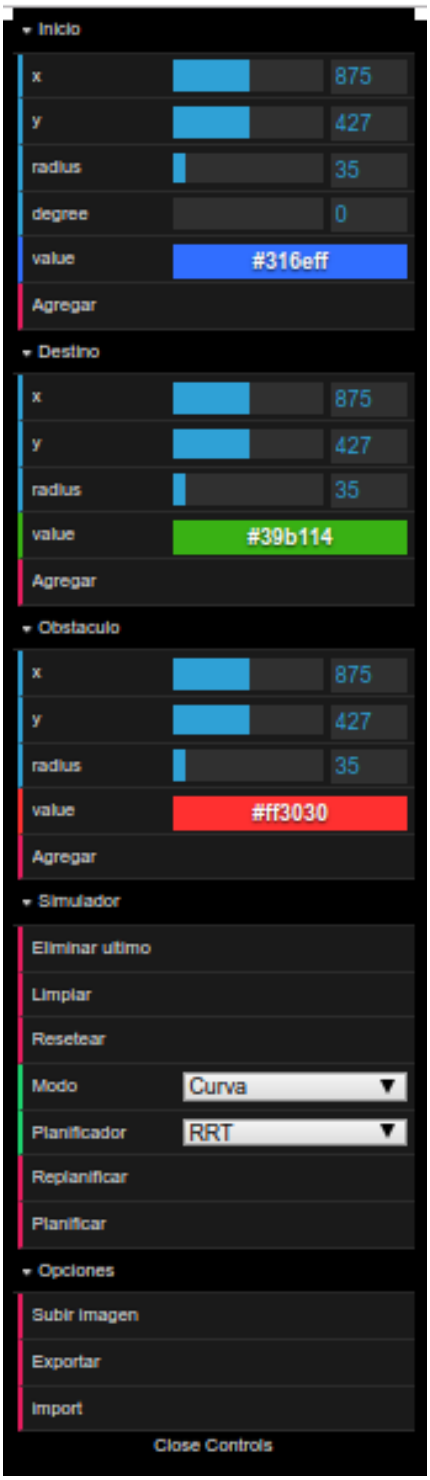

Figura 6.59: Control y configuración de la simulación

Otra sección de la interfaz de simulación es el entorno del simulador, en donde se ven reflejados los objetos de la simulación, es decir, el punto de inicio, destino y obstáculos (junto con su radio y colores), y la ruta trazada.

Para agregar objetos a la simulación, se debe presionar en cualquier parte del simulador y se incluirán los mismos donde corresponda. La figura 6.60 muestra el apartado del simulador.

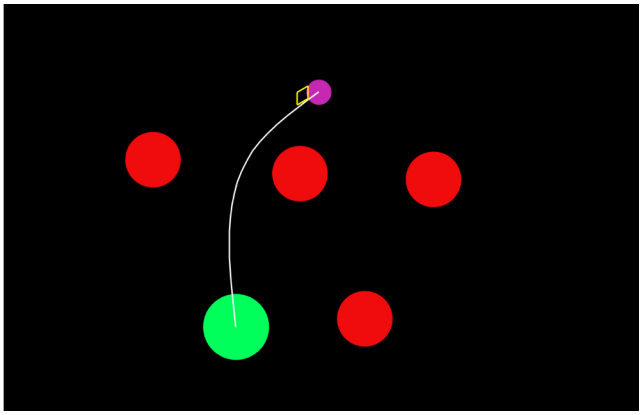

Figura 6.60: Simulador

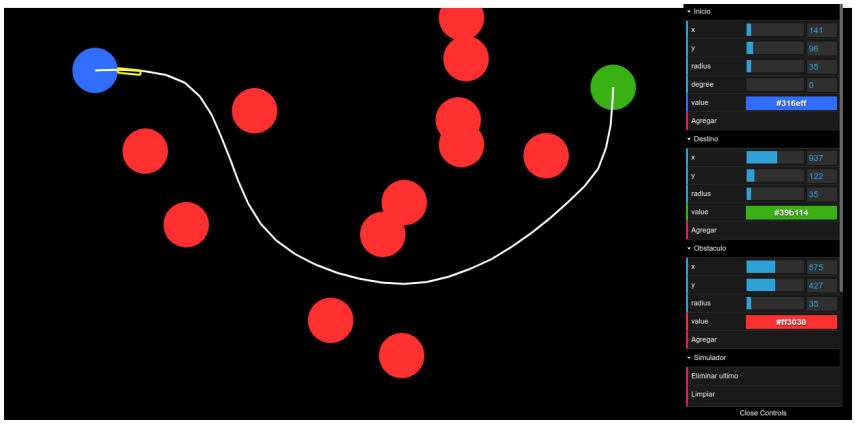

Figura 6.61: Interfaz de simulación completa

6.7.2. Interfaz en tiempo real

Este apartado brinda la posibilidad de gestionar la ejecución y el control del sistema.

Se diseñó una terminal que registra cada uno de los eventos ocurridos en el transcurso de la ejecución del sistema. Esto es de ayuda al usuario para

entender lo que está ocurriendo en cada momento. Un evento está conformado por un actor (Auto Robot, Dron o System), un horario en el que ocurrió el evento, un texto descriptivo de lo ocurrido y un color que representa el grado de correctitud (blanco son mensajes de notificación, verde son mensajes de éxito, rojo son mensajes de error, amarillo mensajes de alerta y azul son eventos de los robots). En la figura 6.62 se muestra un ejemplo de la terminal.

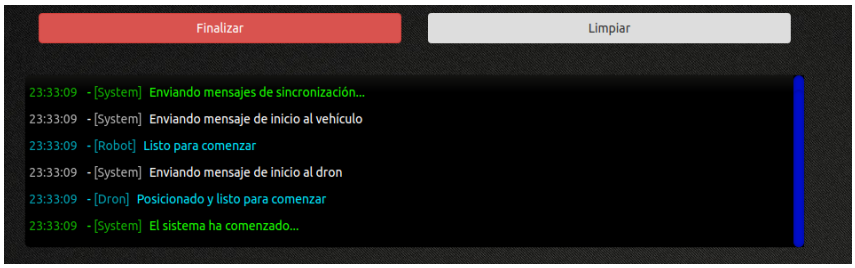

Figura 6.62: Interfaz live: terminal

Por otro lado, la interfaz permite visualizar el streaming recibido y apreciar el resultado de la detección de los objetos y el trazo de la ruta planificada, como muestra la figura 6.63. Ambas acciones son producidas en tiempo real.

Figura 6.63: Interfaz live: visualización del streaming, junto con la detección y planificación

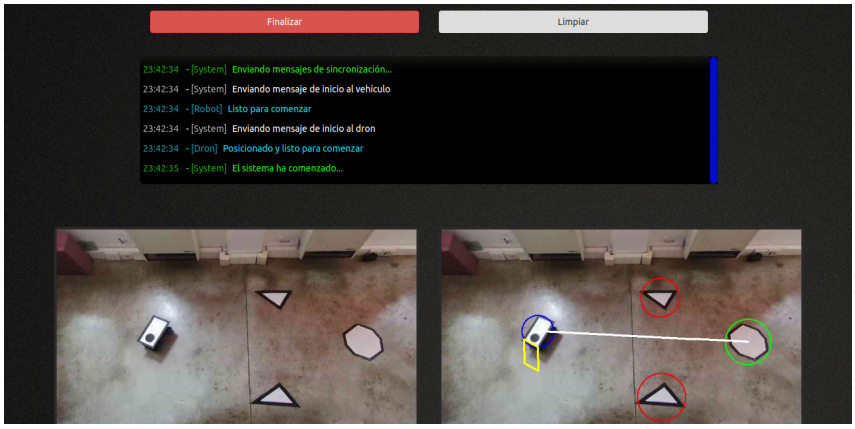

Figura 6.64: Interfaz live completa

6.7.3. Conclusiones

Fue diseñado e implementado una interfaz WEB que permite gestionar los aspectos funcionales del sistema en general. Brinda la posibilidad de realizar investigaciones y experimentos utilizando diversos algoritmos de planificación, de forma manual o en base a una imagen capturada del escenario, con facilidades de guardado y carga de los mismos.

Por otro lado, provee al usuario un apartado en donde puede percibir en tiempo real los eventos ocurridos en el sistema, como también un video del escenario y propicia la visualización de los resultados de las planificaciones de caminos y detecciones de figuras efectuadas.

6.8. Latencia de comunicaciones

Para esta tesina fue de interés realizar distintas mediciones con el fin de comparar el tiempo de respuesta del sistema desarrollado tanto en un Cloud público como así también en un Cloud privado.

Fueron realizadas mediciones de latencia en los diferentes canales de comunicación presentes en el sistema:

- Enlace del Dron a RPi: el dron envía un streaming de video a la placa Raspberry Pi 3, a través de una red Wi-Fi local sobre UDP.
- Enlace RPi a Cloud: la placa RPi redirige los paquetes del streaming hacia el Cloud vía UDP.

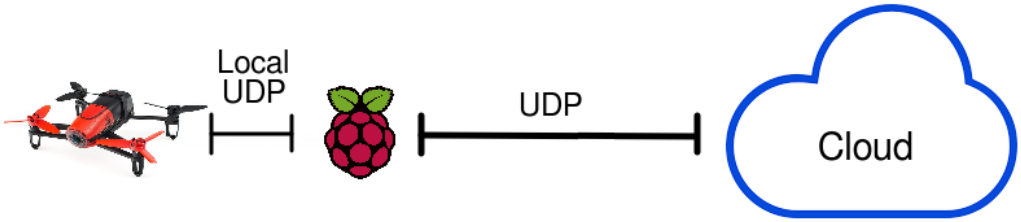

Figura 6.65: Gráfico con las comunicaciones a medir.

6.8.1. Enlace Dron a RPi

Como se explicó anteriormente, la conexión entre el dron y la Rasberry Pi 3 es realizada en forma local. Para esta medición se realizaron 1000 pruebas de ping hacia la IP del dron mediante el comando *"ping 192.168.42.1 -c 1000"* ejecutado desde la RPi.

Los valores obtenidos fueron:

Min.	Max.	Promedio
1.398ms	14.175ms	3.716ms

6.8.2. Enlace RPi al Cloud

Con el fin de medir la latencia generada en el enlace de comunicación entre la placa de desarrollo RPi y el Cloud, fue necesario sincronizar los relojes de ambas plataformas. Esta sincronización se realizó haciendo uso de la herramienta NTP [48].

El propósito de esta medición es conocer el tiempo requerido en el envío de una imagen por completo.

El dron envía fracciones de una imagen codificada en H.264 los cuales tienen un tamaño de 740 bytes aproximadamente; la imagen completa tiene un tamaño aproximado de 13 KBytes (13000 bytes). El streaming de video generado por el dron envía 24 de estas imágenes en un segundo dando un ancho de banda aproximado de 2400Kbits/seg.

Para conocer la latencia de comunicación a través de UDP se realizan los siguientes pasos:

1. Se calcula la fecha actual en milisegundos y se agrega al paquete UDP del streaming.

2. Se envía el paquete al Cloud a través de UDP.

3. Al recibir el paquete, se extrae y almacena la fecha presente en el mismo.

4. Se reciben los paquetes necesarios para totalizar una cantidad de 13Kbytes de información (tamaño aproximado de cada imagen).

5. Obtenidos los 13KBytes, se calcula la diferencia entre el tiempo actual y el tiempo almacenado anteriormente, la cual indica el tiempo transcurrido entre el envío y la recepción de una imagen completa.

Fueron enviados 10000 imágenes encodeadas, tanto en un despliegue en el Cloud público como en el privado.

Los resultados de estas mediciones pueden observarse en la siguiente tabla:

	Mínimo	Máximo	Promedio
Cloud privado	6ms	83ms	40ms
Cloud público	87ms	219ms	123ms

Como puede observarse, la latencia existente en el Cloud público es mínima y posibilita la realización de una comunicación intermitente hacia el mismo. Si bien el Cloud privado genera una menor latencia de comunicación, este despliegue no posee las propiedades de elasticidad y escalabilidad provistas por el Cloud público, generando limitaciones en diferentes aspectos que fueron mencionados a lo largo de la tesina.

Es deseable ahondar más en la investigación de la latencia con el fin de disminuir estos tiempos y optimizar el flujo de comunicación hacia el Cloud.

Capítulo 7

Resultados obtenidos

7.1. Introducción

En esta sección se detallan los resultados obtenidos, a partir los diferentes experimentos planteados en la sección de trabajo experimental.

Para cada prueba, se muestran los resultados obtenidos tanto por el sistema de procesamiento de imágenes como también el sistema de planificación de caminos. Además, se puede visualizar en diversas imágenes el desplazamiento efectuado por el vehículo.

7.1.1. Primer experimento

En esta prueba, se espera detectar únicamente el Auto Robot. El resultado que se obtuvo se observa en al figura 7.1.

Figura 7.1: Experimento N°1: resultado de la detección

Como se esperaba, el procesamiento de imágenes logró detectar e identificar al Auto Robot, a diferencia de los otros elementos del escenario no presentes. El sistema continúa verificando si se ha incorporado el destino. Ante 5 detecciones fallidas, se aborta la ejecución.

7.1.2. Segundo experimento

En esta segunda prueba, se muestran los resultados de una ejecución completa dado que están presentes todos los elementos fundamentales del escenario (Auto Robot y destino), como se ve en la figura 7.2.

Figura 7.2: Experimento N°2: resultado de la detección

Como puede visualizarse en las subsiguientes figuras, la planificación de caminos detectó una trayectoria recta, desde la posición inicial del robot hasta el destino, y encontró en todo momento ambos elementos dentro del escenario.

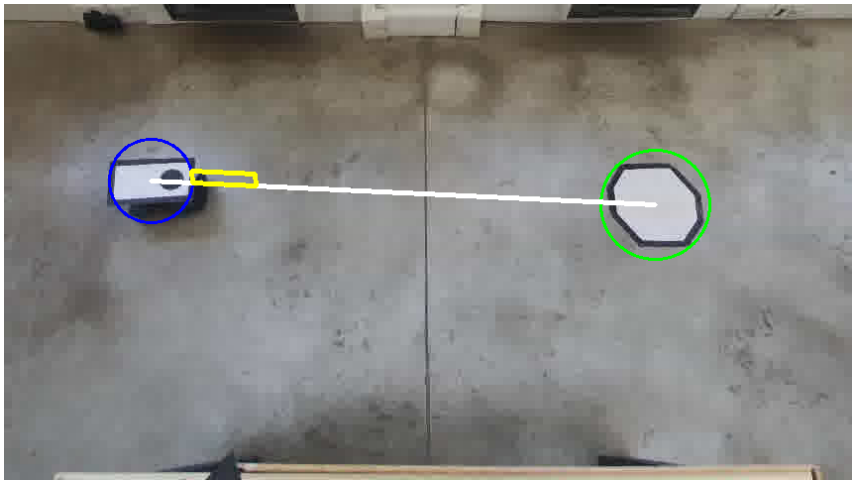

Figura 7.3: Experimento N°2: desplazamiento 1

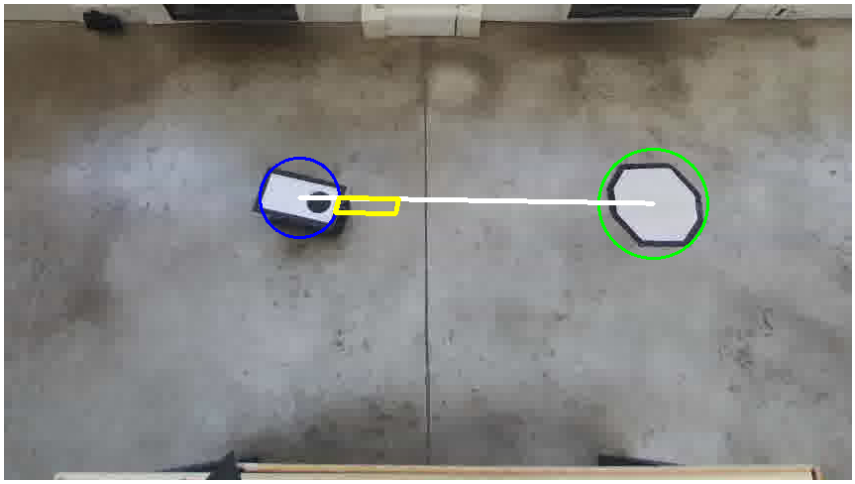

Figura 7.4: Experimento N°2: desplazamiento 2

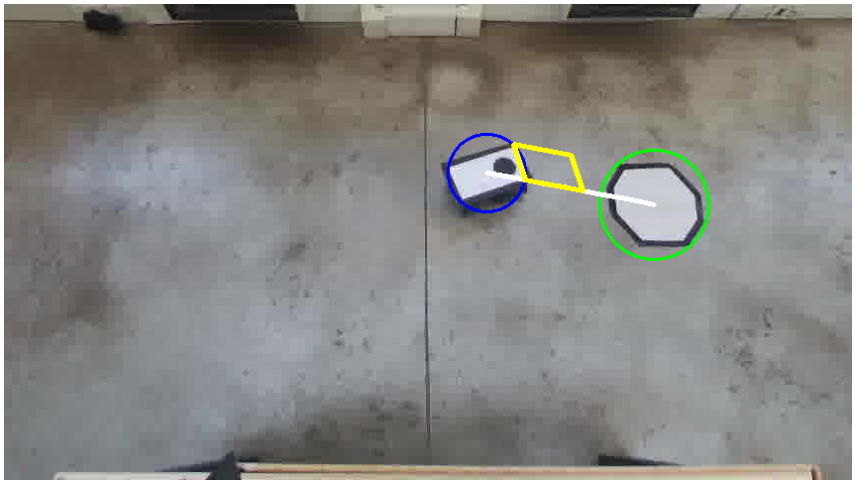

Figura 7.5: Experimento N°2: desplazamiento 3

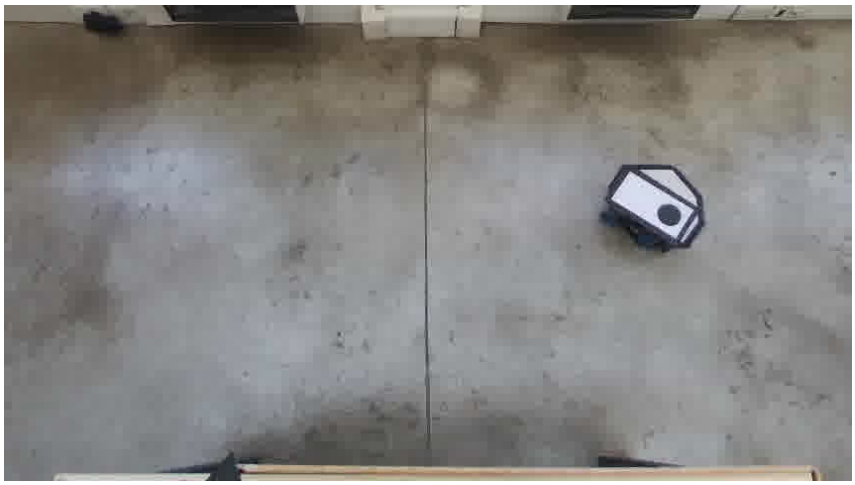

Figura 7.6: Experimento N°2: desplazamiento 4

La figura 7.6 muestra el resultado final de este experimento, donde permite apreciar que el vehículo ingresó al destino, finalizando el sistema satisfactoriamente.

7.1.3. Tercer experimento

El presente experimento tiene como meta evadir el obstáculo, encontrando el camino óptimo que permita al vehículo desplazarse hacia el destino. En la figura 7.7 es posible apreciar el procesamiento de detección tanto del Auto Robot, como así también del obstáculo y la meta.

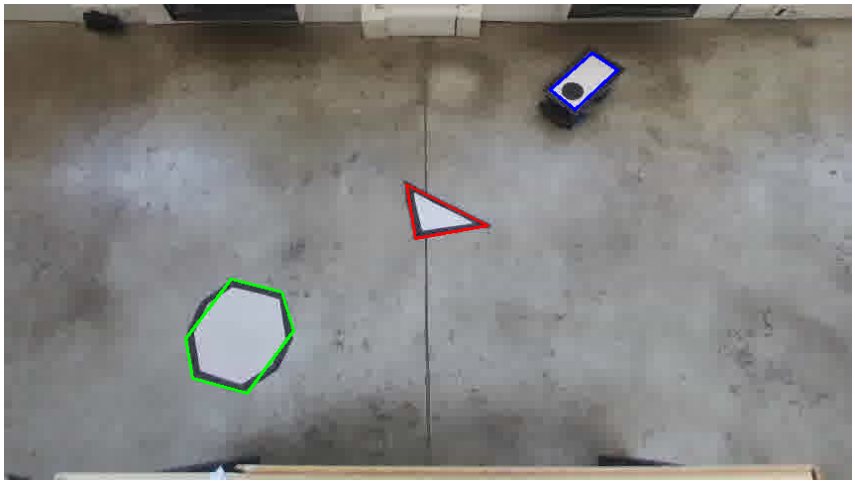

Figura 7.7: Experimento N°3: resultado de la detección

En las siguientes figuras se muestra el desplazamiento ejecutado por el robot, donde se aprecia el modo de evasión del obstáculo junto con el ajuste realizado en tiempo real:

En la figura 7.8 se planifica y encuentra el camino óptimo a realizar. En las figuras 7.9 y 7.10 se puede observar el ajuste en el desplazamiento para esquivar el obstáculo presente.

La figura 7.12 muestra el resultado final del experimento

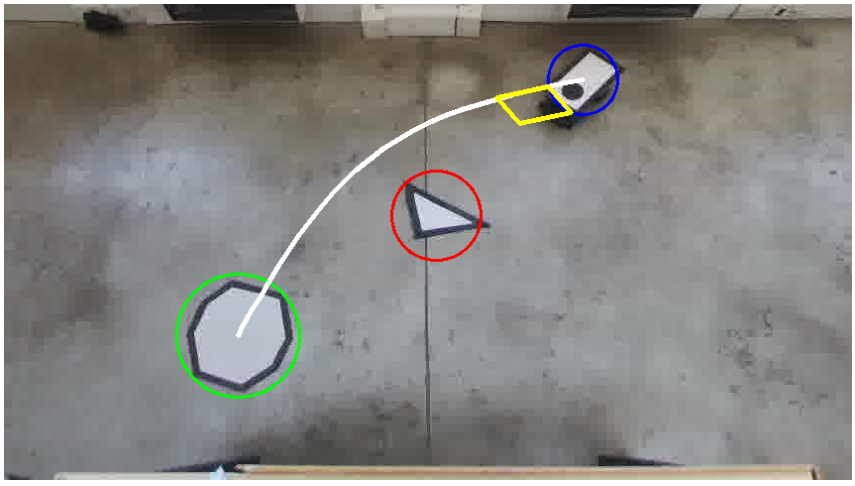

Figura 7.8: Experimento N°3: desplazamiento 1

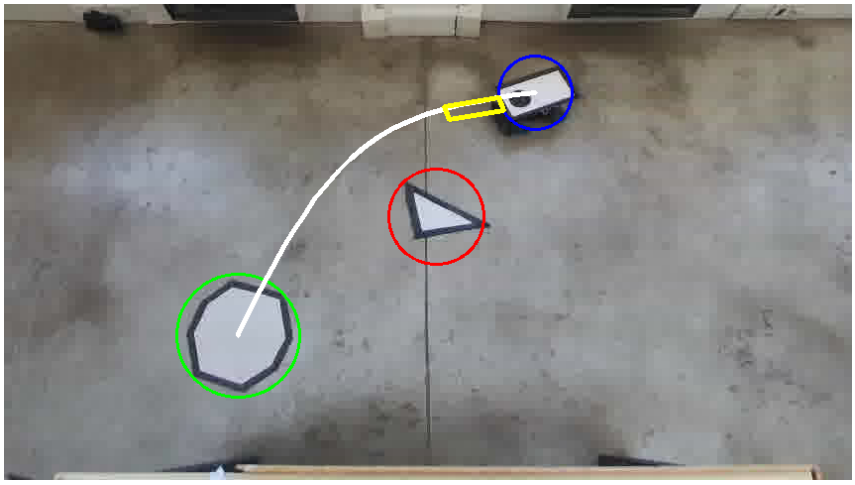

Figura 7.9: Experimento N°3: desplazamiento 3

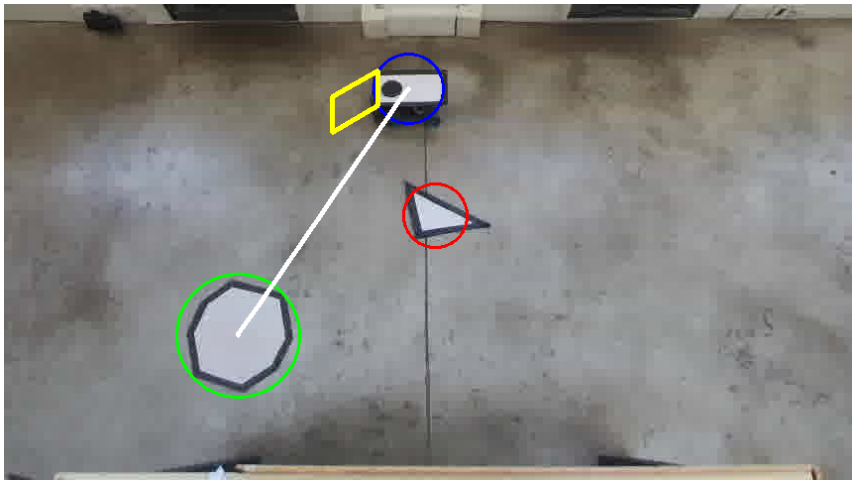

Figura 7.10: Experimento N°3: desplazamiento 3

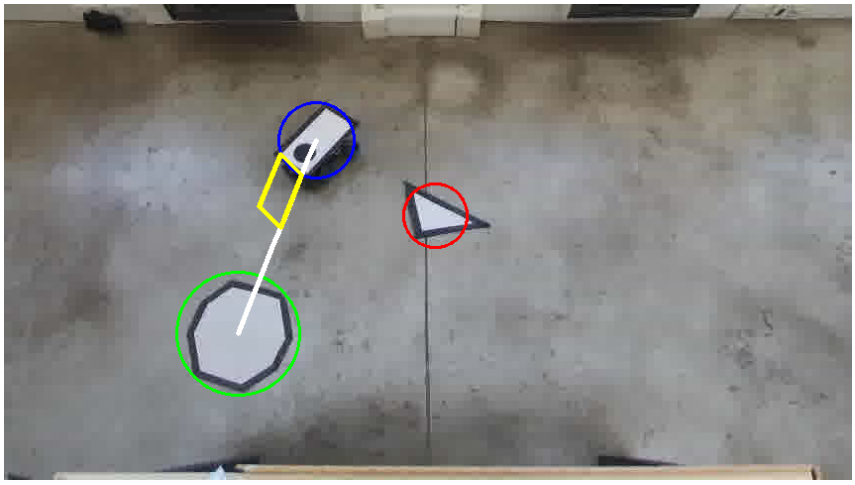

Figura 7.11: Experimento N°3: desplazamiento 4

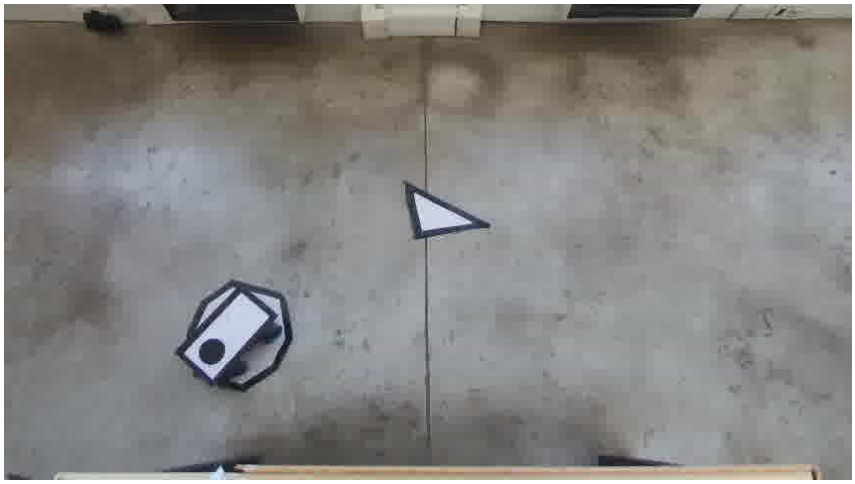

Figura 7.12: Experimento N°3: desplazamiento 5

7.1.4. Cuarto experimento

Este experimento consiste en una barrera que impide la libre circulación del vehículo hacia el destino. En la figura 7.13 se aprecia el procesamiento de detección, donde satisfactoriamente fueron encontrados los 4 obstáculos, el vehículo y el destino.

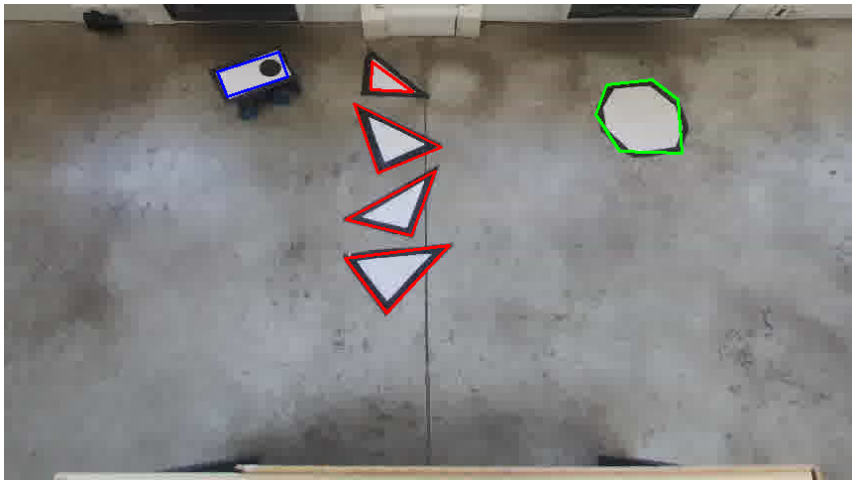

Figura 7.13: Experimento N°4: resultado de la detección

El vehículo comenzó a desplazarse hacia delante como muestra la figura 7.14, hasta que se detecta que el Auto Robot está próximo a colisionar con un obstáculo (figura 7.15), por lo que se invierte el sentido de dirección y orientación del vehículo (figura 7.16).

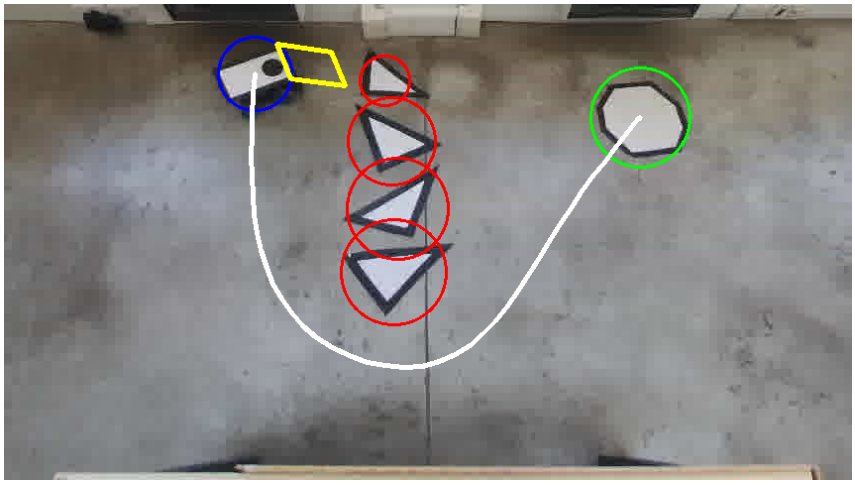

Figura 7.14: Experimento N°4: desplazamiento 1

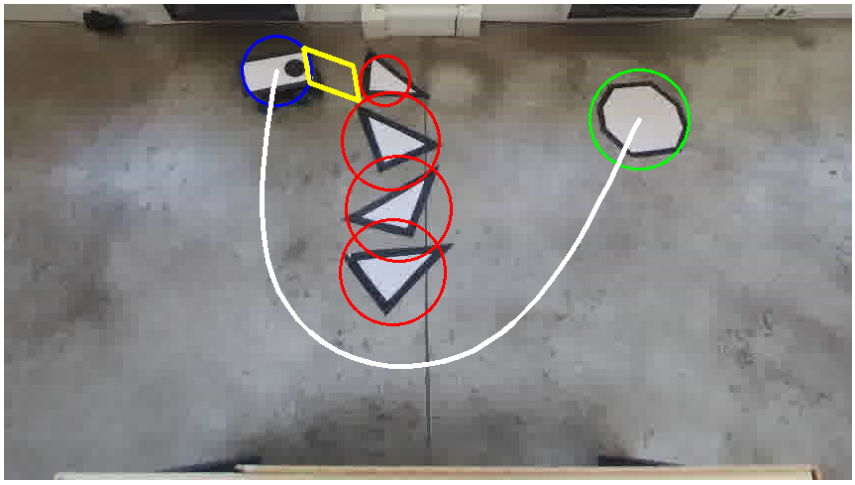

Figura 7.15: Experimento N°4: desplazamiento 3

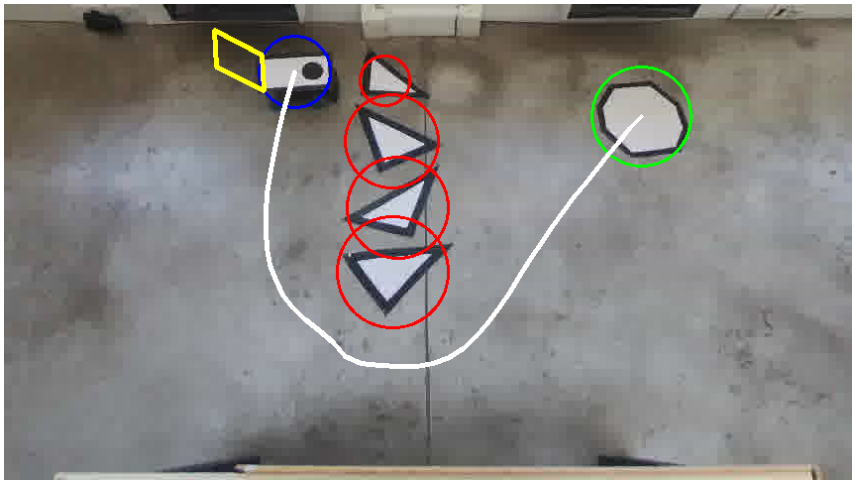

Figura 7.16: Experimento N°4: desplazamiento 3

El vehículo se siguió desplazando hacia la parte baja del escenario (figura 7.17), hasta que se detecta la posibilidad de traspasar la barrera, como muestra la figura 7.18. El vehículo supera los obstáculos y alcanza el destino (figura 7.21)

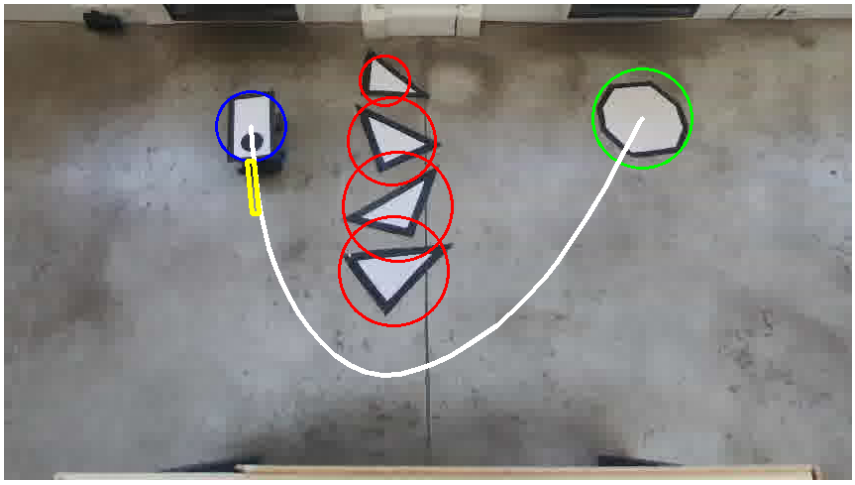

Figura 7.17: Experimento N°4: desplazamiento 4

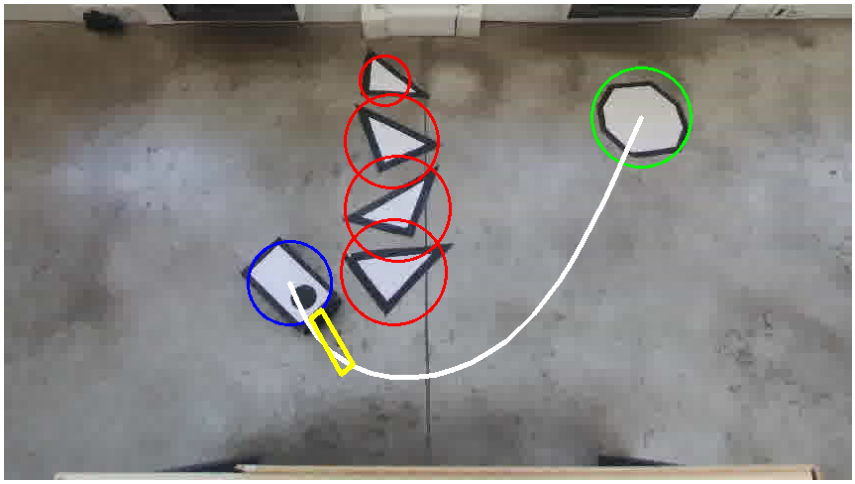

Figura 7.18: Experimento N°4: desplazamiento 5

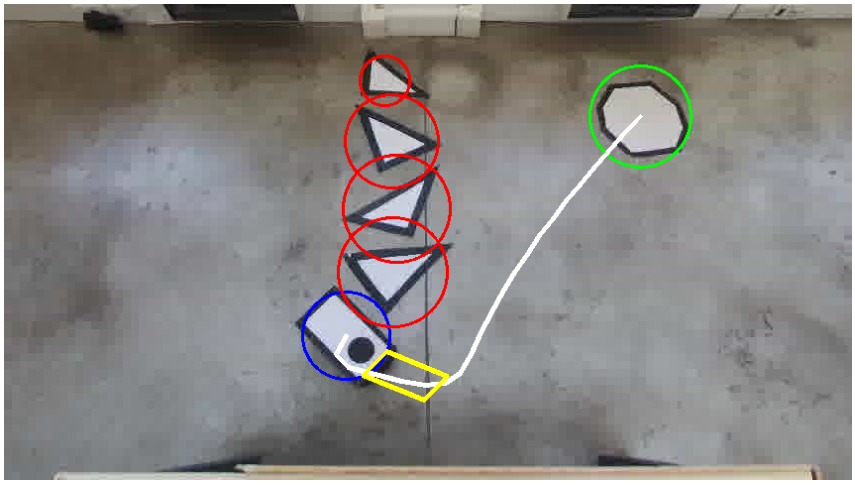

Figura 7.19: Experimento N°4: desplazamiento 6

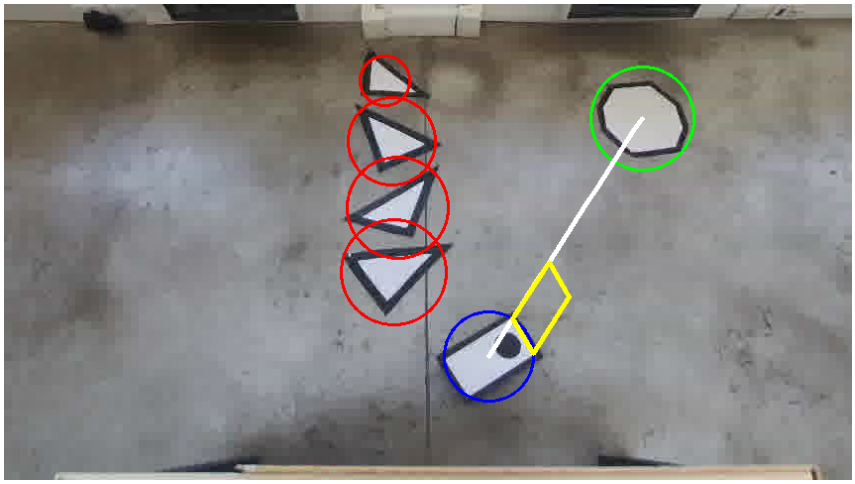

Figura 7.20: Experimento N°4: desplazamiento 7

Figura 7.21: Experimento N°4: desplazamiento 8

7.1.5. Quinto experimento

En la figura 7.22 se puede apreciar que el procesamiento de imágenes detectó tanto el vehículo, el destino y los 4 obstáculos que rodean a este último

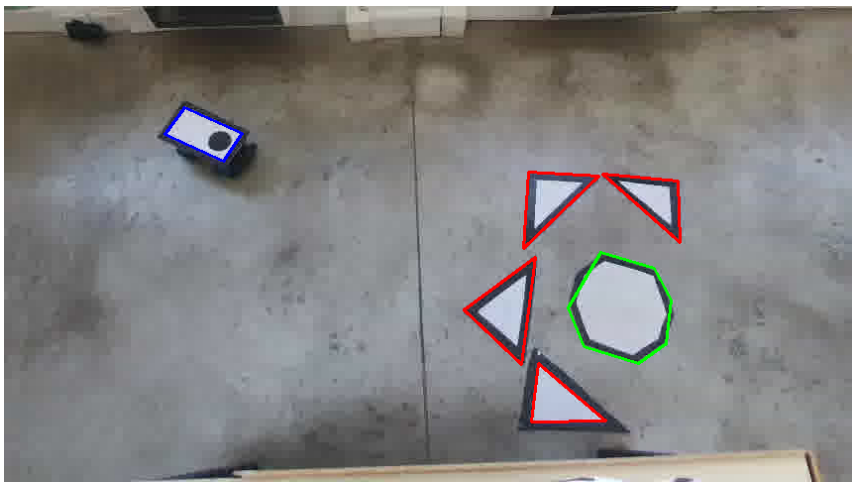

Figura 7.22: Experimento N°5: resultado de la detección

En primera medida se calculó el camino óptimo, como muestra la figura 7.23. Luego el robot fue avanzando (figura 7.24) y acomodando su posición para poder ingresar al entorno de obstáculos, donde reside el destino (figura 7.25 y 7.26). Por último en la figura 7.27, se aprecia el resultado final obtenido.

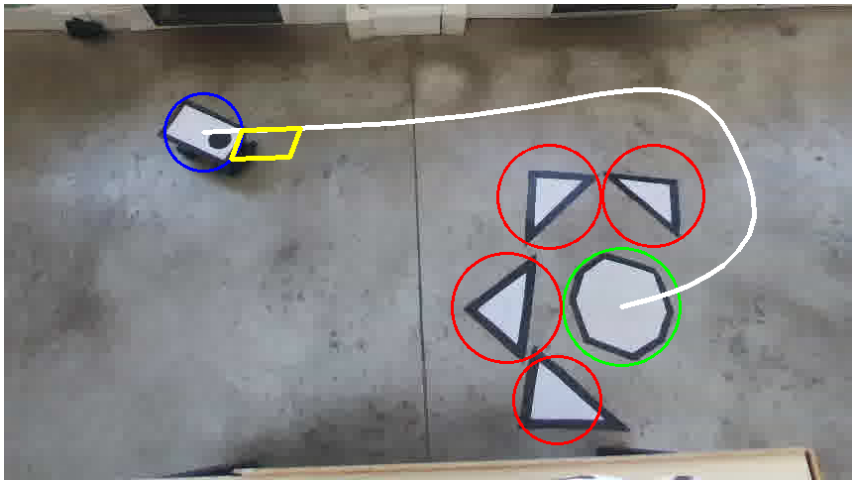

Figura 7.23: Experimento N°5: desplazamiento 1

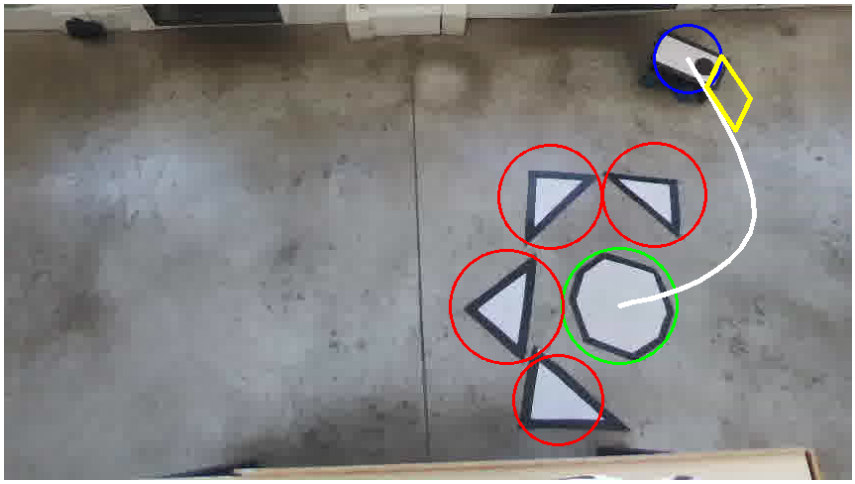

Figura 7.24: Experimento N°5: desplazamiento 2

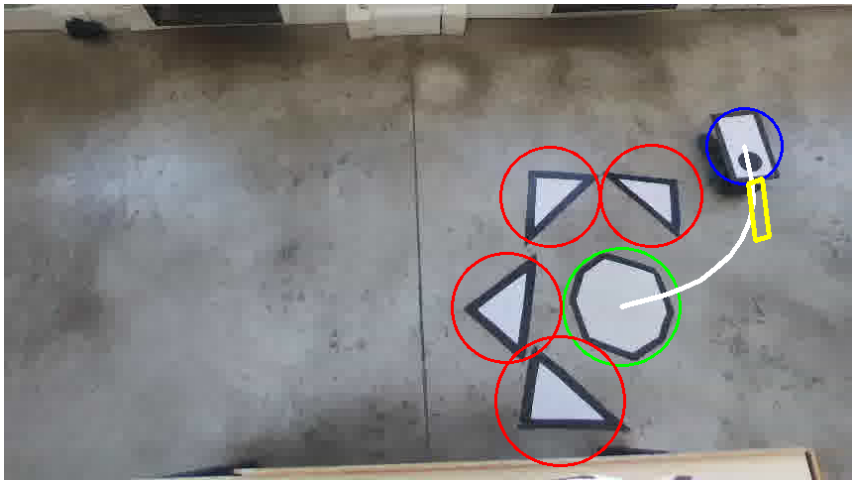

Figura 7.25: Experimento N°5: desplazamiento 3

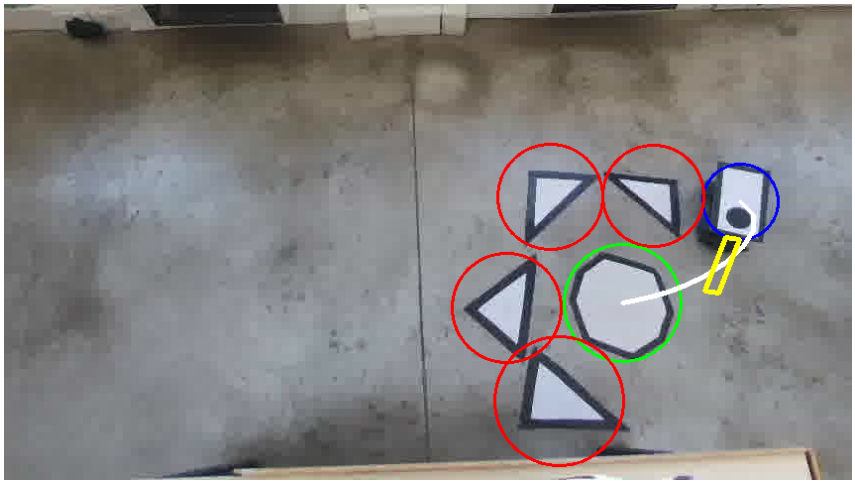

Figura 7.26: Experimento N°5: desplazamiento 4

Figura 7.27: Experimento N°5: desplazamiento 5

7.1.6. Sexto experimento

Para este experimento se detectaron 6 obstáculos, el vehículo y el destino, como muestra la figura 7.28.

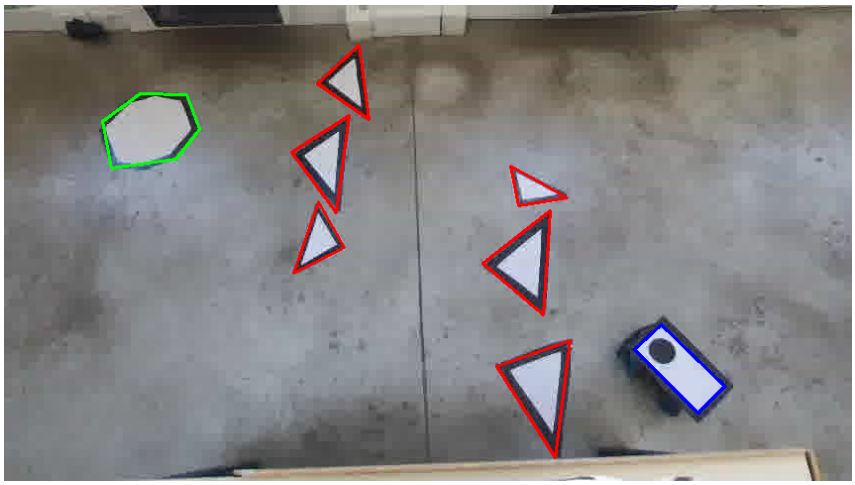

Figura 7.28: Experimento N°6: resultado de la detección

El vehículo comienza su desplazamiento ajustando su orientación a lo alto del escenario (figura 7.29 y 7.30). Cuando la planificación de caminos determina que puede orientarse la izquierda, el vehículo gira sus ruedas y se dirige en ese sentido, como muestra la figura 7.31. Luego se orienta hacia abajo (figura 7.32), hasta tener la posibilidad de traspasar la barrera sin colisionar con los obstáculos. En ese momento modifica su orientación hacia la derecha 7.33 y

se dirige hacia el escenario hasta alcanzar el objetivo, como se aprecia en las figuras 7.34 y 7.35

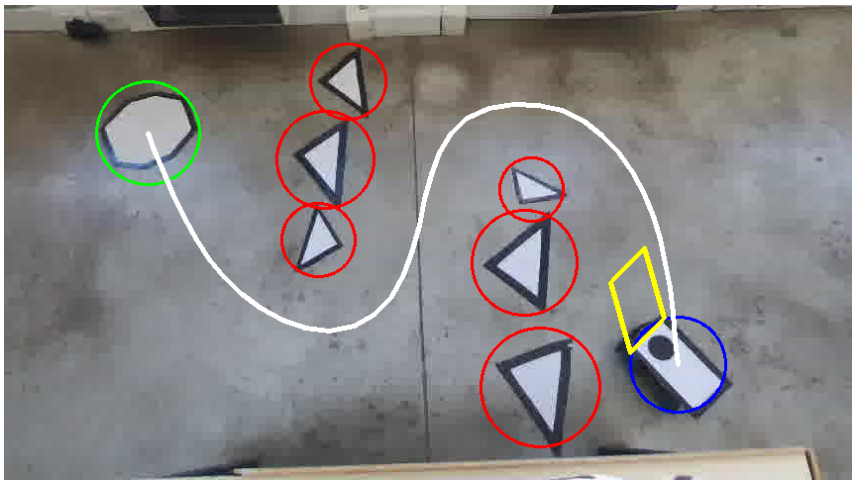

Figura 7.29: Experimento N°6: desplazamiento 1

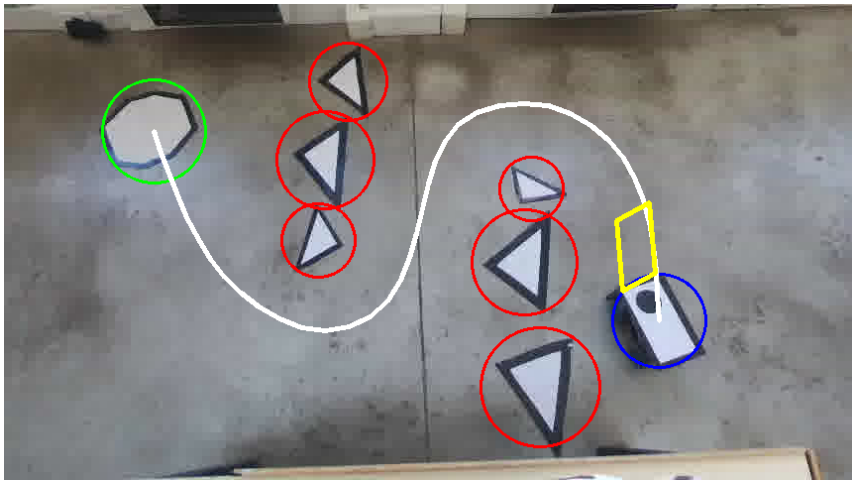

Figura 7.30: Experimento N°6: desplazamiento 2

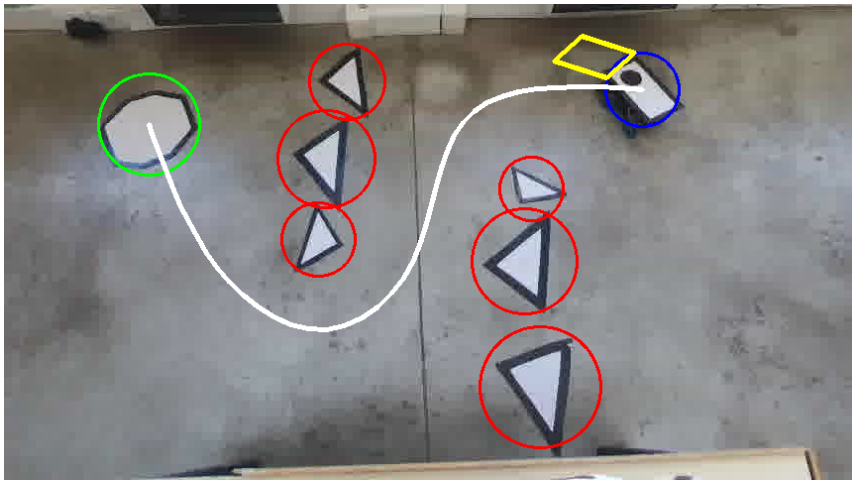

Figura 7.31: Experimento N°6: desplazamiento 3

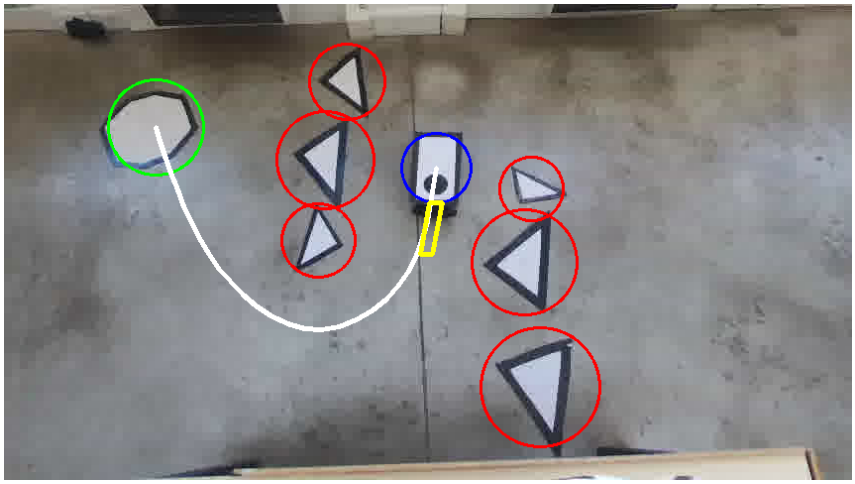

Figura 7.32: Experimento N°6: desplazamiento 4

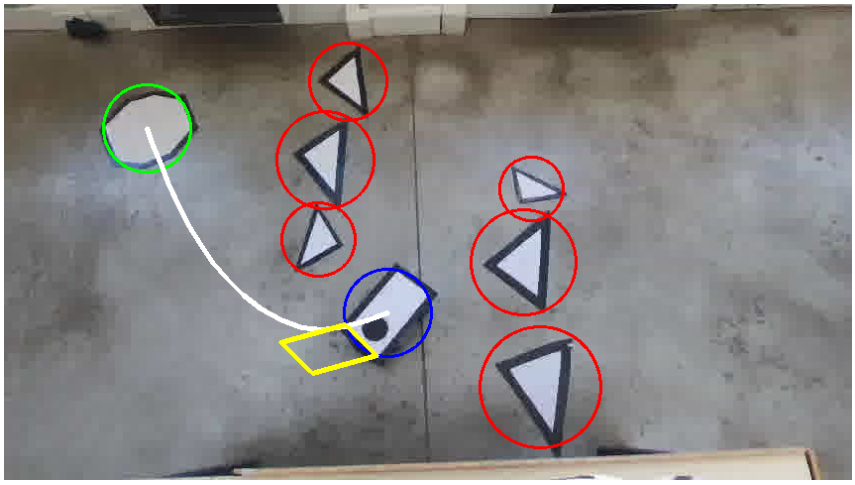

Figura 7.33: Experimento N°6: desplazamiento 5

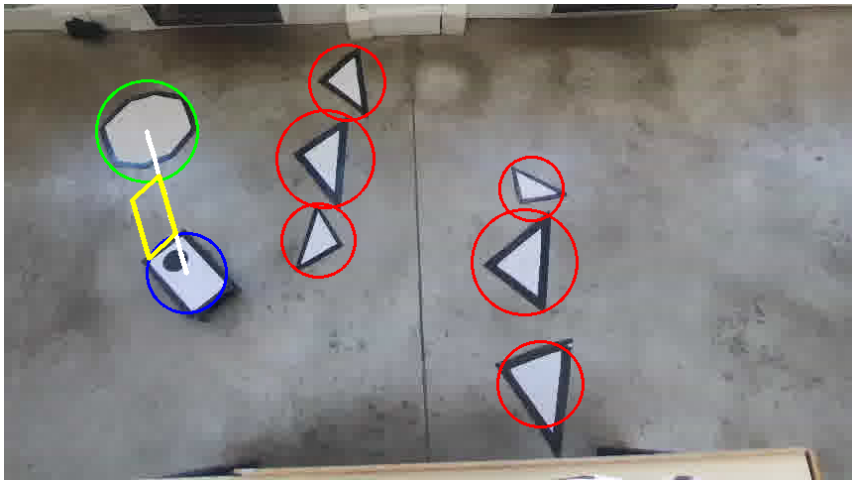

Figura 7.34: Experimento N°6: desplazamiento 6

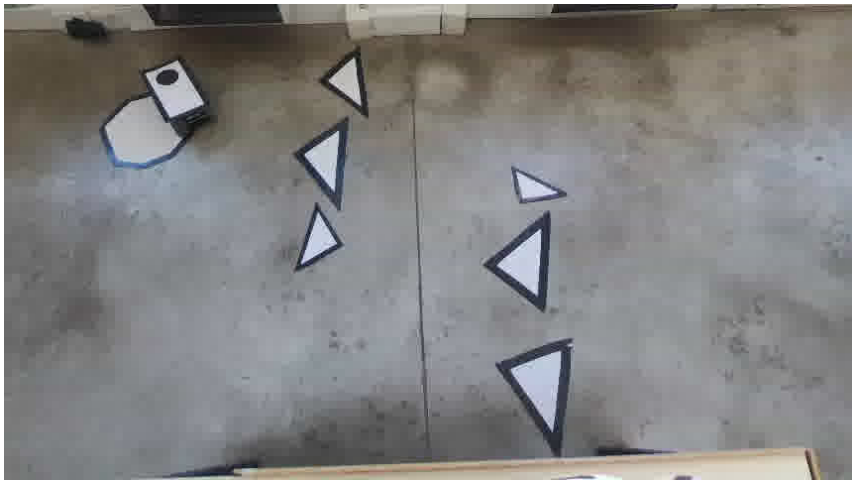

Figura 7.35: Experimento N°6: desplazamiento 7

Capítulo 8

Conclusiones y líneas de trabajo a futuro

8.1. Conclusiones

A lo largo de la tesina se realizó un análisis de las ventajas y desventajas de la utilización del paradigma Cloud Computing en sistemas Multi-Robots, ratificando que el paradigma Cloud Robotics favorecerá en gran medida a estos sistemas, subsanando las problemáticas físicas de los robots en cuanto a poder de cómputo, capacidad de almacenamiento, consumo energético, tareas cognitivas, etc.

Se diseñó, desarrolló e implementó un sistema de múltiples robots que permite el desplazamiento autónomo de un vehículo de 4 ruedas desde un inicio hacia un destino predeterminado, sorteando los obstáculos presentes a lo largo de la trayectoria. Para lograr tal fin, se utilizó una instancia EC2 en el Cloud público de Amazon Web Services, en donde se delega todo el procesamiento de cómputo del sistema, comunicando a los robots las acciones a realizar.

Para desplegar el sistema Multi-Robots, se confeccionó un modelo que simula un vehículo tradicional, es decir, el prototipo empleado es capaz desplazarse tanto hacia delante como atrás y direccionarse en el sentido que corresponda. El vehículo implementado contiene el hardware mínimo indispensable para gestionar los sensores integrados, junto con el software que permite ejecutar las directivas comandadas por el Cloud. Se diseñó un sistema de alimentación que permite suministrar la energía necesaria al vehículo, y se confeccionó un módulo cargador de baterías casero con el fin de gestionar la carga.

Por otro lado, se utilizó un dron que permite ejecutar su pilotaje en forma autónoma y realizar un video en tiempo real del área de trabajo con un ángulo de visión de la cámara fructífero para el cumplimiento del objetivo.

Fue cuestión de debate la utilización de diversos métodos de percepción del entorno y se investigó dos diferentes tipos de drones, nombrando ventajas y desventajas en la utilización de cada uno para el presente trabajo. Se desarrolló un software que posibilita la gestión y el manejo del dispositivo de vuelo y se diseñó el nexo que permite comunicar al dron con el Cloud.

El sistema desarrollado consiste en un dispositivo de vuelo que se eleva y dirige hacia el escenario, enviando un streaming de video del entorno hacia el Cloud; instancia donde se procesan dichas capturas y se envían los movimientos a ejecutar por el Auto Robot. Con este propósito, se utilizó un programa de decodificación y se diseñó la lógica y sincronización necesaria para la obtención del video en tiempo real provisto por el dron. Se desarrollaron algoritmos de procesamiento de imágenes para detección de figuras geométricas, mediante la aplicación de varios filtros que permite transformar las capturas en diferentes aspectos con el fin de detectar los objetos en entornos variados. A su vez, se implementaron algoritmos de planificación de caminos, tanto para encontrar la ruta óptima como así también para replanificar en función de una trayectoria previa. Se diseñó y desarrolló un algoritmo de optimización de caminos, con el fin de reducir la cantidad de puntos de la planificación y así obtener trayectorias menos complejas. Luego, se desarrolló un algoritmo de transformación de la ruta que permite obtener un camino suavizado, evitando la generación de giros bruscos que puedan perjudicar el desplazamiento del vehículo.

Para traducir el camino resultante en movimientos a ejecutar por el auto robot, se implementó y desarrolló un algoritmo que permite conocer tanto el sentido de dirección como el ángulo de orientación que el vehículo deberá imitar para evitar colisionar con los obstáculos, como también con los límites del escenario.

Las funcionalidades y el procesamiento previamente detallado es realizado en tiempo real, por lo que fue necesario mejorar la eficiencia de los algoritmos a través de la paralelización de las funciones y el método de sincronización correspondiente, para obtener resultados en el tiempo y forma requeridos, sin la necesidad del aumento en las prestaciones de la instancia. Se desplegó una interfaz WEB que permite al usuario comenzar con la ejecución del sistema y visualizar en tiempo real las interacciones que se llevan a cabo entre el Cloud y el sistema Multi-Robots, como así también el streaming de video enviado por el dron y las imágenes resultantes de la detección de los objetos junto con la trayectoria trazada y el movimiento a realizar por el vehículo. Además, la interfaz presenta un apartado de simulación, que posibilita la realización de investigaciones y pruebas referentes a la obtención de caminos, haciendo uso de diferentes algoritmos de planificación y diversas facilidades provistas al usuario.

Se constató la viabilidad del sistema tanto en un despliegue de Cloud público como privado, teniendo diferencias de latencias despreciables entre ambos; siendo el Cloud público una solución factible para sistemas en tiempo real como el propuesto, debido a la capacidad de asignar recursos en forma dinámica, sin afectar en el rendimiento del sistema.

8.2. Trabajos futuros

A continuación, se describen los posibles trabajos futuros que se desprenden de esta tesina:

- Mejorar el procesamiento de imágenes de forma tal que se puedan detectar objetos reales en el escenario sin depender de figuras geométricas que los representen.
- Añadir uno o más Auto Robots al sistema adaptando la solución con el fin de simular una ciudad en la cual el entorno Multi-Robots pueda trabajar colaborativamente, analizando el desempeño del paradigma de Cloud Robotics en cuanto a escalabilidad y elasticidad.
- Paralelizar los algoritmos de planificación de caminos con el objetivo de reducir el tiempo de ejecución de los mismos, permitiendo realizar tareas mas complejas y de forma mas precisa en tiempo real.
- Implementar el uso de sensores en el Auto Robot, generando un nivel de autonomía del mismo como contingencia en caso de perder la conexión con el sistema.
- Investigar los tiempos de latencia y overhead generados en la comunicación y procesamiento en el Cloud, con el fin de disminuir los tiempos de respuesta en este paradigma.

Bibliografía

[1] Grance T Mell P. «The NIST Definition of Cloud Computing». En: (2011).

[2] Pettoruti J.; Rodriguez I.; Chichizola F.; De Giusti A. «Análisis de la degradación de las comunicaciones en algoritmos de cómputo científico en un Cloud privado». En: *XII Workshop de Procesamiento Distribuido y Paralelo (WPDP) – XVIII Congreso Argentino de Ciencias de la Computación (CACIC2012)* (2012).

[3] Rodriguez I.; Pettoruti J.E.; Chichizola F.; De Giusti A. «Despliegue de un Cloud Privado para entornos de cómputo científico». En: *Proceedings del XI Workshop de Procesamiento Distribuido y Paralelo (WPDP) - XVII Congreso Argentino de Ciencias de la Computación (CACIC 2011)* (2011).

[4] *RoboEarth*. `http://www.roboearth.org`.

[5] Kuffner J. «Cloud-enabled robots». En: *IEEE-RAS International Conference on Humanoid Robot* (2010).

[6] Wang L.; Liu M.; Meng M.; Siegwart R. «Towards Real-Time Multi-Sensor Information Retrieval in Cloud Robotic System». En: *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)* (2012).

[7] Turnbull L. «Cloud Robotics: Formation Control of a Multi Robot System Utilizing Cloud Infrastructure». En: *Proceedings of IEEE – Southeastcon* (2013).

[8] Atzori L.; Iera A. «The Internet of Things: A survey». En: *ELSEVIER Journal Computer Networks* (2010).

[9] Peter Mell; Timothy Grance. «The NIST Definition of Cloud Computing». En: *National Institute of Standards and Technology Special Publication 800-145* (2011).

[10] *Amazon Web Services*. `https://aws.amazon.com/es/`.

[11] *Amazon Elastic Compute Cloud*. https://aws.amazon.com/es/ec2/.

[12] *Amazon Internet Of Things*. https://aws.amazon.com/es/iot/.

[13] *TCP*. https://searchnetworking.techtarget.com/definition/TCP.

[14] *UDP*. https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol.

[15] *HTTP*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[16] *Protocolo MQTT*. https://www.oasis-open.org.

[17] *OASIS MQTT*. http://mqtt.org.

[18] *RTP*. https://searchnetworking.techtarget.com/definition/Real-Time-Transport-Protocol.

[19] *RPI*. https://www.raspberrypi.org/.

[20] Pablo Sebastián Rodríguez Eguren; Franco Chichizola; Enzo Rucci. «Análisis del Uso de un Cluster de Raspberry Pi para Cómputo de Altas Prestaciones». 2018.

[21] Khuram Shahzad. *Cloud Robotics and Autonomous Vehicles*. IntechOpen, 2016. DOI: 10.5772/64064.

[22] Israel García García. «ESTUDIO SOBRE VEHÍCULOS AÉREOS NO TRIPULADOS Y SUS APLICACIONES». 2017.

[23] L. Enrique Sucar; Giovani Gómez. «Visión Computacional». En: (2011).

[24] Domingo Mery. «Visión por computador». En: (2004).

[25] Yingke Feng; Jinmin Zhang; Siming Wang. «A New Edge Detection Algorithm Based on Canny Idea». En: (2017).

[26] *Opencv threshold*. https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html. Último acceso 30 de Noviembre 2018.

[27] Diego A. López García; F. Gómez Bravo. «Nuevas aportaciones en algoritmos de planificación para la ejecución de maniobras en robots autónomos no holónomos». 2011.

[28] Mika Rantanen. «Improving Probabilistic Roadmap Methods for Fast Motion Planning». 2014.

[29] Mark Foskey; Maxim Garber; Ming C. Lin; Dinesh Manocha. «A Voronoi-Based Hybrid Motion Planner». En: (2000).

[30] *Resolviendo Motion Planing*. http://gamma.cs.unc.edu/VORONOI/VPLAN/iros.pdf. Último acceso 30 de Noviembre 2018.

[31] *Planificación de Trayectorias para Robots Móviles*. http://webpersonal.uma.es/~VFMM/PDF/cap2.pdf. Último acceso 30 de Noviembre 2018.

[32] Roland Geraerts; Mark H. Overmars. «Sampling Techniques for Probabilistic Roadmap Planner». En: (2004).

[33] Ángel Manuel Montes Romero. «Planificación de caminos basada en modelo combinando algoritmos de búsqueda en grafo, derivados de RRT y RRT*». 2017.

[34] Zulfiqar Habib Iram Noreen Amna Khan. «A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms». En: (2016).

[35] Sertac Karaman; Emilio Frazzoli. «Sampling-based Algorithms for Optimal Motion Planning». En: (2011).

[36] Jonathan D. Gammell; Siddhartha S. Srinivasa; Timothy D. Barfoot. «Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristicg». En: (2014).

[37] *Javascript*. https://www.javascript.com/. Último acceso 30 de Noviembre 2018.

[38] *NodeJS*. https://nodejs.org/. Último acceso 30 de Noviembre 2018.

[39] *Python*. https://python.org/. Último acceso 30 de Noviembre 2018.

[40] Tim Bailey. «An Introduction to the C Programming Language and Software Design». En: (2005).

[41] *Node-RED*. https://nodered.org. Último acceso 30 de Noviembre 2018.

[42] *OpenCV*. https://opencv.org/. Último acceso 30 de Noviembre 2018.

[43] *FFMPEG*. https://www.ffmpeg.org/. Último acceso 30 de Noviembre 2018.

[44] *Parrot SDK*. https://developer.parrot.com/. Último acceso 30 de Noviembre 2018.

[45] *Flask*. https://palletsprojects.com/p/flask/. Último acceso 30 de Noviembre 2018.

[46] *MiniDrone Cargo*. https://www.parrot.com/us/minidrones/parrot-airborne-cargo-mars.

[47] *Parrot Bebop drone*. https://www.parrot.com/global/drones.

[48] *Network Time Protocol*. http://www.ntp.org/.

Anexo

Algoritmo 1 Conexión del dron en C: inicialización del dron

```
1  void droneInitialization(void){
2      drone->setPilotingPCMD(drone, 0, 0, 0, 0, 0, 0);
3      drone->setPilotingPCMDFlag(drone, 0);
4      drone->sendCameraOrientation(drone, (int8_t)-90, (int8_t)0);
5      drone->sendPictureSettingsVideoFramerate(drone, 0);
6      drone->sendPictureSettingsVideoResolutions(drone, 0);
7      drone->sendMediaStreamingVideoStreamMode(drone, 0);
8      drone->sendPilotingSettingsMaxAltitude(drone, (float)1.7);
9      drone->sendPilotingSettingsMaxTilt(drone, (float)5);
10     drone->sendSpeedSettingsMaxPitchRollRotationSpeed(drone, (float)80);
11     drone->sendSpeedSettingsHullProtection(drone, (uint8_t)1);
12 }
```

- En la línea 2 se indica al dron que no se desplace en ninguna dirección, seteando todos los valores de movimiento a 0.
- En la línea 3 se actualiza el flag de movimiento en 0, logrando mayor estabilidad.
- En la línea 4 se configura la orientación de la cámara hacia abajo.
- En la línea 5 se configuran los cuadros por segundo a 24.
- En la línea 6 se configura la resolución de la transmisión de video al modo de 480p.
- En la línea 7 se configura el modo de baja latencia para la transmisión de video.
- En la línea 8 se configura la máxima altura que puede volar el dron.

- En la línea 9 se configura el mayor grado de inclinación que puede utilizar el dron.
- En la línea 10 se configura la máxima velocidad de rotación.
- En la línea 11 se indica que el dron va a utilizar protección contra golpes.

Algoritmo 2 Conexión del dron en C: Ejecución de comandos

```
1  void executeCmd (char* cmd)
2  {
3    int time = 0;
4    parseCmd(&cmd, &time);
5    if (deviceController != NULL) {
6      if (strcmp(cmd, "land") == 0) {
7        drone->sendPilotingLanding(drone);
8      }
9      else if (strcmp(cmd, "takeOff") == 0) {
10       drone->sendPilotingTakeOff(drone);
11     }
12     else if (strcmp(cmd, "startStreaming") == 0) {
13       drone->sendMediaStreamingVideoEnable(drone, 1);
14     }
15     else if (strcmp(cmd, "stopStreaming") == 0) {
16       drone->sendMediaStreamingVideoEnable(drone, 0);
17     }
18     else if (strcmp(cmd, "up") == 0) {
19       drone->setPilotingPCMDGaz(drone, 30);
20     }
21     else if (strcmp(cmd, "down") == 0) {
22       drone->setPilotingPCMDGaz(drone, -30);
23     }
24     else if (strcmp(cmd, "forward") == 0) {
25       drone->setPilotingPCMDFlag(drone, 1);
26       drone->setPilotingPCMDPitch(drone, 50);
27     }
28     else if (strcmp(cmd, "backward") == 0 {
29       drone->setPilotingPCMDFlag(drone, 1);
30       drone->setPilotingPCMDPitch(drone, -50);
31     }
```

```
32      else if (strcmp(cmd, "left") == 0) {
33        drone->setPilotingPCMDFlag(drone, 1);
34        drone->setPilotingPCMDRoll(drone, -50);
35      }
36      else if (strcmp(cmd, "right") == 0) {
37        drone->setPilotingPCMDFlag(drone, 1);
38        drone->setPilotingPCMDRoll(drone, 50);
39      }
40      if (time > 0) {
41        sleep(time);
42        drone->setPilotingPCMD(drone, 0, 0, 0, 0, 0, 0);
43        drone->setPilotingPCMDFlag(drone, 0);
44      }
45    }
46  }
```

- En la línea 1 se setea el tiempo a realizar el comando en 0.
- En la línea 2 se separa el texto recibido en el comando y la variable que indica el tiempo.
- En la línea 3 se verifica que el controlador del cuadricóptero provisto por la librería esté disponible.
- En las estructuras de control siguientes, se pregunta por el comando recibido y se traduce a la función correspondiente provista por la librería oficial.
- En caso de haber recibido un valor de tiempo para el comando, este se ejecuta y se espera un tiempo hasta restablecer los valores de movimiento. Esto sucede en el bloque de la línea 39, permitiendo que se ejecute un comando de movimiento durante cierto período de tiempo.

Algoritmo 3 Procesamiento de imágenes: clase Filter

```
1  class Filter():
2      def __init__(self, frame):
3        self.gray = self._gray(frame)
4
5      def _gray(self, frame):
6          return cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
7
8      def _blur(self, t):
9          return cv2.GaussianBlur(self.gray, t, 0)
10
11     def _threshold(self, blur, first, second):
12         return cv2.adaptiveThreshold(blur, 255, 1, 1, first, second)
13
14     def _canny(self, blur, sigma=0.33):
15         v = numpy.median(blur)
16         lower = int(max(0, (1.0 - sigma) * v))
17         upper = int(min(255, (1.0 + sigma) * v))
18         return cv2.Canny(blur, lower, upper)
19
20     def filter_1(self):
21         blur = self._blur((3, 3))
22         return self._canny(blur)
23
24     def filter_2(self):
25         blur = self._blur((7, 7))
26         canny = self._canny(blur)
27         return self._threshold(canny, 3, 1)
```

- En la línea 2 se declara el constructor el cual recibe una imagen y la almacena en la clase convirtiéndola a escala de grises, en la línea 3.
- En la línea 5 se declara el método utilizado para convertir a escala de grises una imagen, haciendo uso de OpenCV.
- En la línea 8 se implementa un método que recibe el tamaño de máscara y aplica un filtro de suavizamiento gaussiano.
- En la línea 11, el método desarrollado aplica un filtro de umbral usando los parámetros recibidos: requiere una imágen con filtro de suavizamiento, el tamaño de máscara y una constante.

- En la línea 14 se declara el método que calcula los parámetros requeridos y llama a la función "Canny" de la librería.
- Luego en las líneas 20 y 24 se declaran dos combinaciones de filtros: la primera realiza un suavizado con máscara 3x3 y aplica el filtro "canny", la segunda realiza un suavizado con una máscara más grande de 7x7 y aplica un filtro canny seguido de un filtro de umbral.

Algoritmo 4 Procesamiento de imágenes: clase ShapeLabeler

```
1  class ShapeLabeler:
2      def get_approx(self, c, value, peri):
3          return cv2.approxPolyDP(c, value * peri, True)
4
5      def is_shape(self, approx, area, hullArea):
6          (x, y w, h) = cv2.boundingRect(approx)
7          solidity = area / float(hullArea)
8          keepDims = w > 12 and h > 12
9          keepSolidity = solidity > 0.9
10         minArea = area > 100
11         return keepDims and keepSolidity and minArea
12
13     def shape_atr(self, c):
14         peri = cv2.arcLength(c, True)
15         area = cv2.contourArea(c)
16         hullArea = cv2.contourArea(cv2.convexHull(c))
17         approx = self.get_approx(c, 0.04, peri)
18
19       if len(approx) >= 3 and self.is_shape(approx, area, hullArea):
20         if len(approx) == 3:
21             return approx, 'triangle'
22         elif len(approx) == 4:
23            return approx, 'square'
24         elif len(approx) > 4:
25             return approx, 'circle'
26       return approx, 'unknown'
```

- En la línea 2 se declara un método que aproxima el contorno reduciendo sus vértices.

- En la línea 5 se declara un método que determina si el contorno y sus atributos pasados por parámetro representan una figura geométrica de interés.
- De la línea 6 se obtienen cuatro valores de la imagen. De estos nos interesan los últimos dos: ancho(w) y alto(h).
- En la línea 7 se calcula la relación entre el área del contorno y la envolvente convexa.
- En las líneas 8, 9 y 10 se verifica que estas variables se encuentren dentro de ciertos umbrales apropiados.
- En la línea 13 se declara un método que dado un contorno verifica el tipo de figura que representa.
- En las líneas 14, 15, 16 y 17 se calculan atributos como el perímetro, área, envolvente convexa y aproximación del contorno.
- En la línea 19 se verifica que el contorno sea una figura geométrica llamando al método "is_shape()" y verificando que tenga como mínimo tres lados.
- A partir de la línea 20 se consulta por la cantidad de vértices que tiene la aproximación del contorno. Se retorna el contorno y su tipo de figura geométrica según corresponda.

Algoritmo 5 Procesamiento de imágenes: verificación de contorno interior

```
1  inner_shape = self.check_inner_shape(hierarchy_tree, 'circle')
2  {...}
3  def check_inner_shape(self, child, id):
4          if child == -1: return None
5
6          approx, shape = ShapeLabeler.shape_atr(self.contours[child])
7          if shape == id:
8              return Contour(approx)
9          else:
10              self.check_inner_shape(self.hierarchy[child][1], id)
```

- En la línea 1 se muestra a modo de ejemplo la llamada al método para detectar si el contorno en cuestión presenta una figura interior de forma circular.

- En la línea 3 se verifica que el contorno tenga algún primer hijo. En caso de que el valor de la jerarquía sea -1 implica que este contorno no tiene hijos. Además, sirve para dar fin a la recursión explicada más adelante.
- En la línea 5 se verifica que el primer hijo del contorno sea una figura a tener en consideración.
- En la línea 6 se verifica que este contorno hijo sea un círculo.
- En la línea 7, cumplidas estas condiciones, se retorna el contorno hijo que representa al círculo interior del contorno padre.
- En la línea 10, en caso de que no se cumplan esas condiciones, se llama de forma recursiva al método pasando como parámetro al hermano del contorno hijo. Estas llamadas se repiten hasta que se encuentre un contorno con las condiciones o hasta que se haya procesado al último hermano.

Algoritmo 6 Algoritmo RRT

```
1  def plan(self):
2      # Flag que indica si encontró el destino.
3      goal_flag = False
4
5      # Lista de nodos del path.
6      self.node_list = [self.start]
7
8      best_path_len = float('inf')
9      tmp_iterations, path, iterations = [], [], []
10
11     start = time.time()
12
13     while (time.time() - start) < self.max_time:
14
15         # Búsqueda aleatoria
16         if random.randint(0, 100) > self.goal_sample_rate:
17             rnd = [random.uniform(self.rand_x[0], self.rand_x[1]), random.ur
18                 self.rand_y[0], self.rand_y[1])]
19         else:
20             rnd = [self.goal.x, self.goal.y]
21
22         # Busca el nodo más cercano.
```

```
23          nind = self.get_nearest_list_index(self.node_list, rnd)
24
25          # Expande el árbol.
26          nearest_node = self.node_list[nind]
27          theta = math.atan2(rnd[1] - nearest_node.y, rnd[0] - nearest_node.x)
28
29          new_node = copy.deepcopy(nearest_node)
30          new_node.x += self.expand_dis * math.cos(theta)
31          new_node.y += self.expand_dis * math.sin(theta)
32          new_node.parent = nind
33
34
35          # Guarda los puntos de inicio y fin de la linea recien generada
36          iterations.append([nearest_node, new_node])
37
38          if not self.collision_check(new_node):
39              continue
40
41          self.node_list.append(new_node)
42
43          # Verifica si se llegó al destino.
44          dx = new_node.x - self.goal.x
45          dy = new_node.y - self.goal.y
46          d = math.sqrt(dx**2 + dy**2)
47
48          if d <= self.expand_dis:
49              tmp_path = self.get_final_path()
50              len_tmp_path = self.get_path_length(tmp_path)
51
52              if len_tmp_path < best_path_len:
53                  best_path_len = len_tmp_path
54                  path = tmp_path
55                  iterations = tmp_iterations
56
57              goal_flag = True
58
59              if self.optimal:
60                  self.node_list = [self.start]
61                  tmp_iterations = []
62              else:
```

```
63                  break
64
65
66      if not goal_flag:
67          raise GoalNotReachedException
68
69      return path, iterations
```
