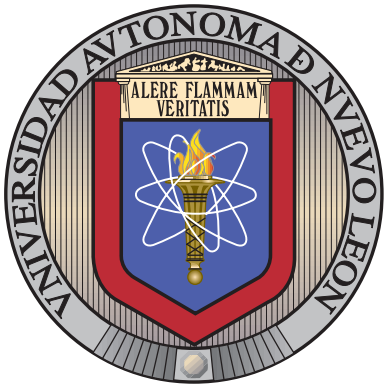
Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Estudios de Posgrado



Sistemas de percepción mínimos para robots colectivos RAOI basados en Aprendizaje Profundo

por

Erik Ricardo Palacios Garza

como requisito parcial para obtener el grado de

maestría en ciencias de la ingeniería eléctrica

junio 2021

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Estudios de Posgrado

Sistemas de percepción mínimos para robots colectivos RAOI basados en Aprendizaje Profundo

por

Erik Ricardo Palacios Garza

como requisito parcial para obtener el grado de

maestría en ciencias de la ingeniería eléctrica

junio 2021

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica
Subdirección de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la Tesis "Sistemas de percepción mínimos para robots colectivos RAOI basados en Aprendizaje Profundo", realizada por el alumno Erik Ricardo Palacios Garza, con número de matrícula 1585199, sea aceptada para su defensa como requisito para obtener el grado de Maestría en Ciencias de la Ingeniería Eléctrica.

El Comité de Tesis

Dr. Luis Martín Torres Treviño
Director

Dr. Juan Ángel Rodríguez Liñán
Revisor

Dr. Jesús Emmanuel Gómez Correa
Revisor

Vo. Bo.

Dr. Simón Martínez Martínez
Subdirector de Estudios de Posgrado

149

San Nicolás de los Garza, Nuevo León, julio 2021

Ciudad Universitaria Pedro de Alba s/n, C.P. 66455. A.P. 076 Suc. "F"
San Nicolás de los Garza, Nuevo León, México. Tels: (81) 8332 0903 /
Conm.: 8329 4020 / Fax: (81) 8332 0904

Agradecimientos

A mi asesor de tesis, el Dr. Luis Martín Torres Treviño, por su apoyo y enseñanza durante el desarrollo de esta tesis.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT), la Universidad Autónoma de Nuevo León (UANL) y la Facultad de Ingeniería Mecánica y Eléctrica (FIME), por otorgarme la beca y los medios para realizar este proyecto.

A los profesores del Posgrado en Ingeniería Eléctrica por sus enseñanzas durante mis estudios.

A mi madre y mis hermanas por su apoyo, guía y enseñanzas.

Índice general

Índice de figuras

Índice de tablas

Nomenclaturas

RAOI	Repulsión, Atracción, Orientación e Influencia.
CNC	Convolutional Neural Networks.
YOLO	You Only Look Once.
SSD	Single Shot Detector.
ResNet	Residual Network.
IoU	Intersection Over Union.
ZOR	Zone of Repulsion.
ZOO	Zone of Orientation.
ZOA	Zone of Attraction
ZOI	Zone of Influence

Resumen

Erik Ricardo Palacios Garza.

Candidato para obtener el grado de Maestría en Ciencias de la Ingeniería Eléctrica.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: Sistemas de percepción mínimos para robots colectivos RAOI basados en Aprendizaje Profundo.

Número de páginas: 94.

Objetivos y método de estudio: Con la integración de técnicas de inteligencia artificial en la visión computacional principalmente el aprendizaje profundo con la implementación de redes neuronales convolucionales, se han podido implementar sistemas de visión altamente complejos los cuales incluyen el reconocimiento, clasificación y detección de múltiples objetos en una misma imagen, sin embargo en la robótica de enjambres es de práctica común utilizar como sistema de percepción conjuntos de sensores de señales simples, por ejemplo, sensores infrarrojos, ultrasónicos, sensores de luz, entre otros, limitando el desarrollo de comportamientos cada vez más completos y adaptables a aplicaciones reales.

En la presente tesis, se propone un sistema de percepción basado en aprendizaje profundo y sensores, el cual se encargará del reconocimiento clasificación y detección de estímulos de influencia, integrantes del enjambre y obstáculos. Además de su implementación en diferentes prototipos de diferentes capacidades computacionales. Como objetivo principal se tiene diseñar un sistema de percepción basado en aprendizaje profundo que permita a un enjambre de robots reconocer robots vecinos, obstáculos y marcas o estímulos de influencia. Para esto se debe llevar a cabo el desarrollo de diferentes algoritmos de visión basados en aprendizaje profundo y su implementación en un sistema físico para demostrar la utilidad de estos. Para

cumplir con los objetivos anteriormente mencionados se pretende utilizar modelos de aprendizaje profundo, estos modelos estarán basados en redes neuronales convolucionales por su capacidad de extracción de características y generalización de conceptos, en contrario a diferentes instrumentos de visión artificial que presentan diversas restricciones en las imágenes capturadas, algunas de las estructuras mas utilizadas en la literatura son las redes ResNets y modelos de detección diversos, por ejemplo, YOLO (you only look once) y SSD (single shot detector), por mencionar algunos. Finalmente para la plataforma de robots se pretende la utilización de configuraciones diferenciales por su simplicidad y la movilidad que presentan, así como la utilización de ordenadores como *Raspberry pi®* para la ejecución del algoritmo completo.

Contribuciones y conclusiones: Con la experimentación realizada se puede decir que la implementación de un sistema de percepción basado en aprendizaje profundo permite incrementar el numero y tipo de estímulos y objetos que puede detectar el enjambre y al mismo tiempo poder tomar decisiones en base a estos. Se pudo observar también el aumento de la complejidad a la hora de desarrollar diferentes comportamientos, al tener mas variables a manejar utilizando este tipo de sistemas de percepción. La principal contribución de este trabajo consiste en ampliar el número y las características de los estímulos que recibe el enjambre para el desarrollo de comportamientos cada vez más complejos, además de lograr realizar esta implementación en una arquitectura no centralizada.

Firma del asesor: _____

Dr. Luis Martín Torres Treviño

Capítulo 1

INTRODUCCIÓN

1.1 Motivación

La robótica de enjambres está enfocada en realizar tareas que para un solo robot sería complicado de completar o tendría una baja eficiencia realizándola, ésta propone el uso de múltiples robots de características simples Navarro y Matía (2013) los cuales colaboren para el cumplimiento de tareas como búsqueda, agrupamiento, traslado de objetos entre otras actividades. Cuando se menciona que los robots deben ser de características simples quiere decir que los elementos que conformen a éstos no deben de ser costosos o complejos en su funcionamiento, esta misma característica permite el fácil desarrollo de estas plataformas y al mismo tiempo limita el desarrollo de comportamientos que resuelvan tareas cada vez más complejas.

Teniendo este concepto minimalista, la utilización de sensores sencillos como infrarrojos, ultrasónicos, y de iluminación por mencionar algunos, se han podido desarrollar comportamientos sumamente complejos como depredador - presa o traslado de objetos.

Ahora con el desarrollo de sistemas de visión basados en aprendizaje profundo, se han podido implementar en procesadores tamaños compatibles con las dimensiones comunes de un robot miembro de un enjambre, por lo que, sin pérdida de simplicidad

en el diseño de un enjambre de robots, se pueden implementar estos sistemas de visión en la percepción de cada robot logrando además alcanzar sistemas centralizados en su totalidad.

El desarrollo de comportamientos en los enjambres de robots en los cuales se basa el presente trabajo consta del control de cuatro parámetros elementales, repulsión, orientación, atracción e influencia, donde este último trata de señales que percibe el enjambre y lo estimulan para realizar ciertas acciones, con los sistemas de percepción basados en sensores de una sola magnitud física el número y características de los estímulos de influencia es reducido a la percepción de los sensores simples mencionados y a su vez limita el desarrollo de comportamientos nuevos.

Por lo que con la implementación de un sistema de visión es posible ampliar de manera drástica el número y la diversidad de estímulos de influencia y como consecuencia favorecer al desarrollo de más comportamientos en el enjambre, sin embargo, existen diferentes tipos de sistemas de visión en general los basados en procesamiento de imágenes y basados en aprendizaje profundo, los primeros presentan la limitante de tener que desarrollar un algoritmo de procesamiento de imágenes específico para cada objeto que se desee clasificar o detectar, mientras que los sistemas basados en aprendizaje profundo tienen la flexibilidad de poder adaptar un mismo modelo a diferentes objetos, es por esto que la aportación del presente trabajo radica en la implementación de un sistema de visión basado en aprendizaje profundo logrando aumentar las capacidades de percepción de los enjambres sin la necesidad de modificar la estructura del sistema de visión desarrollado.

1.2 Antecedentes

La robótica de enjambres trata de resolver problemas complejos usando un conjunto de robots de baja complejidad y bajo costo, como por ejemplo, tareas exploración, localización, construcción etc. Una de las ventajas de trabajar con estas

configuraciones es que pueden cumplir con la tarea asignada aun con la pérdida o inclusión de miembros nuevos al enjambre, así como disminuir la complejidad de los algoritmos necesarios para la realización de diversas tareas, al lograr que emerjan comportamientos propios y autónomos del enjambre.

En la mayoría de los casos, como se menciona en Ordaz-Rivas *et al.* (2019b) estos robots son construidos con un cierto nivel de percepción, el cual consiste en un conjunto de sensores que les permiten tener cierta información del espacio que los rodea, los obstáculos que se podrían encontrar, u objetos con los cuales interactuar, por ejemplo los robots utilizados en este mismo artículo utilizan sensores de proximidad y luz para medir la distancia con robots vecinos u objetos, además de obtener información con la luz del lugar donde se encuentren, con esta percepción asignada a cada robot se han obtenido resultados sorprendentes como los mencionados en el artículo, sin embargo, tienen la limitante de solo obtener información local de medio ambiente restringiendo la diversificación de tareas que ya realizan.

En Schmickl *et al.* (2006) se menciona una problemática de percepción similar, donde la comunicación a corta distancia, representa un reto que propicia el desarrollo de algoritmos más complejos para la robótica de enjambres, presentando la percepción asignada a cada robot como restrictiva.

Existen algunos otros sistemas de percepción para robótica de enjambres más complejos que los sensores antes mencionados, pero que a su vez proporcionan una mayor capacidad de trabajo a cada robot y al sistema completo, como es el caso de Giusti *et al.* (2012), donde utilizan gestos de manos para manipular el comportamiento grupal de los robots, este reconocimiento de gestos manuales está basado en un preprocesado de la imagen capturada de la mano, una extracción de características y un clasificador basados en SVM´s. Claramente agregando un sistema de visión de este nivel se puede observar que se tienen comportamientos que considerando un sistema de visión por sensores serian complicados de obtener.

En el procesamiento de imágenes existen mascaras prediseñadas para la extrac-

ción de ciertas características, pero debido a que muchas veces estas mascaras no son suficientemente específicas para encontrar las características deseadas se recurre a las CNN las cuales buscan mascaras óptimas para la extracción de características específicas de una imagen y así poder hacer reconocimiento, clasificación o detección de objetos en imágenes Zhao *et al.* (2019), las CNN constan de una parte donde se utilizan convoluciones y se extraen diferentes atributos de la imagen de entrada, una vez obtenidos estos atributos son procesados por una parte densa que realiza un reconocimiento, clasificación o detección de los objetos en la imagenZhang (2016) para proporcionar una salida que pueda ser interpretada por el usuario de la red.

En Nagi *et al.* (2012) también se plantea la idea de usar redes neuronales convolucionales, dichas redes tienen la ventaja de tener una alta precisión en la tarea de visión que se les asigne, además poseer gran robustez a las condiciones ambientales donde se capture la imagen a analizar, en la actualidad existe equipo en donde se pueden embeber para ser utilizadas en sistemas robóticos móviles, la desventaja ahora presente es el poder de cómputo que se necesita para su entrenamiento, donde es necesario tanto un gran número de imágenes como computadoras suficientemente potentes para ejecutar el entrenamiento que requieren.

En este último artículo se propone un modelo de CNN donde cambian la configuración densa comúnmente usada (perceptrón) por una SVM, dando como resultado una mejora en la precisión de la CNN. Con esta implementación en un enjambre de robots se podrían abrir una serie de tareas y comportamientos más complejos o mejorar las tareas que ya se realizan en la actualidad. Sin embargo, si en lugar de implementar una CNN entrenada para detectar gestos realizados con las manos, se entrenan para poder percibir su medio ambiente, se podría dar lugar a comportamientos más eficientes y más autónomos que podrá realizar el enjambre sin la necesidad de intervención humana, un ejemplo muy tangible de estas nuevas alternativas a estos comportamientos es la solución a una problemática planteada en Ordaz-Rivas *et al.* (2019a) donde se menciona que con sensores sencillos como los antes mencionados un robot no tiene la posibilidad de conocer la orientación de

sus vecinos, con la implementación de una CNN se puede abrir esta capacidad a los robots.

Finalmente, La utilización de sistemas de percepción basados en aprendizaje profundo permite la adaptación de algoritmos de cognición en el comportamiento del enjambre, como es el caso de Jin *et al.* (2020), donde proponen un sistema de visión utilizando aprendizaje profundo obteniendo datos suficientes del entorno que rodean a los robots para lograr el entrenamiento de un algoritmo por refuerzo y poder así generar actividades nuevas en el enjambre.

1.3 Planteamiento del problema

Para el desarrollo de robots colectivos RAOI (swarm robotics) se requiere la creación de sistemas de percepción, que logren distinguir obstáculos, otros robots, y estímulos de influencia asociados con factores diversos, por lo que puede ser compleja la tarea de diseñar un sistema de percepción específico para cada comportamiento que se requiera en el enjambre.

Por lo tanto, es indispensable diseñar sistemas de percepción más generales al reconocimiento y detección de estímulos de influencia, robots vecinos y obstáculos para el desarrollo de comportamientos cada vez más complejos y con aplicaciones más aptas al mundo real.

1.4 Hipótesis

Es posible diseñar un sistema de aprendizaje profundo en un esquema de preentrenamiento que permita la detección de obstáculos y robots, así como el incremento del número de estímulos de influencia que puede reconocer un enjambre de robots.

1.5 Objetivos

1.5.1 Objetivo General

Diseñar un sistema de percepción basado en aprendizaje profundo que permita a un enjambre de robots reconocer robots vecinos, obstáculos y marcas o estímulos de influencia.

1.5.2 Objetivos Particulares

- Objetivo 1.1: Desarrollo de esquemas de percepción generales basados en aprendizaje profundo adaptables a sistemas embebidos.
- Objetivo 1.2: Diseñar sistema de percepción apto para un enjambre de robots.
- Objetivo 1.3: Implementación del sistema de percepción en plataforma prototipo de robots.

1.6 Metodología

- Desarrollo de redes convolucionales capaces de reconocer, clasificar y detectar objetos, estas redes serán empleadas en versiones reducidas para su implementación en sistemas embebidos, estas redes podrán constar desde perceptrones hasta redes detección actuales como YOLO o arquitecturas SSD.
- Obtención de una base de datos utilizando perspectivas de la visión del robot diseñando un algoritmo de captura y almacenamiento de imágenes dentro de un miembro del enjambre, esta base de datos debe de contener imágenes del

robot estático y en movimiento para la mejor generalización de las redes a utilizar.

- Entrenamiento de los modelos de aprendizaje profundo implicados en el sistema de percepción, utilizando un computador externo a la tarjeta de cada robot, realizando una transferencia de aprendizaje.
- implementación en robots físicos móviles para experimentación, realizando tareas propias de un enjambre de robots.

1.7 Contribuciones

Las contribuciones de esta tesis son las siguientes.

- La apertura de la utilización de métodos de aprendizaje profundo aplicados a sistemas de percepción no centralizados de enjambres de robots.
- Desarrollo de sistema de percepción basado en aprendizaje profundo apto para enjambres de robots no centralizados.
- La extensión de la variedad de estímulos de influencia que puede detectar un enjambre de robots.

1.8 Organización de la Tesis

En el capítulo dos se presentan algunos temas preliminares que sirven como base para el trabajo desarrollado. Se da una descripción de manera general de las redes neuronales y las estructuras utilizadas en la visión artificial. Se presenta la estructura de las redes de detección SSD (Single Shot Detector), YOLO (You Only

Look Once) además de la red de clasificación ResNet (Residual Networks). Se presenta el modelo propuesto por Lain Couzin para el movimiento de seres en grupo. Además, se muestra el controlador utilizado en cada robot y los sistemas embebidos más comunes para el manejo de redes neuronales y controladores.

En el capítulo 3 se describe el comportamiento seleccionado. Se da una descripción del funcionamiento de cada miembro de la pandilla en la plataforma de robots implementada, así como una descripción de las estructuras de las redes neuronales utilizadas en el sistema de visión y las bases de datos utilizadas en cada una de ellas.

En el capítulo 4 se presentan los resultados obtenidos en cada prototipo. se describen las condiciones de experimentación, así como los parámetros utilizados en cada prueba, así como los resultados del entrenamiento de cada red neuronal, finalmente algunas imágenes de los experimentos con la plataforma de robots.

En el capítulo 5 se muestran las conclusiones de este trabajo, así como los trabajos futuros.

Capítulo 2

Preliminares

2.1 Redes Neuronales

Las redes neuronales artificiales son modelos computacionales de redes neuronales biológicas, las cuales tratan de imitar la manera en el cerebro humano aprende y toma decisiones, cada una de estas redes constan de unidades (neuronas) las cuales segmentan y procesan la información, estas neuronas reciben información mediante conexiones con otras neuronas estas conexiones que transportan dicha señal son el análogo de las dendritas en las redes neuronales biológicas, después de recibir estas señales son procesadas mediante pesos en los cuales se basa el aprendizaje de estas redes, por último la información procesada es transmitida hacia otras neuronas para seguir siendo procesada un ejemplo de estas neuronas se muestra en la figura 2.1.

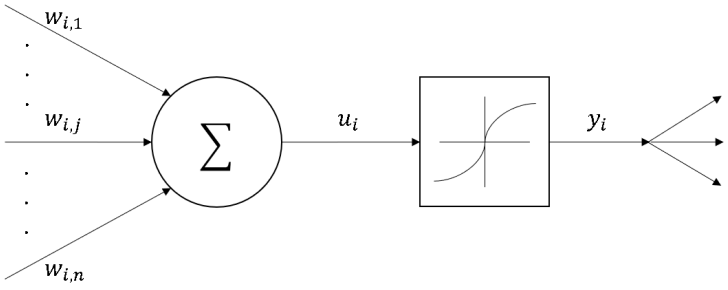


Figura 2.1: Modelo de neurona artificial

2.1.1 Redes Convolucionales

A partir de estas neuronas artificiales se pueden crear estructuras que puedan procesar y aprender diferentes tipos de información ya sea numérica, gráfica, sonidos, grafos entre otros. El presente trabajo se centrará en procesar información gráfica mediante la captura de imágenes con una cámara de vídeo, para lograr esto se utilizan redes neuronales convolucionales.

$$C = f\left(I * K + b\right) \tag{2.1}$$

$$C(i,j) = f\left(\sum_{u=0}^{Fi}\sum_{v=0}^{Co} I(i+u, j+v)\cdot K(u,v) + b\right) \tag{2.2}$$

Donde:

- I: Es la imagen a la que se le aplicará la convolución.
- K: Filtro convolucional.
- C: Imagen Convolucionada.
- b: Sesgo
- i, j: Posiciones del centro del filtro en la imagen a convolucionar.
- Fi: Filas de la imagen.
- Co: Columnas de la imagen.

Las redes neuronales convolucionales logran la extracción de características específicas deseadas en imágenes, mediante convoluciones de filtros desarrollados por la misma red con la imagen de entrada correspondiente a una determinada capa de la red estos filtros se encuentran en las flechas punteadas de la figura 2.2 y se pueden apreciar en la figura 2.3, este proceso se puede observar en las ecuaciones 2.1

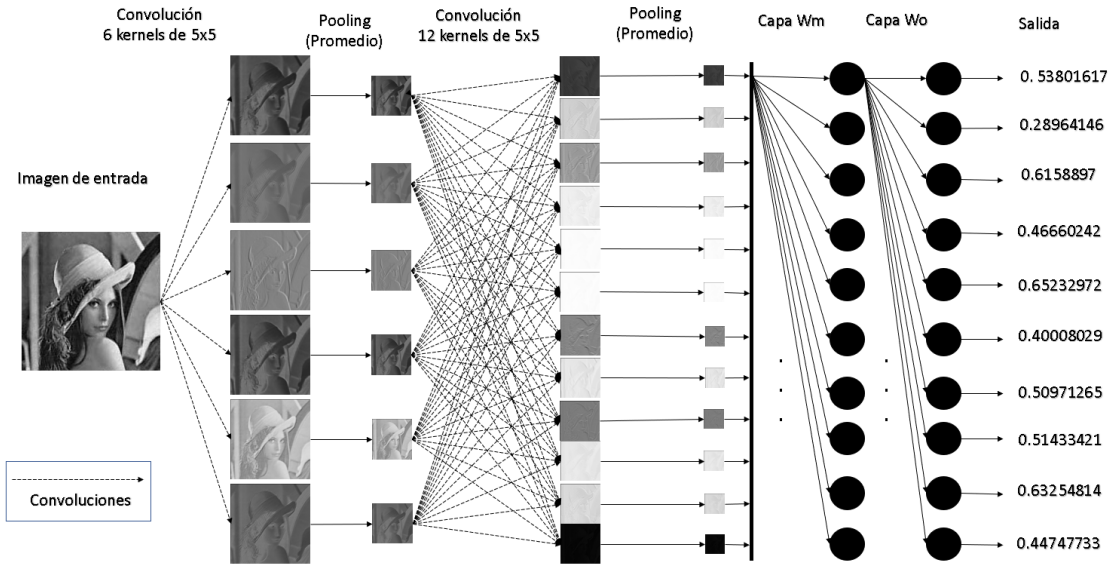


Figura 2.2: Estructura básica de una red neuronal convolucional

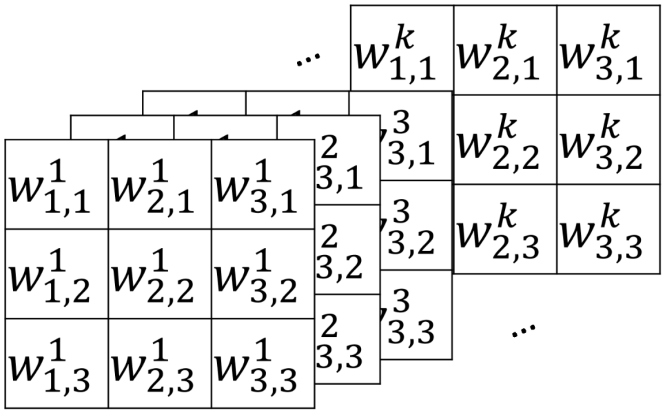


Figura 2.3: Filtros convolucionales

y 2.2 y en la figura 2.2 se puede observar una estructura básica de una red neuronal convolucional.

$$S(i,j) = \frac{1}{4}\sum_{u=0}^{1}\sum_{v=0}^{1} C(i+u, j+v) \tag{2.3}$$

Además de convoluciones se realizan otras operaciones básicas comúnmente usadas dentro de este tipo de redes, dentro de las que se pueden observar en la figura 2.2 se encuentra el pooling, el cual ayuda a disminuir las dimensiones de la imagen de entrada en cada capa de la red, de esta manera simplificar la información para su mejor procesamiento por capas siguientes y disminuir la carga computacional, esta operación consiste en mapear un filtro a través de la imagen de entrada de la capa y realizar alguna operación que reduzca la dimensión de la imagen por ejemplo tomar el mínimo o el máximo de los píxeles que toque el filtro a medida que avance, o tomar el promedio de estos píxeles como se puede apreciar en la ecuación 2.3 donde S Es la imagen redimensionada.

Al momento de realizar convoluciones la imagen de entrada reduce su tamaño al no considerar las orillas debido al tamaño del filtro, esto produce una pérdida de información que en ciertos casos específicos puede ser de importancia, en estos caso se hace uso de la operación padding, la cual agrega un marco de ceros a la imagen de entrada para poder convolucionar el filtro con cada pixel, esta operación se muestra en las figuras 2.4 y 2.5

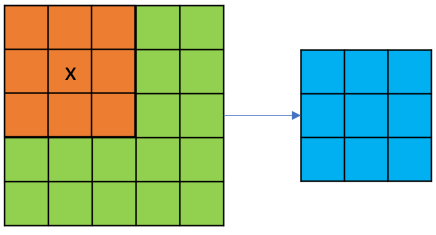


Figura 2.4: Convolución sin padding

$$v(k) = C(i,j) \tag{2.4}$$

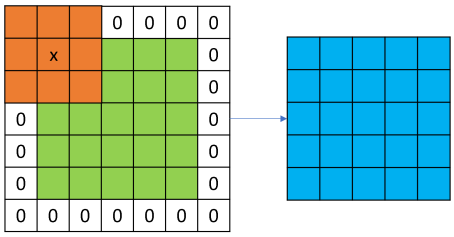


Figura 2.5: Convolución con padding

Por último después de la última capa perteneciente a operaciones básicas de las redes convolucionales, se tiene un conjunto de imágenes las cuales contienen información de las características del objeto que se requiere reconocer, para poder interpretar estas características se requiere integrar una parte densa que pueda decodificarlas y dar una salida fácil de interpretar, sin embargo, la entrada de la red densa es de una sola dimensión y la salida de cualquier capa convolucional es de 3 dimensiones, para poder adaptar la salida de la última capa de la parte convolucional a la entrada de la parte densa se utiliza la operación de vectorización, la cual apila las filas de cada imagen en el arreglo de 3 dimensiones, esta operación se puede apreciar en la ecuación 2.4 donde $i = 0, 1, 2...Fi$, $j = 0, 1, 2...Co$.

2.1.2 Redes Residuales

Un problema que se presenta en las redes neuronales convolucionales es que a medida que avanza el entrenamiento existe una pérdida de información, ya que el gradiente se va desvaneciendo entre más profunda sea la red, es decir, más capas tenga, esto limita las capacidades de las redes neuronales convolucionales ya que impide el cambio adecuado en los pesos de la red, incluso podría detener el entrenamiento.

Para resolver este problema se proponen en He *et al.* (2016) conexiones residuales, estas conexiones transmiten información de la entrada de una cierta capa a la salida de otra, ayudando a la propagación del gradiente, evitando el desvanecimiento mencionado se puede ampliar la profundidad de las redes neuronales convolucionales

convirtiéndose en redes neuronales residuales.

En la figura 2.6 se puede apreciar una conexión residual, en la mayoría de las ocasiones las dimensiones de la entrada x no concuerdan con las de la salida de la capa de activación $F(x)$ por lo que se agrega una capa convolucional en la capa residual para ajustar esta diferencia de dimensiones.

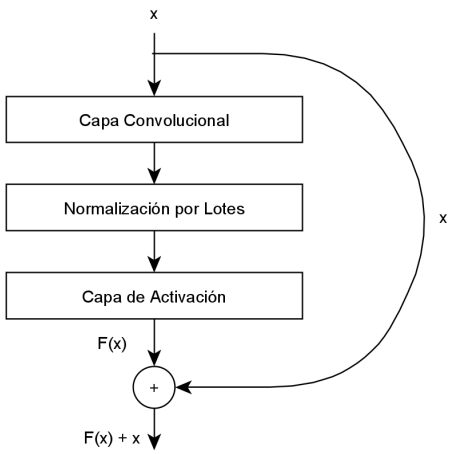


Figura 2.6: Ejemplo de conexión residual

Al conjunto de capas que salta una conexión residual se le conoce como bloque residual, existen diferentes bloques residuales, sin embargo, hay capas que son constantes en la mayoría de los casos por ejemplo capas convolucionales, capas de activación, capas de pooling y además se puede observar en la figura 2.6 una capa con la operación de normalización por lotes. Este tipo de normalización ayuda a mejorar la velocidad de entrenamiento y reduce la importancia de la forma en que se inicializan los pesos en la red, entre otras ventajas.

$$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.5}$$

$$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\beta)^2 \tag{2.6}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}} \tag{2.7}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i) \tag{2.8}$$

Esta normalización es del tipo estándar, en donde se obtiene el promedio y la desviación estándar del lote de imágenes en proceso como se observa en las ecuaciones 2.5 y 2.6, la operación completa se puede apreciar en la ecuación 2.7, por último en la ecuación 2.8 se presenta una des normalización que se utiliza en caso que alguna función de activación lo requiera.

Donde:

- x_i: Es la salida de la capa donde se aplicará la normalización.
- m: Tamaño del lote de imagenes.
- $hatx_i$: Salida normalizada.
- γ: Factor de escala.
- β: Factor de desplazamiento.
- $haty_i$: Salida normalizada escalada y desplazada.

Utilizando un conjunto de bloques residuales se puede formar una red residual, las cuales se caracterizan por ser de gran profundidad y por consecuencia de buena precisión a la hora de realizar tareas de predicción, un ejemplo del mejoramiento de la eficiencia usando conexiones residuales se puede apreciar en la figura 2.7 donde se observa en la red de la izquierda la configuración de la red Plain de 34 capas y a su derecha la configuración de una ResNet de 34 capas, se puede apreciar que las estructuras son prácticamente idénticas por excepción de las conexiones residuales,

siendo estas las que mejoran la precisión de la red en un 12.29 % de acuerdo con He *et al.* (2016).

En la red residual de la figura 2.7 se pueden observar 16 bloques residuales cada uno con dos capas de convolución, cada bloque residual consta de un color el cual se distingue por el tamaño de la imagen que se está procesando y el número de filtros que se está utilizando, en los bloques de color morado se tiene un tamaño de imagen de 56 x 56 pixeles y 64 filtros, en los bloques de color verde se tiene un tamaño de imagen de 28 x 28 con 128 filtros, en los de color naranja se tienen imágenes de 14 x 14 y 256 filtros, por último se tienen los bloques de color azul los cuales constan de un tamaño de imagen de 7 x 7 y 512 filtros, al final de la red se puede observar el bloque que representa la parte densa de clasificación.

2.1.3 Redes de detección

Pese a la gran importancia y utilidad de las redes de clasificación existen ciertas limitaciones que en muchas aplicaciones son muy importantes, sobre todo en tareas relacionadas con la robótica. Dentro de estas limitaciones se encuentran principalmente el poder clasificar dos objetos dentro de una misma imagen y el poder obtener la ubicación de los objetos que se clasifiquen en la imagen, esto es de gran utilidad a la hora que se desee interactuar con determinado objeto utilizando un robot, por ejemplo un robot que se encargue de la calidad de frutas en una banda transportadora necesitará poder clasificar varios objetos en su campo de visión, así como obtener la posición de las frutas que no cumplan con los estándares de calidad que se soliciten. Cuando se logra dar la ubicación de los objetos clasificados en una imagen la red de clasificación se convierte en una red de detección.

Para lograr la detección de objetos en esta tesis se utilizó el procedimiento que se presenta a continuación.

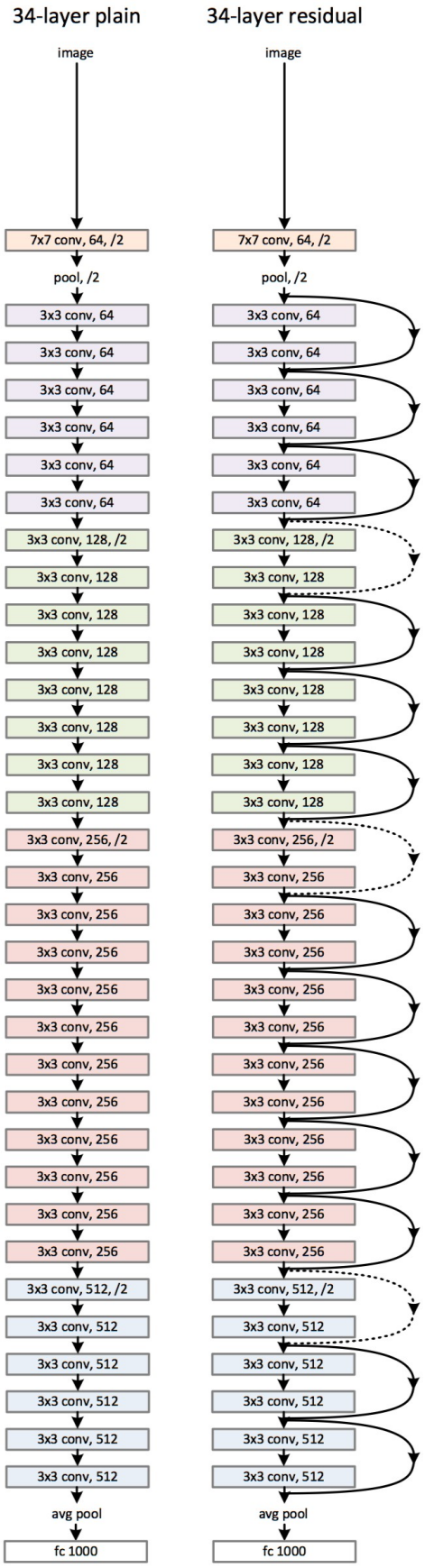


Figura 2.7: **Izquierda :** Plain - 34, **derecha :** Resnet – 34 He *et al.* (2016)

2.1.3.1 Celdas

Para analizar la imagen en todas sus áreas esta se divide en celdas, cada una de estas celdas dará cierta información acerca de lo que posiblemente se encuentre dentro de ellas, para llegar a dividir la imagen en estas celdas es necesario que esta pase por una serie de convoluciones modificando sus dimensiones y extrayendo al mismo tiempo la información necesaria para la detección.

Como ya se mencionó antes las salidas de las capas de convolución son de 3 dimensiones, las primeras dos representan el largo y el ancho de la matriz de salida de determinada capa y la tercera el número de filtros que hay en esa salida, en este caso la salida de la última capa convolucional utilizada en la estructura de la red de detección constará de las celdas antes mencionadas y la información requerida de la detección, siendo las primeras dos dimensiones de la salida la posición de cada celda, y la tercera representará cada uno de los datos extraídos.

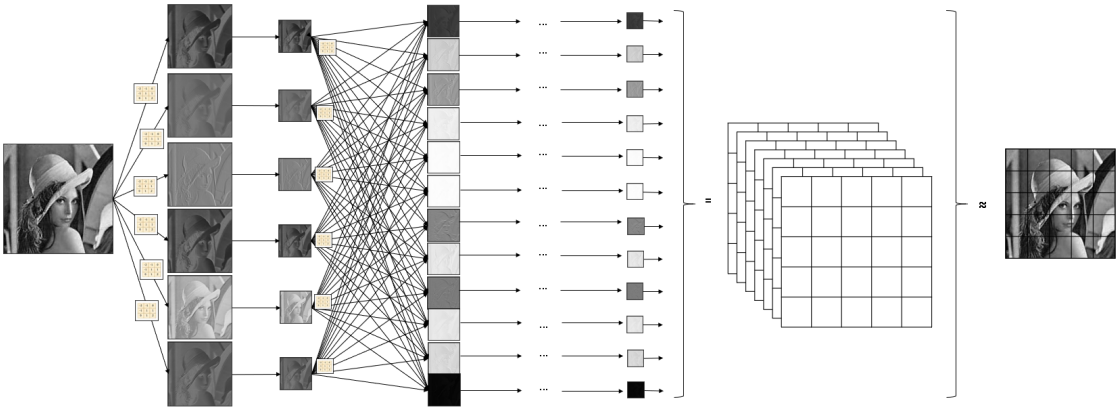


Figura 2.8: Formación de celdas de detección

2.1.3.2 Cajas delimitadoras

Cada una de las celdas anteriormente mencionadas genera una serie de cajas delimitadoras o por su nombre en inglés "bounding boxes", cada una de estas bounding boxes tratarán de enmarcar de la manera más precisa posible los objetos de

interés en la imagen de entrada.

Los datos requeridos para poder dibujar estas bounding boxes son las coordenadas de su centro así como su ancho y largo, sin embargo, como se menciona en Redmon y Farhadi (2018); Redmon *et al.* (2016) el entrenamiento para el cálculo de estos datos se vuelve complicado además de limitar el número de bounding boxes que puede generar cada celda, por lo que en Ren *et al.* (2015) se propone el uso de cajas ancladas o por su nombre en inglés anchor boxes, éstas son un tipo de cajas que se quedan fijas en el centro de cada una de las celdas anteriormente mencionadas y sirven para sustituir el cálculo de coordenadas por el cálculo de offsets de las predicciones de las bounding boxes con respecto a las anchor boxes.

Con este cambio en la forma de predecir las bounding boxes se puede incrementar sustancialmente el número de cajas utilizadas en cada celda, pasando de unas pocas cajas a cientos de ellas por imagen.

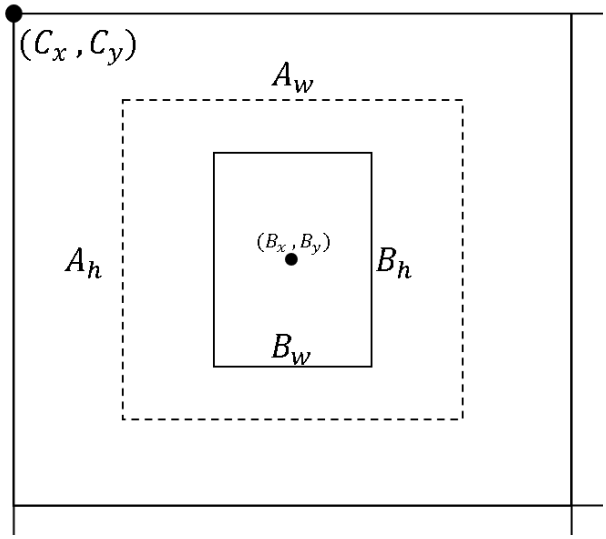


Figura 2.9: Visualización de una bounding box y una anchor box

En la figura 2.9 se puede apreciar un ejemplo de anchor box denotada por líneas punteadas siendo A_w y A_h su ancho y su altura, además de un ejemplo de bounding box donde B_w, B_h, B_x y B_y son su ancho, altura y coordenadas de su centro respectivamente.

Sin embargo, la red neuronal no entregará los valores de B_w, B_h, B_x y B_y, si no que entregará los offsets que existan entre la bounding box y la anchor box denotados por t_x, t_y, t_w, t_h, por lo que para el cálculo de los datos de la bounding box se realizan las siguiente operaciones.

$$B_x = \sigma(t_x) + C_x \tag{2.9}$$

$$B_x = \sigma(t_y) + C_y \tag{2.10}$$

$$B_w = A_w e^{t_w} \tag{2.11}$$

$$B_h = A_h e^{t_h} \tag{2.12}$$

De igual manera se codifican los datos despejando las ecuaciones para entrenar la red con los valores t_x, t_y, t_w, t_h correspondientes a la bounding box que demos como dato de entrada a la red. Con esto se describieron 4 de los datos que entrega la red de detección.

2.1.3.3 Datos de salida de la red

Como ya se mencionó anteriormente la salida de la red consta de tres dimensiones donde las primeras dos representan la ubicación de cada celda, y la tercera el número de datos diferentes que entrega la red, en Çajas Delimitadoras"se mencionaron 4 de estos datos, t_x, t_y, t_w y t_h, pero además la red entrega una evaluación de la posible existencia de un objeto en la celda y la clasificación del objeto.

Describiendo más formalmente la salida de una red de detección quedaría como se muestra a continuación en la figura 2.10.

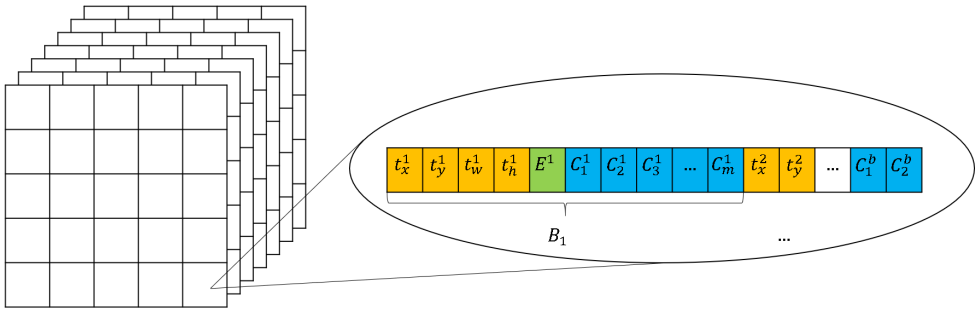


Figura 2.10: Salida de una red neuronal de detección

Donde:

- t_i^b donde $i \in \{c_x, c_y, w, h\}$ y $b \in \mathbb{R}^+$: Representa los offsets de las bounding boxes con las anchor boxes, el súper índice b es el número de bounding boxes en esa salida.
- E^b: Representa la existencia de un objeto en la bounding box.
- C_m^b donde $m \in \mathbb{R}^+$: Representa la probabilidad de que el objeto que encierra la bounding box pertenezca a la clase m.

La dimensión de cada vector como el que se aprecia en la figura 2.10 dependerá del número de clases que se deseen clasificar y del número de bounding boxes que se generen en la celda, por ejemplo si se desearan clasificar 100 clases y la celda genera 5 bounding boxes, las dimensiones del vector de salida de la celda serian $1x[(4+1+100)*5]$, con esto y con la ayuda de las ecuaciones 2.9, 2.10, 2.12 y 2.11 se obtiene la salida de la red ya decodificada y lista para dibujar la bounding box en la imagen de entrada original.

2.1.3.4 Umbralización de la salida

Después de la descripción de la salida que se hizo en la sección "Datos de salida de la red", se sigue teniendo el problema de saber cuál de las bounding boxes

generadas se adapta más al objeto que se desea detectar.

Para esto se utiliza el parámetro de salida E^b el cual como ya se mencionó indica la probabilidad de que exista un objeto en la bounding box b, con ello se pueden descartar todas las bounding boxes que tengan una E^b menor que un cierto umbral, esto delimita todas la bounding boxes que no estén enmarcando el objeto de interés o solo encierren una pequeña parte de este mismo.

Después de la umbralización del parámetro E^b aún siguen existiendo varias bounding boxes, por lo que para encontrar la óptima se genera otro parámetro conocido como intersección sobre unión o por sus siglas en ingles IoU, el cual nos indica que tan buena es la predicción de las dimensiones y posición de la bounding box que se le introdujo a la red como entada (ground true).

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} \tag{2.13}$$

En la ecuación 2.13 se puede apreciar esta operación, la cual consiste en calcular el área donde se traslapan la bounding box predicha y la dada para entrenamiento así como calcular el área de unión entre estas dos bounding boxes.

Una manera más gráfica de apreciar esta operación es la que se muestra en la figura 2.11, donde el área sombreada seria el área a calcular para realizar la operación.

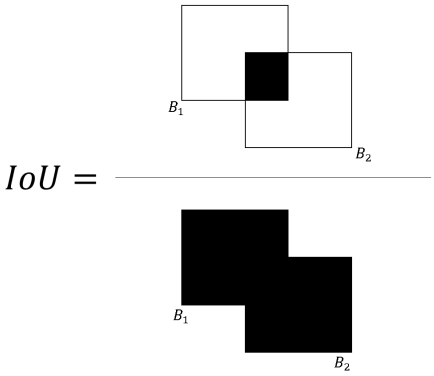


Figura 2.11: Operación IOU

2.1.3.5 Transferencia de aprendizaje en redes de detección

la transferencia de aprendizaje consta de la utilización de redes preentrenadas con bases de datos de millones de imágenes y una gran variedad de clases, este tipo de redes preentrenadas generan un aprendizaje en capas convolucionales muy generalizado, el cual puede ser utilizado para extraer características de clases que no estaban en la base de datos original, como por ejemplo bordes, esquinas, texturas, entre muchas otras. En Mehra *et al.* (2018) se hace una comparación entre diferentes redes utilizando transferencia de aprendizaje y entrenamiento "from scratch", este último es simplemente entrenar la red completa con una base de datos específica de las clases que se necesitan clasificar, se mostró que las redes entrenadas con transferencia de aprendizaje tuvieron una precisión del 92.6 % mientras que las redes "from scratch.ªpenas alcanzaron el 80 % de precisión, entre las redes utilizadas en el estudio anterior se encuentran, VGG-16, VGG-19 y ResNet50, redes muy utilizadas en tareas de detección Alippi *et al.* (2018); Jaworek-Korjakowska *et al.* (2019); Ray (2018). Este concepto es muy utilizado en arquitecturas de redes de detección como se mostrará en las próximas secciones.

2.1.3.6 Arquitecturas de redes de detección

Existen diferentes arquitecturas y procedimientos para la detección de objetos, sin embargo, para el presente trabajo se utilizaron dos en específico, la red You Only Look Once (YOLO) y Single Shot Detector (SSD), estas redes comparten las características y procedimientos anteriormente mencionados. A continuación, se describen sus arquitecturas y funciones de error.

- You Only Look Once version "tiny":

  Esta red está basada en la arquitectura de la red YOLOv3, YOLO en su versión tiny es una reducción de la versión v3 donde esta consta de más capas

de convolución y una salida redimensionada extra, más adelante se explicarán mejor estos términos.

Anteriormente las redes de detección constaban de una sola salida con un cierto número de celdas, sin embargo, este tipo de redes se le dificultaba la detección de los objetos cuando estos variaban su tamaño en las muestras que se les daban, una solución a esto es establecer diferentes salidas con diferente número de celdas, logrando detectar objetos de diferentes escalas.

Basándose en esto la red YOLOv3 consta de 3 salidas de 13x13, 19x19 y 52x52, caracterizándose por una buena precisión de detección y una gran velocidad de predicción.

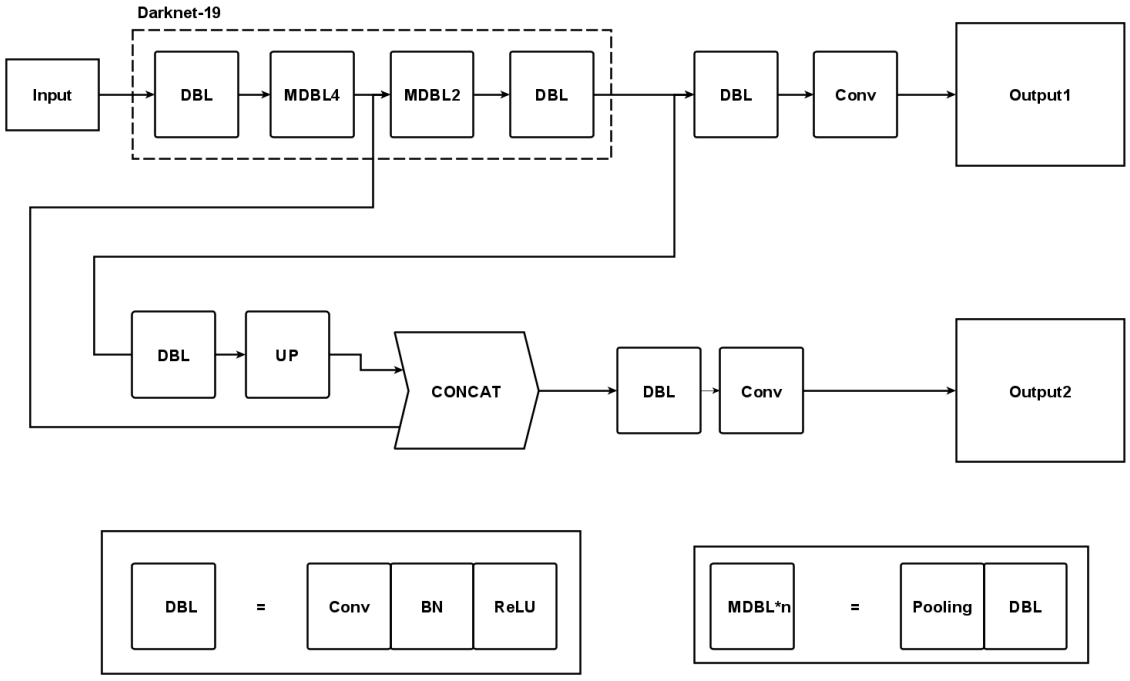


Figura 2.12: Red YOLO versión tiny

Sin embargo para lograr el objetivo de realizar el sistema de percepción descentralizado es necesario trabajar con una red más pequeña para agilizar el proceso de percepción y a su vez la velocidad de trabajo y respuesta del robot, por lo que se optó por trabajar con la red YOLO en su versión "tiny", la cual reduce su precisión en la detección de objetos pero incrementa conside-

rablemente su velocidad de respuesta, lo cual la hace adecuada para sistemas embebidos como el de la presente tesis.

La estructura de la red YOLO en su versión "tiny"se puede apreciar en la figura 2.12, esta red tiene dos salidas de tamaños 13x13 y 26x26, además de una reducción del número de convoluciones en comparación como la versión YOLOv3, esta red consta de dos clases de bloques llamados DBL y MDBL*n, el bloque DBL consta de una capa de convolución, una capa de normalización por lotes y una capa de activación. La capa $MDBL_n$ consta del mismo bloque DBL precedido por una capa de pooling, el subíndice n indica el número de veces que se repite esta capa, también se puede apreciar una etapa llamada "concat" la cual cumple la función de una conexión residual y une diferentes puntos de la red mediante la suma de las salidas de estos, por último se puede apreciar un bloque llamado up el cual representa una operación inversa al pooling llamada "upsampling" donde se incrementa la dimensión de la entrada a esa capa, un ejemplo de esta operación se puede apreciar en la ecuación 2.14.

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, b = upsamplig(a) = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix} \tag{2.14}$$

Por último, en cuanto a la descripción de la arquitectura de la red, se puede apreciar un recuadro punteado con la descripción "darknet-19", esta es una red pre-entrenada con la base de datos "ImageNet" de 1000 clases diferentes, lo cual proporciona un set de pesos capaces de extraer una gran variedad de características de las imágenes.

Como su nombre lo menciona esta red consta de 19 capas convolucionales incluyendo una parte densa propia de cualquier red de clasificación, sin embargo para su adaptación a la red YOLO tiny se desacopla la parte densa obteniendo una salida de kernels de características, también para su adaptación a una versión tiny se usa la configuración que se aprecia en la figura 2.12 y en la tabla

Capa	Filtros	Tamaño del filtro
CONV	16	3x3
POOL		2x2
CONV	32	3x3
POOL		2x2
CONV	128	3x3
POOL		2x2
CONV	256	3x3
POOL		2x2
CONV	512	3x3
POOL		2x2
CONV	1024	3x3

Tabla 2.1: Configuración Darknet-19 versión tiny

2.1.3.6.

Como se mencionó anteriormente esta red entrega información acerca de la bounding box que mejor encierra al objeto, como son su altura, anchura, centro de la bounding box, la posibilidad que en dicha caja se encuentre un objeto y la clase del objeto, con esta información se planteó la función de pérdida en Redmon y Farhadi (2018).

$$L = \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \gamma_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] +$$

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \gamma_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] +$$

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} \gamma_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 +$$

$$\lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \gamma_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 +$$

$$\sum_{i=0}^{s^2} \gamma_i^{obj} \sum_{c \in clases} (p_i(c) - \hat{p}_i(c))^2 \tag{2.15}$$

La ecuación 2.15 representa la función de pérdida antes mencionada, ésta se puede dividir en tres partes:

- Parte de clasificación:

$$\sum_{i=0}^{s^2} \gamma_i^{obj} \sum_{c \in clases} (p_i(c) - \hat{p}_i(c))^2 \tag{2.16}$$

  Donde:

  - s^2: es el número de celdas de la salida de la red.
  - γ_i^{obj}: Indica con un 1 si existe una imagen en determinada celda, en caso contrario toma el valor de cero.
  - $\hat{p}_i(c)$: indica el valor de la predicción de la clase c en la celda i

- Parte de localización:

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \gamma_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] +$$
$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \gamma_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \tag{2.17}$$

  Donde:

  - B: Número de bounding boxes predecidas.
  - γ_{ij}^{obj}: Indica si éxito un objeto en la bounding box j de la celda i
  - λ_{coord}: Incrementa el peso de esta parte de la función de pérdida.

- Parte de confianza:

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} \gamma_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 +$$
$$\lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \gamma_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \tag{2.18}$$

Donde:

- γ_{ij}^{noobj}: representa $\left(1 - \gamma_{ij}^{obj}\right)$
- λ_{noobj}: Incrementa el peso en esta parte de la función de pérdida.

Capa	Filtros	Tamaño del filtro
CONV 1-1	64	3x3
CONV 1-2	64	3x3
POOL		3x3
CONV 2-1	128	3x3
CONV 2-2	128	3x3
POOL		3x3
CONV 3-1	256	3x3
CONV 3-2	256	3x3
CONV 3-3	256	3x3
POOL		3x3
CONV 4-1	512	3x3
CONV 4-2	512	3x3
CONV 4-3	512	3x3
POOL		3x3
CONV 5-1	512	3x3
CONV 5-2	512	3x3
CONV 5-3	512	3x3
POOL		3x3
DENSE		
DENSE		
DENSE		

Tabla 2.2: Red VGG-16 completa

- Single Shot Detector:

  Al igual que la red anterior a continuación se presenta la estructura de la red SSD.

  Esta red contiene múltiples salidas de múltiples dimensiones, donde la primera de estas salidas viene de una red llamada VGG-16, la cual es una red convolucional preentrenada con la base de datos ImageNet, esta base de datos

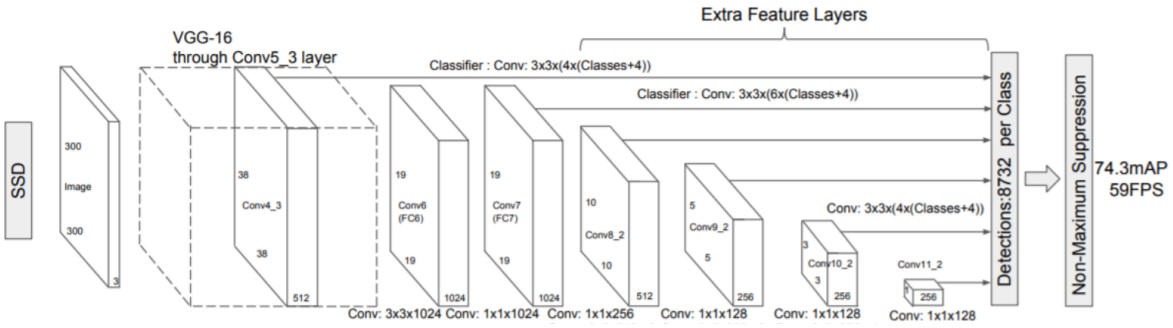


Figura 2.13: Red SSD

consta de más de 14 millones de imágenes, con más de 20,000 clases diferentes, al igual que en la red YOLO este preentrenamiento genera pesos capaces de identificar una gran variedad de características y atributos en imágenes que se le presenten, la configuración completa de este red se puede apreciar en la tabla 2.2.

Sin embargo como se puede apreciar en la figura 2.13 la red VGG-16 se desacopla a partir de la capa de convolución CONV 4-3 para poderse integrar a la SSD.

Esta red utiliza la siguiente función de pérdida:

$$L\left(x,c,l,g\right)=\frac{1}{N}\left(L_{conf}\left(x,c\right)+\alpha L_{loc}\left(x,l,g\right)\right) \tag{2.19}$$

La cual se compone de una función de pérdida de localización escalada por α:

$$L_{loc}=\left(x,l,g\right)=\sum_{i\in Pos}^{N}\sum_{m\in cx,cy,w,h}x_{i,j}^{k}smooth_{L1}\left(l_{i}^{m}-\hat{g}_{j}^{m}\right) \tag{2.20}$$

- $\hat{g}_j^m$: La Representa el offset de la bounding box dada para entrenamiento y el anchor box.

$$\hat{g}_{j}^{cx}=\left(g_{j}^{cx}-d_{i}^{cx}\right)/d_{i}^{w} \tag{2.21}$$

$$\hat{g}_j^{cx} = \left(g_j^{cy} - d_i^{cy}\right) / d_i^h \tag{2.22}$$

$$\hat{g}_j^w = log\left(\frac{g_j^w}{d_i^w}\right) \tag{2.23}$$

$$\hat{g}_j^h = log\left(\frac{g_j^h}{d_i^h}\right) \tag{2.24}$$

  - d_i^m: Representa la información de la anchor box.
  - g_i^m: Representa la información de la bounding box dada para entrenamiento.

- $x_{i,j}^k$: Indicador de coincidencia entre la anchor box i y la bounding box j de la categoría k.
- l_i^m: Predicción de la red.

Y como función de pérdida de clasificación se tiene:

$$L_{conf}\left(x, c\right) = -\sum_{i \in Pos}^{N} x_{i,j}^P log(\hat{c}_i^p) - \sum_{i \in Neg}^{N} log(\hat{c}_i^0) \tag{2.25}$$

- $x_{i,j}^p$: En este caso el indicador toma el valor de ground true con la siguiente propiedad.

$$x_{i,j}^P = \begin{cases} 1 & \text{Si el valor IoU} > 0.5 \text{ en la clase p} \\ 0 & \text{otherwise} \end{cases} \tag{2.26}$$

- C_i^0: Es el valor de predicción de la clase "Background".

2.2 Enjambres de robots

Existen diferentes animales que exhiben comportamientos colectivos, como por ejemplo las hormigas, abejas, lobos entre otros muchos animales. Todos estos conjuntos de animales colaboran para cumplir tareas específicas para su supervivencia y

preservación, siendo de cada uno de ellos habilidades sencillas que complementadas logran objetivos altamente complejos.

Los enjambres de robots tratan de imitar ciertos comportamientos de estos animales, tratándose de robots se características simples y minimalistas que juntando una cierta cantidad de integrantes pueden emerger comportamientos que resuelvan tareas de alta complejidad que un solo robot no podría por sí solo.

2.2.1 Características

Cuando se trata de imitar los comportamientos de ciertos animales sociales se sabe que cada uno de estos tiene percepción y toma de decisiones individuales, por lo que los enjambres de robots cumplen con la condición de ser sistemas descentralizados, tomando las decisiones individuales desde el procesador de cada robot y emergiendo comportamientos colectivos.

Un atributo importante de estos enjambres es la adaptabilidad al número de integrantes que los conforman, siendo tolerantes a la disminución o el aumento de integrantes, sin afectar el cumplimiento de tareas que se les asignen.

2.2.2 Comportamientos emergentes

Algunos ejemplos de estos comportamientos emergentes son los siguientes:

- Comportamientos de organización espacial:
  - Agregación: Consiste en conseguir que los robots se mantengan juntos mientras se desplazan. Este es uno de los principales fundamentos en el desarrollo de enjambres de robots.

- Formación: Siguiendo ciertas reglas de comportamiento los miembros del enjambre pueden realizar ciertas formaciones de patrones en sus desplazamientos.
- Auto ensamblaje: Determinadas configuraciones de robots pueden permitir que los miembros del enjambre puedan conectarse físicamente entre ellos con el fin de formar estructuras más grandes y complejas que permitan desarrollar tareas de igual características.
- Agrupación de objetos: Consiste en cierta interacción de los robots con su medio ambiente permitiéndoles mover objetos y agruparlos en algún punto de su medio ambiente.

- Comportamientos de navegación:
  - Exploración colectiva: Este comportamiento se caracteriza por una exploración en un ambiente desconocido para el enjambre en donde mediante algún tipo de comunicación entre los miembros del enjambre puedan guiarse y cumplir el objetivo asignado.
  - Transportación colectiva: Este comportamiento se hace presente cuando la tarea a cumplir es el transporte de un objeto el cual no es posible mover por un solo robot, por lo que es necesario la formación colectiva para su transporte.

- Comportamientos de decisión colectiva:
  - Consenso: Se da cuando el enjambre tiene varias alternativas para realizar la tarea asignada y este trata de encontrar la solución que maximice su desempeño.
  - Distribución de tareas: En este caso las actividades para cumplir con la tarea asignada se distribuyen en los miembros del enjambre para maximizar su desempeño.

2.2.3 Reglas de comportamiento

Existen constantes en los comportamientos de animales colectivos las cuales provocan que estos mantengan ciertas formaciones o puedan cumplir objetivos específicos de su especie, un modelo de estos comportamientos es propuesto por Couzin et al Couzin *et al.* (2002). que utiliza las reglas de congregación (flocking) propuestas por Reynolds Reynolds (1987). Estas reglas se basan en zonas que determinan ciertos comportamientos, estos son zonas de repulsión, atracción y orientación.

La zona de repulsión (ZOR) de radio r_r se encarga de indicar si existe un compañero u obstáculo muy cerca de algún integrante del enjambre haciendo que este trate de alejarse para mantener objetos fuera de esta zona. Después se tiene la zona de orientación (ZOO) de radio r_o, la cual tratará de orientar al integrante del enjambre hacia la misma dirección de los vecinos que se encuentren en ella. Por último se tiene la zona de atracción (ZOA) de radio r_a, la cual por el contrario de la zona de repulsión esta tratará de aproximar al sujeto hacia los vecinos que se encuentren en esta zona, estas zonas se pueden apreciar en la figura 2.14.

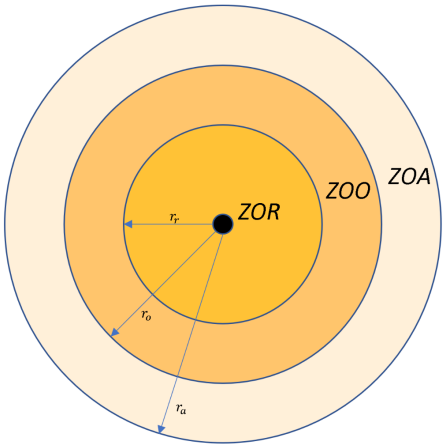


Figura 2.14: Zonas de repulsión, orientación y atracción Reynolds (1987)

En Ordaz-Rivas *et al.* (2019b) se agregó una nueva zona llamada de influencia, en la cual si se encuentra algún estímulo de influencia el enjambre tendera a cambiar o enfatizar su comportamiento hacia lo que el estímulo indique, esta zona se puede

apreciar en la figura 2.15.

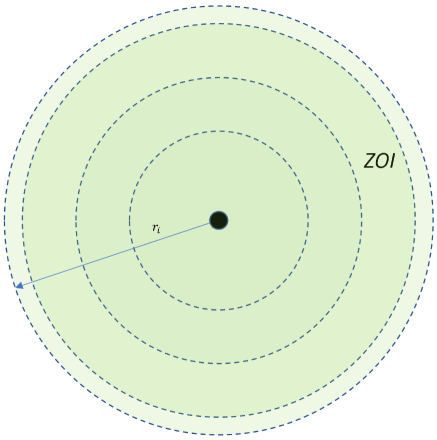


Figura 2.15: Zona de influencia Ordaz-Rivas *et al.* (2019b)

2.3 Robots móviles

Existen diferentes configuraciones de robots dependiendo del tipo de tarea que deban realizar, para los propósitos de la presente tesis se utilizarán robots móviles, los cuales permitirán reproducir las reglas de comportamiento antes mencionadas.

2.3.1 Robot diferencial

Dentro de la clase de robots móviles existen diferentes configuraciones, entre ellas se destacan la configuración Ackerman la cual es utilizada para la construcción de automóviles y se puede apreciar en la figura 2.16, caracterizada por constar de dos ruedas traseras y dos delanteras, teniendo tracción solo en un par de ellas y dirección en las ruedas delanteras, sin embargo, esta configuración presenta limitaciones en la movilidad ya que su modelo contiene restricciones no holonómas.

También se tiene la configuración diferencial la cual se puede apreciar en la figura 2.17, la cual consiste en dos ruedas que se encargan de la dirección y de la

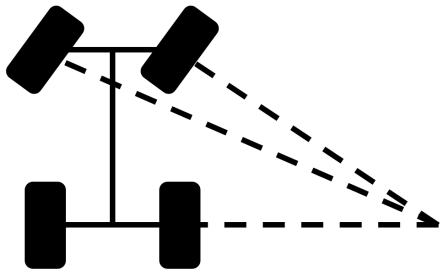


Figura 2.16: Configuración Ackerman

tracción del robot, esta configuración es fácil de implementar además de tener una movilidad adecuada para las posibles tareas de un enjambre de robots, ya que a diferencia de la configuración Ackerman pueden girar sobre su propio eje.

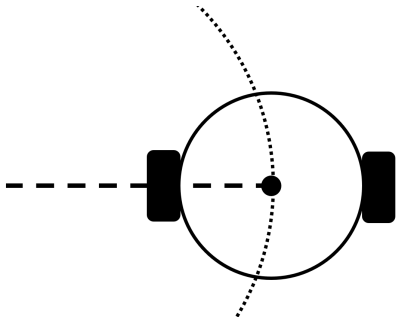


Figura 2.17: Configuración diferencial

Como ya se mencionó antes la configuración diferencial es la más adaptable para la mayoría de tareas ejercidas por un enjambre de robots, por lo cual fue la seleccionada para la experimentación del presente trabajo.

2.3.2 Modelo Matemático

A continuación, se presenta el modelo matemático basado en Ordaz Rivas (2020) de un robot diferencial:

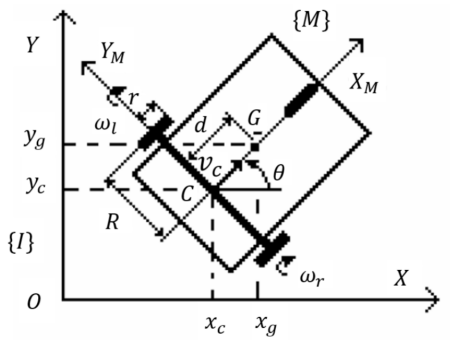


Figura 2.18: Consideraciones al modelar un robot diferencial.

$$\begin{bmatrix} \dot{x_c} \\ \dot{y_c} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}cos(\theta) & \frac{r}{2}cos(\theta) \\ \frac{r}{2}sin(\theta) & \frac{r}{2}sin(\theta) \\ \frac{r}{2R} & -\frac{r}{2R} \end{bmatrix} \tag{2.27}$$

2.3.2.1 Cinemática

En la figura 2.18 se aprecian algunas consideraciones para el desarrollo del modelo matemático del robot diferencial, las velocidades lineales y angular con respecto al centro C aparecen en la ecuación 2.27.

Donde:

- r: Es el radio de las ruedas del robot.
- θ: Es el ángulo del robot con respecto al eje x.
- $2R$: Es la distancia entre las ruedas.

$$x_g = x_c + dcos(\theta) \tag{2.28}$$

$$y_g = y_c + dsen(\theta) \tag{2.29}$$

$$\begin{bmatrix} \dot{x_g} \\ \dot{y_g} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}cos(\theta) - \frac{rd}{2R}sin(\theta) & \frac{r}{2}cos(\theta) + \frac{rd}{2R}sin(\theta) \\ \frac{r}{2}sin(\theta) + \frac{rd}{2R}cos(\theta) & \frac{r}{2}sin(\theta) - \frac{rd}{2R}cos(\theta) \\ \frac{r}{2R} & -\frac{r}{2R} \end{bmatrix} \tag{2.30}$$

Teniendo las coordenadas del centro de gravedad 2.28 y 2.29, se puede diferenciar para obtener las velocidades en el marco inercial, finalmente se muestra el modelo cinemático completo en la ecuación 2.30.

2.3.2.2 Dinámica

Tomando las ecuaciones 2.28 y 2.29 se pueden reescribir en forma polar.

$$\vec{P_g} = \vec{P_c} + de^{j\theta} \tag{2.31}$$

Donde:

- P_g: Posición del centro de masas del robot
- P_c: Posición del origen del marco móvil.
- d: Distancia del origen del marco móvil al centro de masas

Obteniendo su segunda derivada:

$$a_g = \left(\dot{v_c} - d\dot{\theta}\right) e^{j\theta} + j\left(v_c\dot{\theta} + d\ddot{\theta}\right) e^{j\theta} \tag{2.32}$$

La parte real de la ecuación 2.32 representa el movimiento lineal del robot y la parte imaginaria representa el movimiento rotatorio, por lo que a partir de estas ecuaciones se pueden describir la fuerza F_g 2.33 y el par 2.34.

$$F_g = m\dot{v}_c - md\dot{\theta}^2 \tag{2.33}$$

$$\tau_c = \left(I + md^2\right)\ddot{\theta} + mdv_c\dot{\theta} \tag{2.34}$$

Considerando que estas dos ecuaciones son equivalentes a $F_g = \frac{1}{r}(\tau_r + \tau_l)$ y $\tau_g = \frac{R}{r}(\tau_r + \tau_l)$ la dinámica del sistema se puede escribir como:

$$\begin{bmatrix} m & 0 \\ 0 & I_p + m \cdot d^2 \end{bmatrix} \begin{bmatrix} \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} -m \cdot d \cdot \dot{\theta}^2 \\ m \cdot d \cdot v \cdot \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{r} & \frac{1}{r} \\ \frac{R}{r} & -\frac{R}{r} \end{bmatrix} \begin{bmatrix} \tau_{dr} \\ \tau_{dl} \end{bmatrix} \tag{2.35}$$

Lo cual se puede escribir en una estructura general como la siguiente:

$$\mathbf{M} \cdot \dot{v} + H(v) = \mathbf{B} \cdot \tau \tag{2.36}$$

Considerando motores de CD cuyo modelo se obtiene con base a la teoría de perturbaciones singulares, suponiendo valores pequeños de inductancia de armadura, se tiene:

$$J_{m,i}\dot{w}_i + b_i w_i + \frac{k_{\tau,i}k_{fem,i}}{R_{a,i}}w_i + \frac{1}{C_i^2}\tau_i = \frac{k_{\tau,i}}{C_i R_{a,i}}u_i \tag{2.37}$$

Con $i \in \{r, l\}$, $J_{m,i}$ es la inercia del rotor y la rueda, $R_{a,i}$ es la resistencia de armadura, $k_{\tau,i}$ es la constante par-motor, $k_{fem,i}$ es la constante de fuerza contraelectromotriz, C_i es la relación de reducción de engranes y v_a es el voltaje de entrada a los motores. Dividiendo el modelo por $\frac{k_{\tau,i}k_{fem,i}}{R_{a,i}}$ se puede reescribir de la siguiente manera:

$$\tau_{s,i}\dot{w}_i + w_i + k_{L,i}\tau_i = k_{s,i}u_i \tag{2.38}$$

donde:

$$\tau_{s,i} = \frac{J_{m,i}}{b_i + \frac{k_{\tau,i}k_{fem,i}}{R_{a,i}}}, k_{L,i} = \frac{\frac{1}{C_i^2}}{b_i + \frac{k_{\tau,i}k_{fem,i}}{R_{a,i}}}, k_{s,i} = \frac{\frac{k_{\tau,i}}{C_i R_{a,i}}}{b_i + \frac{k_{\tau,i}k_{fem,i}}{R_{a,i}}} \tag{2.39}$$

Finalmente colocando en términos de $V = [v_g, \dot{\theta}]$ se puede integrar la dinámica del robot con el modelo de los motores CD como se muestra en la ecuación 2.40.

$$\left(M + BK_L^{-1}TA^{-1}\right)\dot{\mathbf{v}} + \left(H(\mathbf{v}) + BK_L^{-1}A^{-1}\mathbf{v}\right) = BK_L^{-1}K_{in}u \tag{2.40}$$

donde:

$$T = \begin{bmatrix} \tau_{s,r} & 0 \\ 0 & \tau_{s,l} \end{bmatrix}, K_L = \begin{bmatrix} k_{L,r} & 0 \\ 0 & k_{L,l} \end{bmatrix}, K_{in} = \begin{bmatrix} k_{s,r} & 0 \\ 0 & k_{s,l} \end{bmatrix}, A = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2R} & -\frac{r}{2R} \end{bmatrix}$$

Capítulo 3

Sistema de percepción y comportamientos implementados

En este capítulo se explican los componentes del sistema de percepción propuesto. El modelo de comportamiento está basado en el propuesto por Iain Counzin {couzin haciendo los ajustes necesarios para su implementación física y agregando un proceso de exploración a los comportamientos. Se describen los objetivos experimentales a realizar por el enjambre. Se describe la plataforma física de robots en la cual el sistema de percepción y el comportamiento de experimentación fueron implementados.

3.1 Plataforma de experimentación

Para la implementación de los algoritmos del comportamiento y del sistema de percepción se utilizaron dos prototipos de experimentación, ambos robots diferenciales. El primer prototipo constó de una tarjeta *Raspberry pi®* 3 B, 4 sensores infrarrojos, un magnetómetro, y la cámara correspondiente al *Raspberry pi®* 3 B, el prototipo se puede apreciar en la figura 3.1.

El segundo prototipo experimental se diseñó con base a la apariencia de un

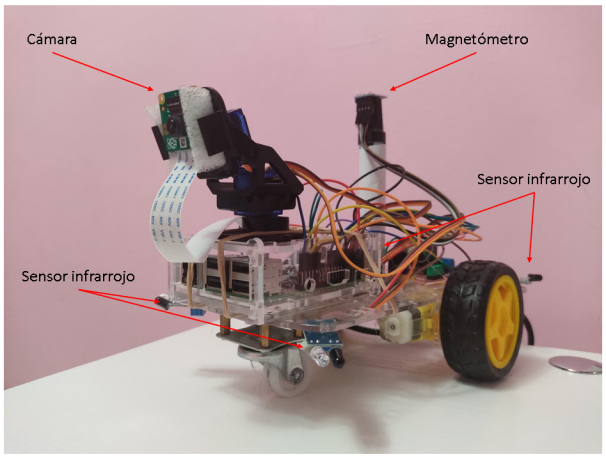


Figura 3.1: Primer prototipo experimental

lobo para darle una personalización al enjambre, nombrándolo Wolfbot. Los robots Wolfbots constaron de la tarjeta NVIDIA® Jetson Nano™ la cual se especializa en el desarrollo y ejecución de sistemas de inteligencia artificial, además de tener consumos de energía adecuados para tareas de enjambres de robots. La configuración de sensores de este robot es la misma que el prototipo 1 a diferencia de la exclusión del magnetómetro y la utilización de un sensor ultrasónico, en la figura 3.2 se puede apreciar la apariencia de un Wolfbot.

El propósito de los sensores utilizados en ambos robots a excepción del magnetómetro es el de detectar obstáculos, la detección de otros objetos se deja al sistema de percepción propuesto, el magnetómetro se utiliza como retroalimentación de orientación en el primer prototipo.

3.2 Comportamientos de experimentación

Para comprobar la utilidad del sistema de percepción desarrollado en esta tesis, se propone un comportamiento de exploración colectiva, donde cada robot será capaz de detectar obstáculos, robots vecinos y estímulos de influencia. Los objetivos principales de este comportamiento son:

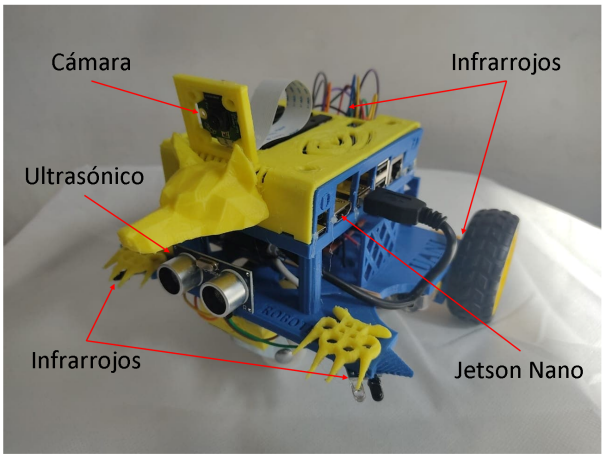


Figura 3.2: Segundo prototipo experimental

- Exploración colectiva por la interacción con robots vecinos.
- Búsqueda de estímulos de influencia.

Para la implementación de este comportamiento de exploración colectiva se utilizó una máquina de estados finitos compuesta por tres estados: búsqueda, atención y evasión. En el estado de búsqueda se encargará de ejecutar una rutina de exploración la cual permitirá explorar su ambiente por si solo, así como identificar si existe algún objeto de interés en la imagen de la cámara del robot, esta rutina de exploración consta de tres movimientos, un giro aleatorio de aproximadamente 60 grados seguido por otro giro de aproximadamente 120 grados en sentido contrario al primero, finalmente un avance hacia enfrente de aproximadamente 10 cm.

El estado de evasión como su nombre lo indica se encarga de esquivar obstáculos, utilizando los sensores infrarrojos se mueve en dirección contraria a la detección que haga alguno de ellos, la lógica de estos sensores se puede observar en la tabla 3.2, donde los sensores infrarrojos, frontal derecho, frontal izquierdo, trasero derecho y trasero izquierdo están representados por FD, FI, TD y TI respectivamente.

El sensor ultrasónico sirve para evitar obstáculos frontales, en dado caso que el sensor ultrasónico registre una distancia menor de 15 centímetros el WolfBot

FD	FI	TD	TI	Adelante	Atras	Izquierda	Derecha	Alto
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0	0
0	0	1	1	1	0	0	0	0
0	1	0	0	0	0	0	1	0
0	1	0	1	0	0	0	0	1
0	1	1	0	0	0	0	0	1
0	1	1	1	0	0	0	0	1
1	0	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	1
1	0	1	0	0	0	0	0	1
1	0	1	1	0	0	0	0	1
1	1	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	1
1	1	1	0	0	0	0	0	1
1	1	1	1	0	0	0	0	1

Tabla 3.1: Lógica de sensores infrarrojos

recorrerá hacia atrás una distancia de 10 centímetros y realizará un giro de 60 grados en algún sentido aleatorio.

Finalmente se tiene un estado de atención, en donde el robot localiza un estímulo de influencia o un objeto de interés, se orienta hacia él y empieza a aproximarse hasta una cierta distancia predeterminada, la máquina de estados completa se puede apreciar en la figura 3.3.

Una de las grandes ventajas del sistema de percepción es la gran variedad de estímulos de influencia que ahora se pueden integrar al comportamiento de cada robot, además de estímulos también se pueden detectar objetos que puedan ayudar a la toma de decisiones del enjambre, por ejemplo, el reconocimiento de otros robots que es el caso de esta tesis.

Cuando se detecta un estímulo de influencia el robot tendera a aproximarse hacia él y quedarse sin movimiento, mientras que si detecta a un compañero este igualmente tenderá a acercase a él, sin embargo, llegando a cierta distancia de su

vecino este robot regresará al estado de búsqueda para seguir explorando y lograr encontrar un estímulo de influencia.

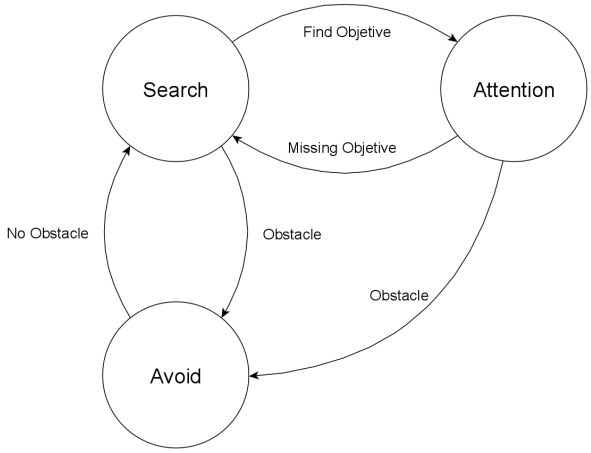


Figura 3.3: Comportamiento de experimentación

3.3 Sistema de percepción

En el capítulo 2 se mencionaron conceptos básicos de redes neuronales de clasificación y detección, las cuales conforman el sistema de percepción de cada miembro del enjambre de robots.

El primer prototipo se implementó en una tarjeta *Raspberry pi®* 3 B, teniendo una capacidad de procesamiento limitada, por lo que utilizar una red de detección en todo el tiempo activo de cada robot haría que tuvieran un desempeño deficiente y no alcanzar a cumplir con los objetivos asignados, por lo que se optó por agregar una red de clasificación la cual reduce el tiempo de actividad de la red de detección.

La red de clasificación se usará en el estado de búsqueda ya que en este no se necesita tener la ubicación del objeto y está activo la mayor parte del tiempo de trabajo del enjambre, en cuanto esta red reconoce algún objeto de interés se realiza un cambio de estado, pasando del estado de búsqueda al de atención activándose la red de detección.

La red de detección como ya se ha mencionado se encargará de ubicar al objeto de interés en la imagen para que cada miembro del enjambre pueda realizar una acción en base a la información que la red otorga. Estos datos sirven como retroalimentación a la máquina de estados y a los controladores del robot, el funcionamiento de este sistema de percepción se puede apreciar en la figura 3.4.

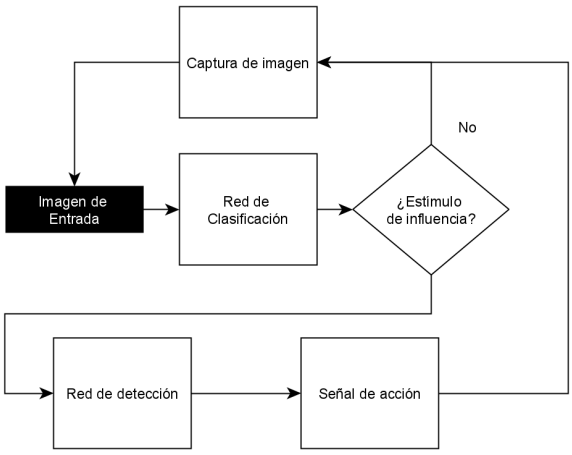


Figura 3.4: Sistema de percepción

3.3.1 Red de clasificación propuesta para prototipo 1

Aun disminuyendo el tiempo activo de la red de detección, utilizar un modelo al azar de clasificación para el estado de búsqueda sigue siendo un gran gasto computacional consumiendo un gran tiempo de ejecución considerando el primer prototipo, para solucionar esto se propuso una red residual de pocos bloques residuales. Este tipo de arquitectura nos permite trabajar con tamaños de imágenes pequeños, reduciendo el número de operaciones que se realizan durante las convoluciones y el número de bloques, disminuye el tiempo de ejecución de la red.

Esta red está compuesta por tres bloques residuales como el que se observa en la figura 3.5 donde el bloque “I” es la imagen de entrada, este bloque residual está compuesto por tres bloques residuales básicos que a su vez se conforman por una

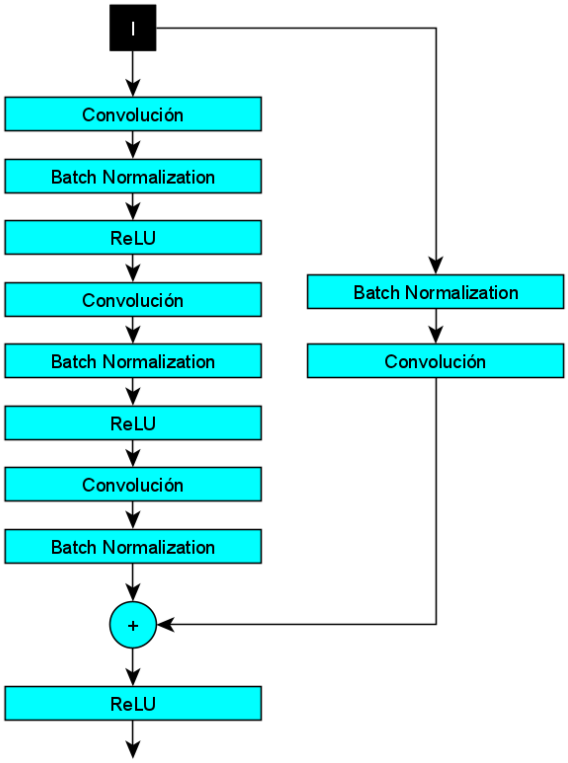


Figura 3.5: Bloque residual propuesto.

capa de convolución, una capa de normalización por lotes y una capa de activación, en el caso de este bloque residual la salida de la última capa de normalización no empata con las dimensiones de la imagen de entrada, por lo que se agrega una capa de convolución en la conexión residual que empata tanto la salida antes de la sumatoria y la entrada al bloque, en la tabla 3.3.1 se encuentran las características principales de esta red.

3.3.2 Red de clasificación para Wolfbot

Debido a la gran velocidad de procesamiento que brinda la tarjeta NVIDIA® Jetson Nano™ se puede utilizar una red de clasificación más potente a la anteriormente mencionada. Por lo que se optó por la utilización de la red ResNet-18 debido a ser una red de pocas capas y de fácil implementación, la estructura de esta red se

Características de la red de clasificación	
Batch size	32
Épocas	200
Taza de aprendizaje	0.001
Imagen de entrada	64x64
# de filtros iniciales	16
# de filtros finales	256

Tabla 3.2: características básicas de la red de clasificación

puede apreciar en la tabla 3.3.2.

Capa	Filtros	Tamaño del filtro
CONV1	64	7x7
POOL		3x3
$CONV2_x$	64	3x3
$CONV2_x$	64	3x3
$CONV2_x$	64	3x3
$CONV2_x$	64	3x3
$CONV3_x$	128	3x3
$CONV3_x$	128	3x3
$CONV3_x$	128	3x3
$CONV3_x$	128	3x3
$CONV4_x$	256	3x3
$CONV4_x$	256	3x3
$CONV4_x$	256	3x3
$CONV4_x$	256	3x3
$CONV5_x$	512	3x3
$CONV5_x$	512	3x3
$CONV5_x$	512	3x3
$CONV5_x$	512	3x3

Tabla 3.3: Configuración ResNet-18

3.3.3 Redes de detección para prototipo 1 y Wolfbot

Para ubicar los objetos en las imágenes captadas por cada robot se utilizaron las redes explicadas en el capítulo 2, para el prototipo 1 se utilizó la red YOLO tiny por el reducido número de operaciones al que se puede adaptar. Para los WolfBots se utilizó la red SSD por su potencia y su facilidad de adaptación a la tarjeta Jetson Nano™.

3.4 Control

Una vez encontrada la ubicación de algún objeto de interés en la imagen percibida por el robot, este se tiene que orientar y empezar a aproximarse hacia él, para esto se utilizaron controladores PD en ambos prototipos, a continuación se precisan detalles de cada prototipo.

3.4.1 Control utilizado en el prototipo 1

Una de las propuestas que se han hecho para que el sistema de percepción funcione adecuadamente en la tarjeta *Raspberry pi®* 3 B, es que se reduzca el tiempo activo de la red de detección, por lo que una vez entrando al estado de atención la primera ejecución de la red de detección dará las coordenadas del objeto de interés en la imagen, además de poder extraer el largo y ancho de la bounding box del objeto, esto nos da una referencia del tamaño de este.

A partir de estos datos entregados por la red se toma de referencia el centro de la bounding box para la retroalimentación del controlador de orientación, sin embargo, esto haría deficiente el poder orientarse hacia el objeto ya que deberían de tomarse numerosas muestras de la posición del objeto, para compensar esto se utilizó un magnetómetro que mediante una relación lineal entre la ubicación del centro de

la bounding box y una aproximación del campo de visión del robot se puede orientar hacia el objeto con una sola muestra de la red de detección.

Con base en el ángulo calculado se utilizó un controlador PD para orientar correctamente al robot hacia el objeto deseado. Ahora se tiene un caso diferente en cuanto al controlador de aproximación, para poder generalizar la percepción a estímulos de influencia y objetos en general tridimensionales como es el caso de robots vecinos, no se recomienda la utilización de un sensor ultrasónico como retroalimentación al controlador de aproximación, a menos que se cuente con un diseño robusto de los miembros del enjambre, ya que muy comúnmente las llantas de los robots tienen irregularidades o no están completamente alineadas, lo que hace que los robots se les dificulte ir en línea recta y a su vez en el caso del presente trabajo pueda conllevar a lecturas falsas por parte de un sensor ultrasónico de retroalimentación al controlador de la aproximación.

Dicho lo anterior para la retroalimentación del controlador de aproximación, se debe utilizar la misma red de detección, usando como magnitud el largo o ancho de la bounding box, ya que esto aproxima el tamaño del objeto y por consiguiente es una estimación de que tan cerca o lejos está el objeto del robot. De igual manera para reducir el tiempo activo de la red de detección, se propone usar un muestreo de poca frecuencia, ajustando el parámetro de repulsión a una distancia adecuada para que el robot no choque mientras se toma una nueva muestra.

3.4.2 Control utilizado en los Wolfbots

Gracias a la gran capacidad de la tarjeta Jetson Nano™ la velocidad de ejecución de la red de detección SSD es suficientemente corto como para utilizar la información que proporciona como retroalimentación en ambos controladores, tomando como retroalimentación para el controlador de orientación el centro de la bounding box y para el controlador de aproximación el largo de la bounding box,

algunos de los elementos mencionados se pueden apreciar en la figura 3.6.

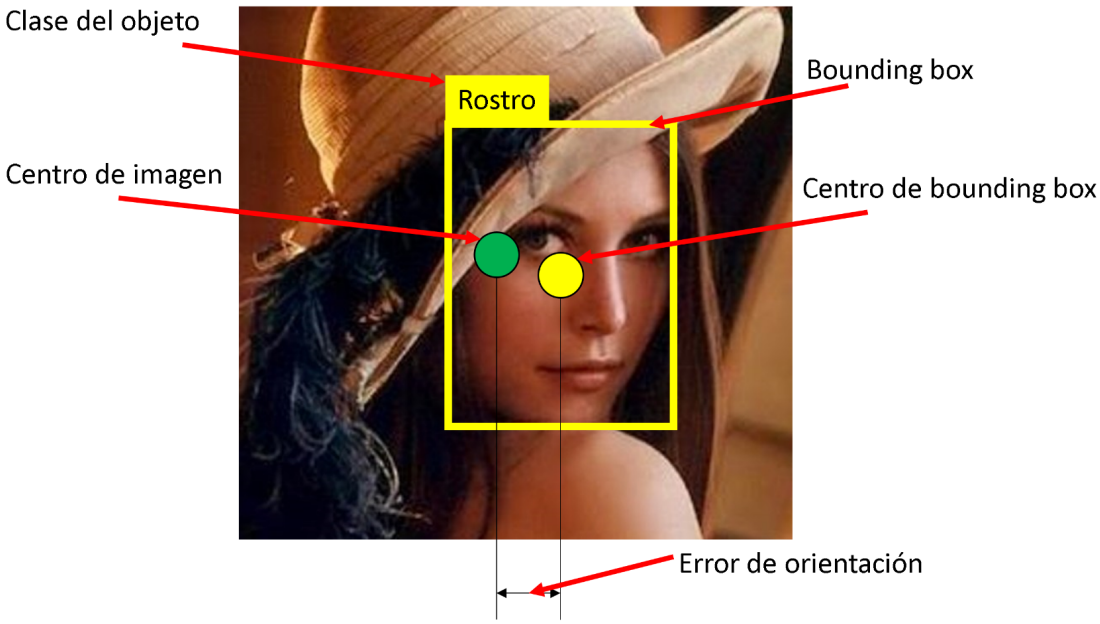


Figura 3.6: Parámetros utilizados para los controladores.

Capítulo 4

Experimentación

En este capítulo se presentan los detalles de la implementación del sistema de percepción y el comportamiento propuesto en los dos diferentes prototipos que se plantearon en capítulos anteriores, además se detallan algunas características de los ambientes de trabajo de los robots, así como el tipo de estímulos de influencia que se utilizaron.

4.1 Descripción de los experimentos

La experimentación con el prototipo 1 se realizó con un solo robot probando su capacidad de encontrar un solo estímulo de influencia, esto se realizó en un área de 180x50 cm delimitada por hojas de papel color blanco y paredes particulares de una habitación.

Para estas pruebas con el prototipo 1 se utilizó un solo estímulo de influencia como el que se muestra en la figura 4.1, con un tamaño de 20.34x19.4 cm.

En el caso de los WolfBots se utilizaron 3 robots para realizar pruebas en un espacio de 147x205cm, donde se utilizaron tres estímulos de influencia diferentes, los cuales se pueden apreciar en la figura 4.2 cada uno con un tamaño de 15.33x20.83cm, 20.81x26.19cm y 21.08x21.08cm respectivamente.

Figura 4.1: Estímulo de influencia para prototipo 1.

Figura 4.2: Estímulo de influencia para WolfBots.

Se utilizaron los mismos parámetros RAOI en los dos prototipos los cuales fueron:

- Repulsión: 10cm para robots vecinos y 20 cm para estímulos de influencia.
- Atracción e influencia: estas dos zonas constan de un radio de 100cm, donde es una aproximación de la distancia de percepción de cada robot.

4.2 Resultados de entrenamientos de las redes de clasificación y detección

A continuación, se presentan los errores obtenidos en los entrenamientos de las redes de clasificación y detección.

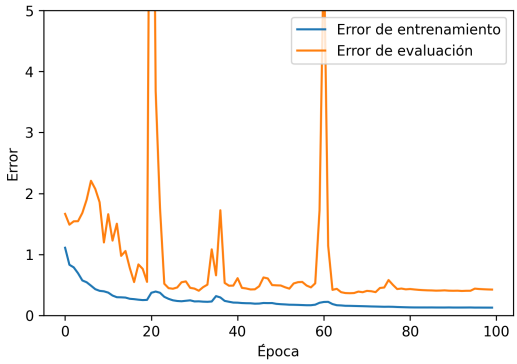


Figura 4.3: Error ResNet de tres bloques.

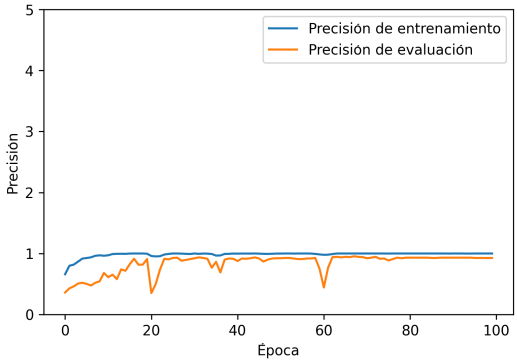


Figura 4.4: Precisión en ResNet de tres bloques.

En la figura 4.3 se puede apreciar la evolución del error a lo largo del entrenamiento de la red ResNet propuesta, se pueden observar algunos picos en la gráfica del error de evaluación, esto se debe a que en la sección de evaluación seleccionada de la base de datos existen elementos que no contienen información útil a la red, es por esto que la red al momento de ser evaluada falla en encontrar un objeto deseado en la imagen de evaluación. En la figura 4.4 se puede apreciar un correcto aumento de la precisión de la red a medida que el error disminuye.

En cuanto al entrenamiento de la red YOLO Tiny se puede ver en la figura 4.5 el comportamiento de la función de error implementada en la red, debido a la eficiencia que tiene, se puede apreciar la rápida convergencia del entrenamiento.

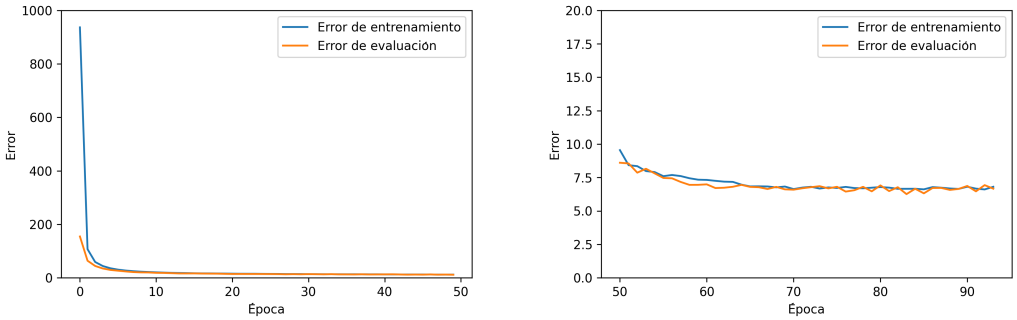


Figura 4.5: Precisión en ResNet de tres bloques.

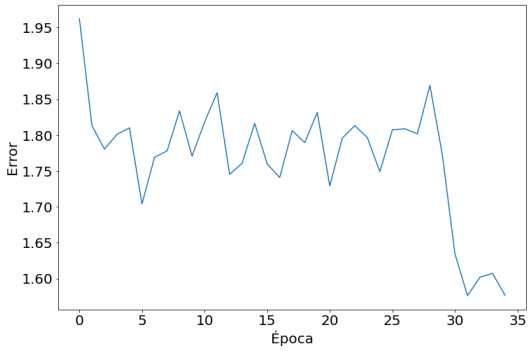


Figura 4.6: Error ResNet-18.

Para la experimentación con los WolfBots como ya se mencionó anteriormente se utilizaron las redes ResNet-18 y SSD, en la figura 4.6 se puede apreciar la evolución de error en la red ResNet-18 en donde por heurística se usó un entrenamiento a 35 épocas y como se puede a preciar en la figura 4.6 llegando a este número de épocas se ha podido reducir el error considerablemente. Esto mismo se puede observar en la figura 4.7 donde de igual manera por heurística se utilizaron 30 épocas e igualmente se observa como el error llega a un mínimo adecuado para la utilización de la red.

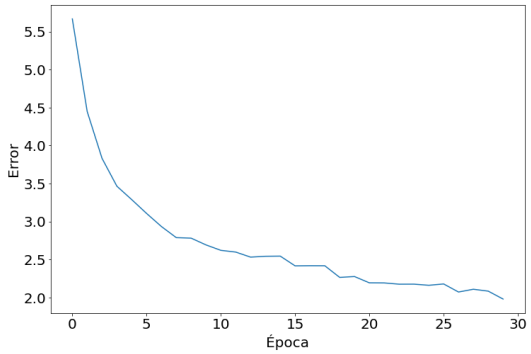


Figura 4.7: Error ssd.

4.3 Resultados de experimentación con prototipo 1

A continuación, se muestran secuencias de imágenes correspondientes a cada experimento con el prototipo 1.

Cada figura está compuesta por dos columnas de imágenes, las imágenes de la primera columna se puede observar una visión externa del movimiento del robot y en la columna derecha la percepción del robot correspondiente a los frames de la columna izquierda.

En la primera fila de las figuras 4.8, 4.9, 4.10 y 4.11 se puede observar el estado de búsqueda donde el robot se encuentra explorando su área de trabajo, después se observa como la red de clasificación encuentra el estímulo de influencia dándole paso al estado de atención que se puede observar en la tercera fila de cada figura, en la imagen de la percepción del robot se puede observar como la red de detección marca la bounding box correspondiente al estímulo de influencia, por último en la última fila se puede apreciar como el robot se ha aproximado hacia el estímulo de influencia.

Figura 4.8: Experimento 1 con prototipo 1.

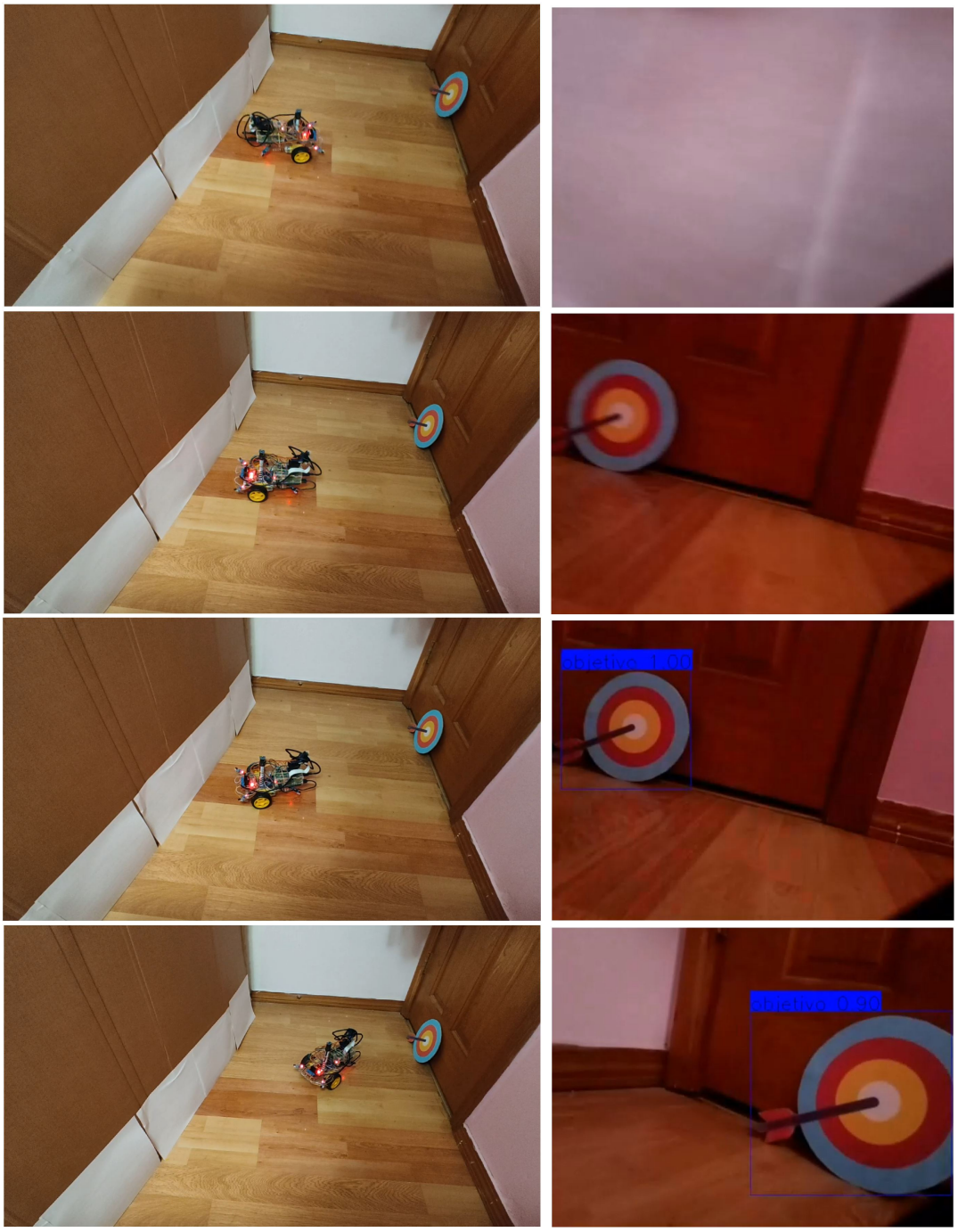


Figura 4.9: Experimento 2 con prototipo 1.

4.4 Resultados de experimentación con WolfBots

A continuación, se presentan secuencias de imágenes de los experimentos realizados con el enjambre de WolfBots, cada conjunto de imágenes se puede observar

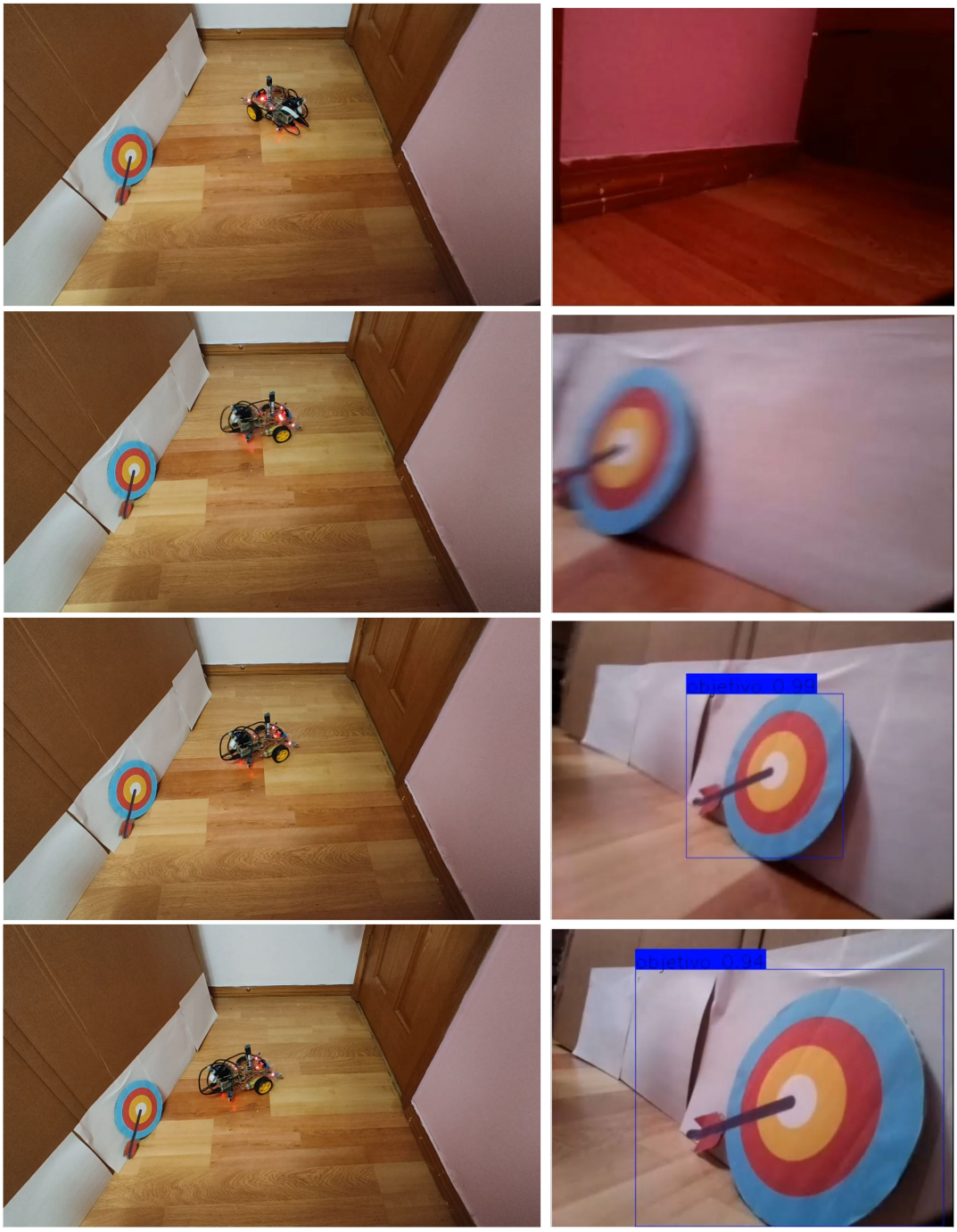


Figura 4.10: Experimento 3 con prototipo 1.

un experimento en particular, además de poder observar en cada figura muestras del estado de búsqueda y atención, en algunos frames se puede observar el estado de evasión de obstáculos donde en la imagen de la percepción de los robots no se puede apreciar la etiqueta de clasificación ni alguna bounding box de la red de detección.

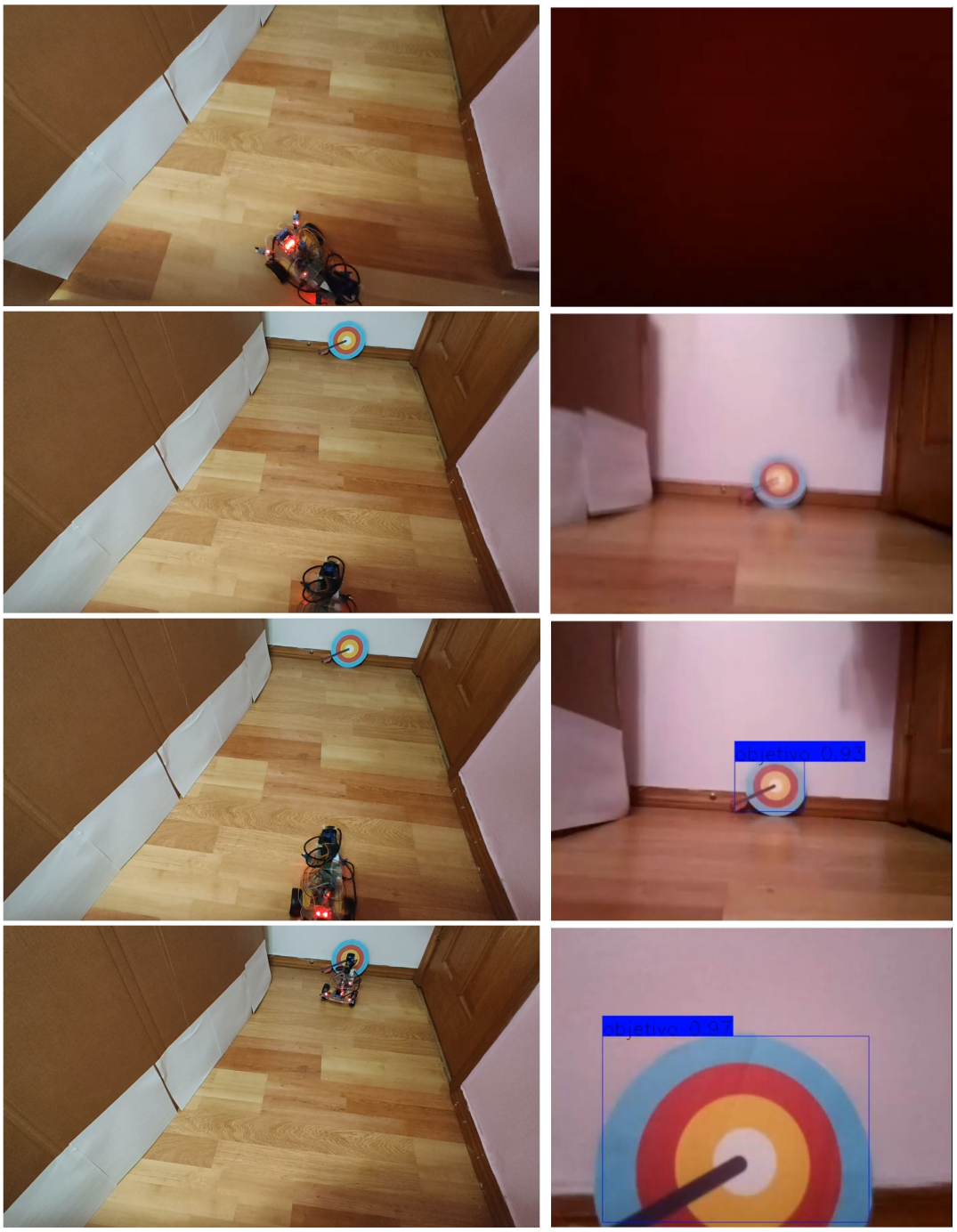


Figura 4.11: Experimento 3 con prototipo 1.

Dentro de cada frame de cada figura se puede apreciar una imagen con una vista externa de los robots y tres más pequeñas mostrando la imagen que reciben cada uno.

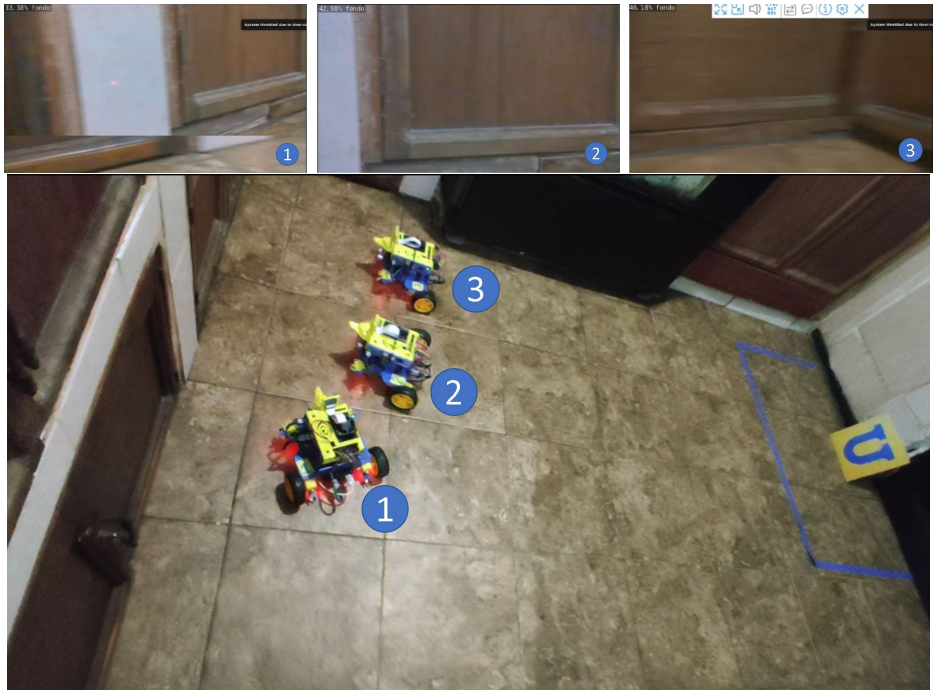


Figura 4.12: Experimento 1, frame 1 WolfBots

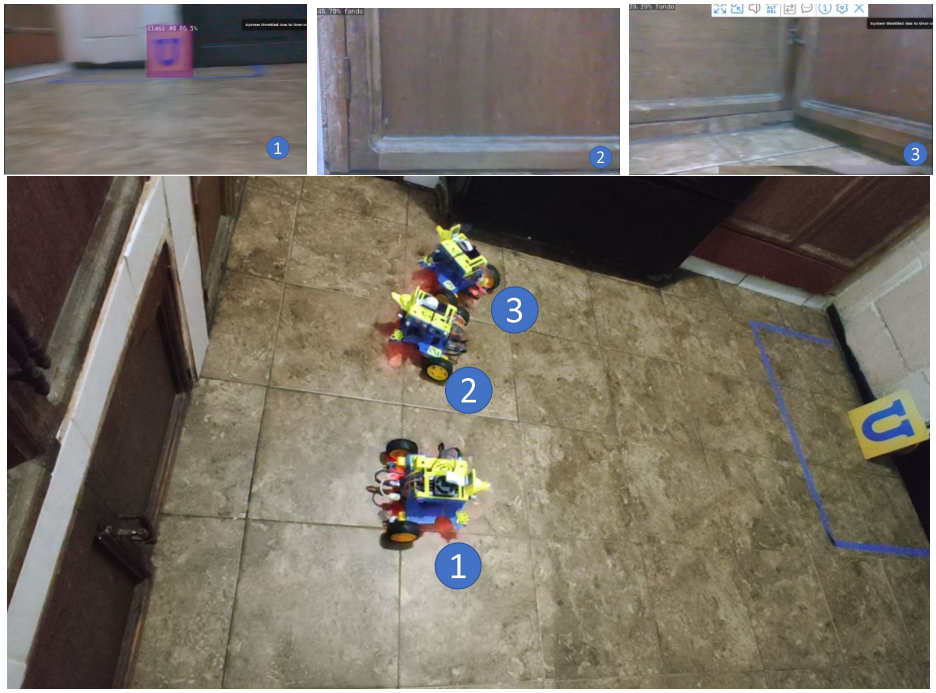


Figura 4.13: Experimento 1, frame 2 WolfBots

En la figura 4.30 se puede apreciar el primer experimento donde no existe un

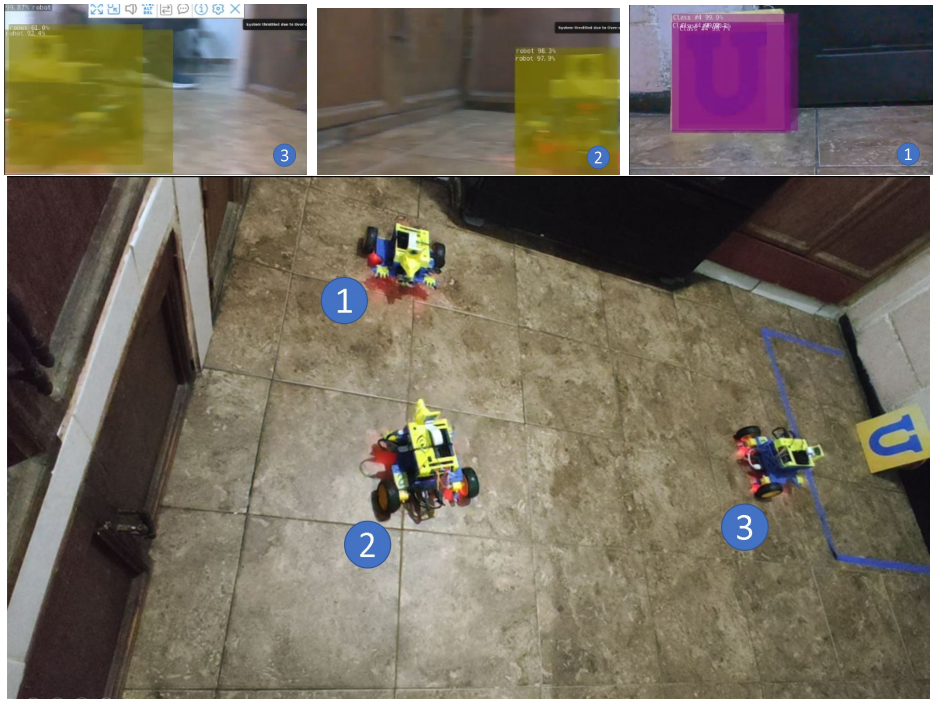


Figura 4.14: Experimento 1, frame 3 WolfBots

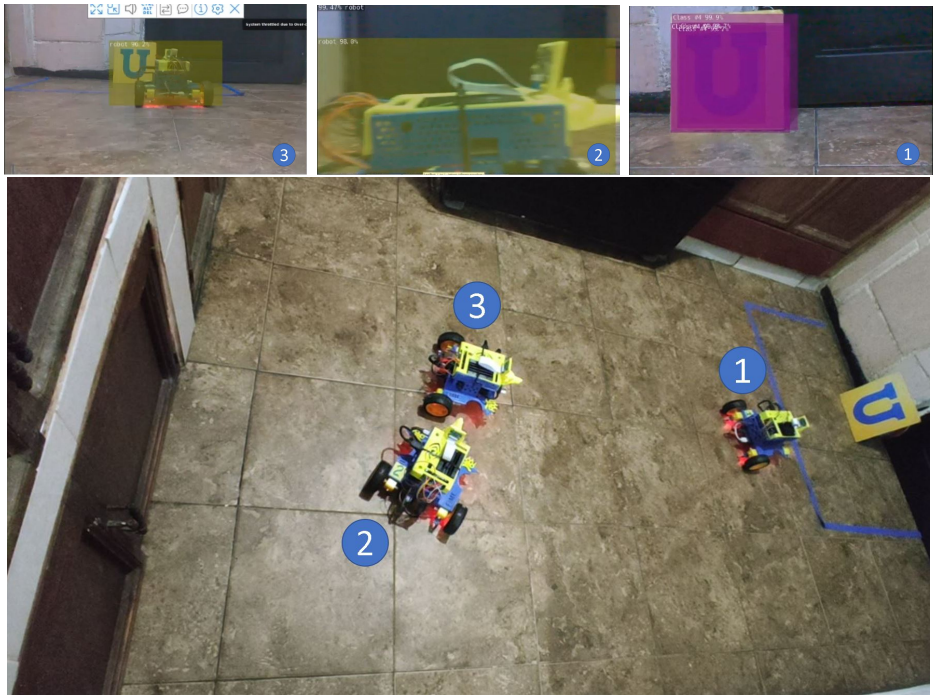


Figura 4.15: Experimento 1, frame 4 WolfBots

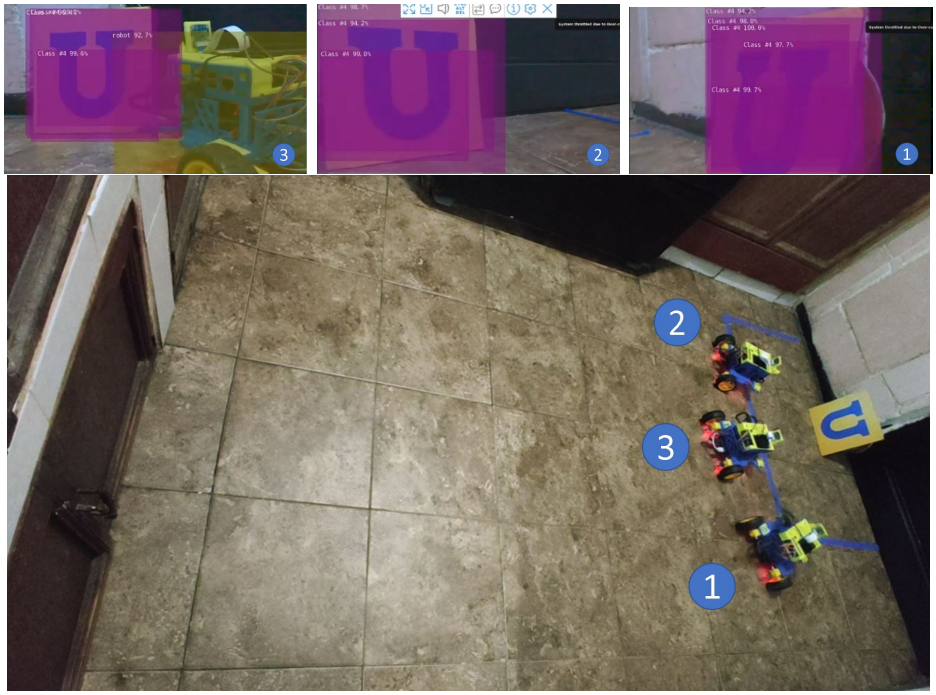


Figura 4.16: Experimento 1, frame 5 WolfBots

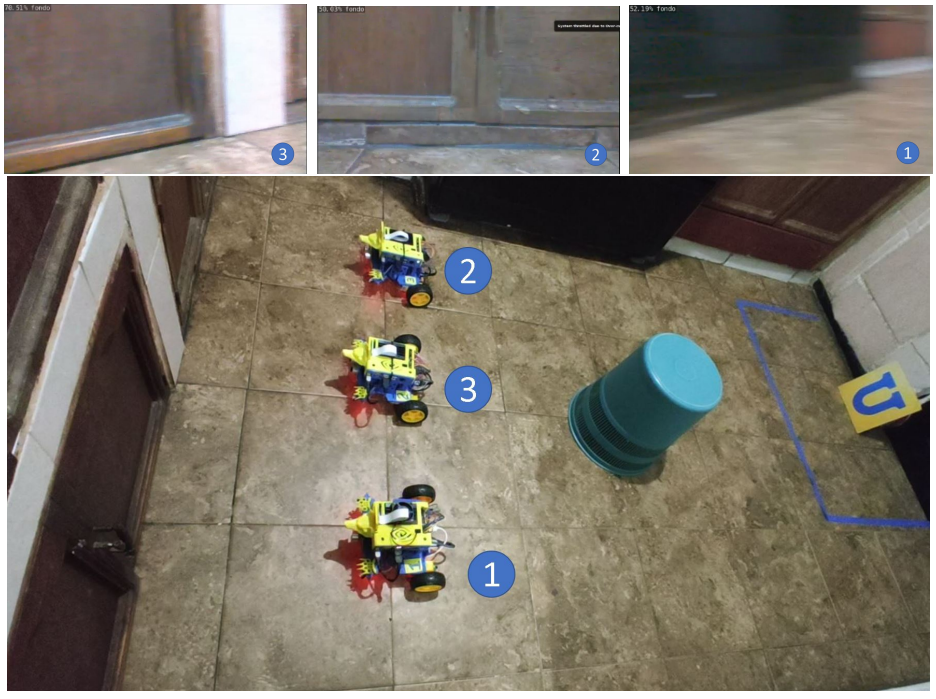


Figura 4.17: Experimento 2, frame 1 WolfBots

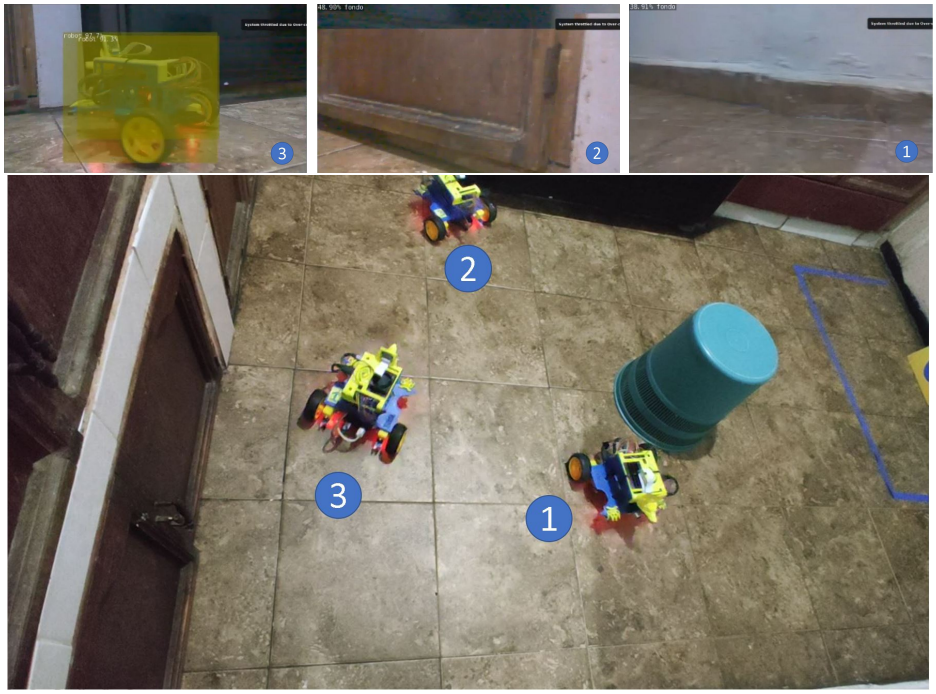


Figura 4.18: Experimento 2, frame 2 WolfBots

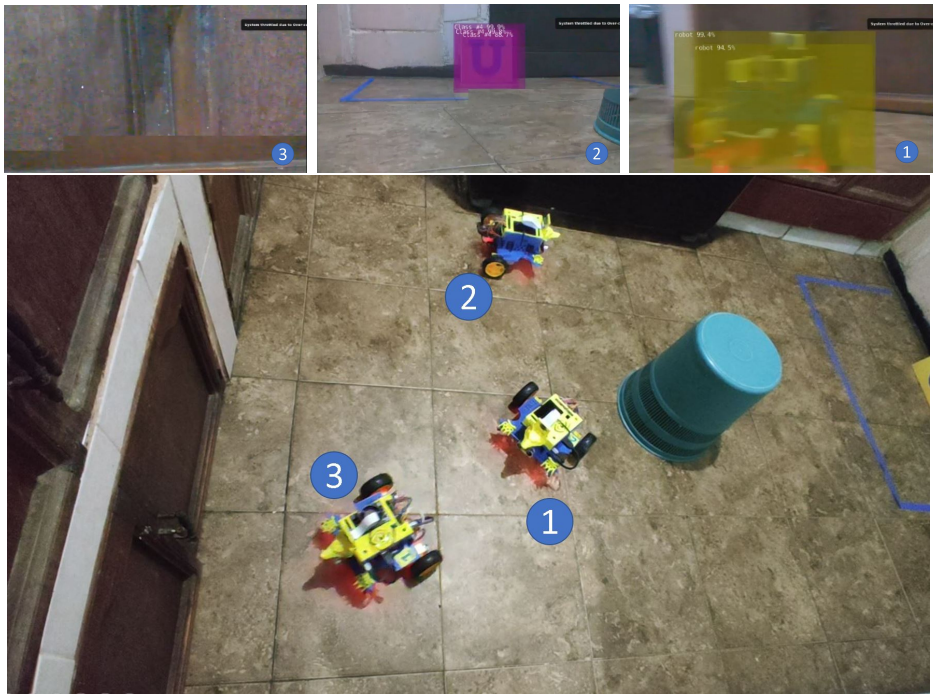


Figura 4.19: Experimento 2, frame 3 WolfBots

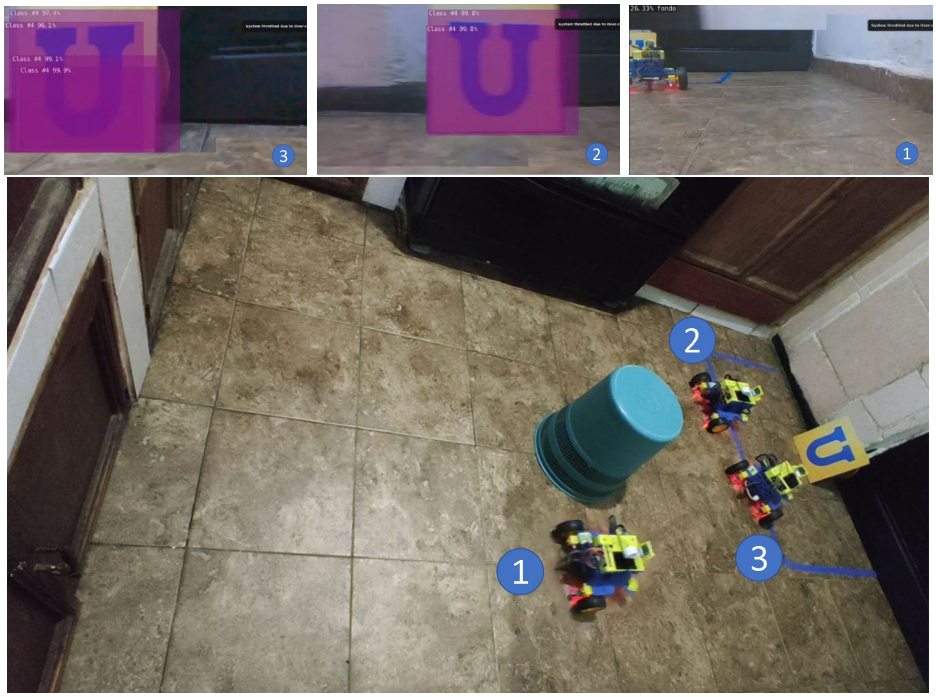


Figura 4.20: Experimento 2, frame 4 WolfBots

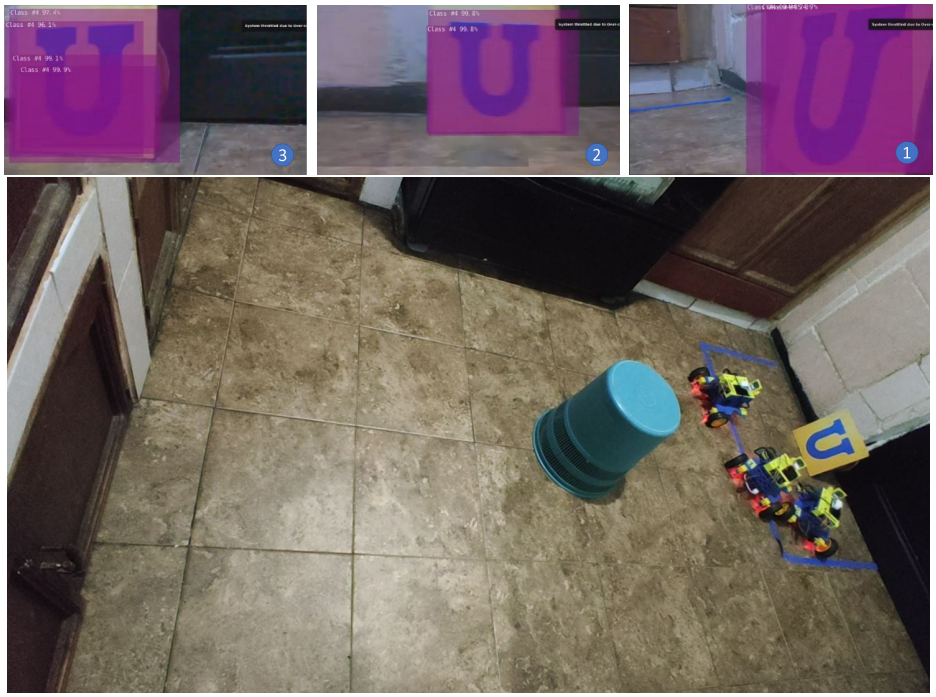


Figura 4.21: Experimento 2, frame 5 WolfBots

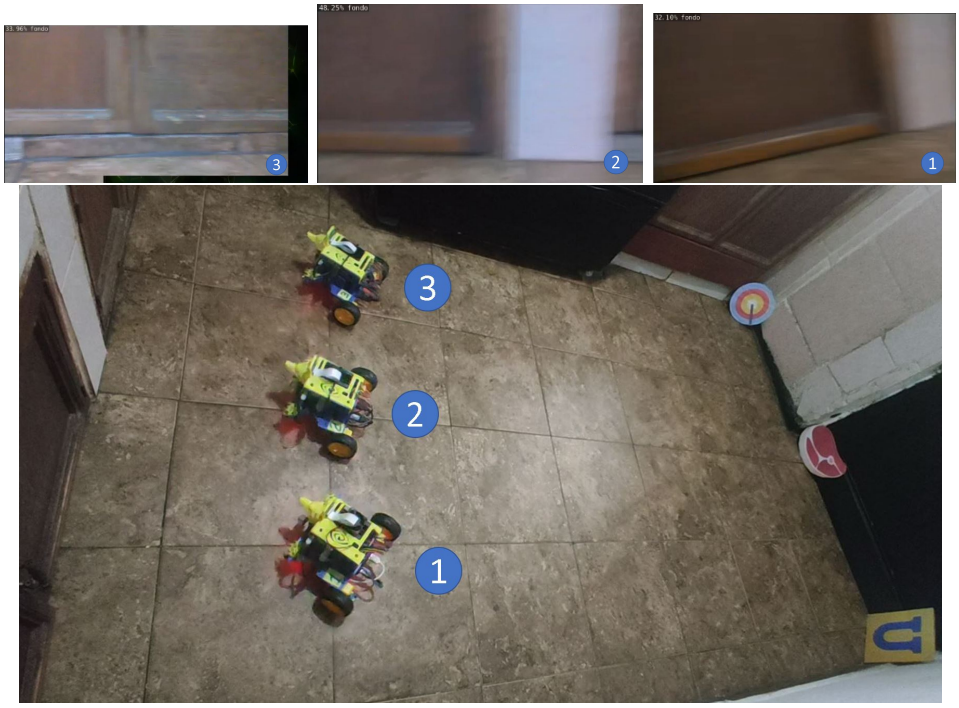


Figura 4.22: Experimento 3, frame 1 WolfBots

Figura 4.23: Experimento 3, frame 2 WolfBots

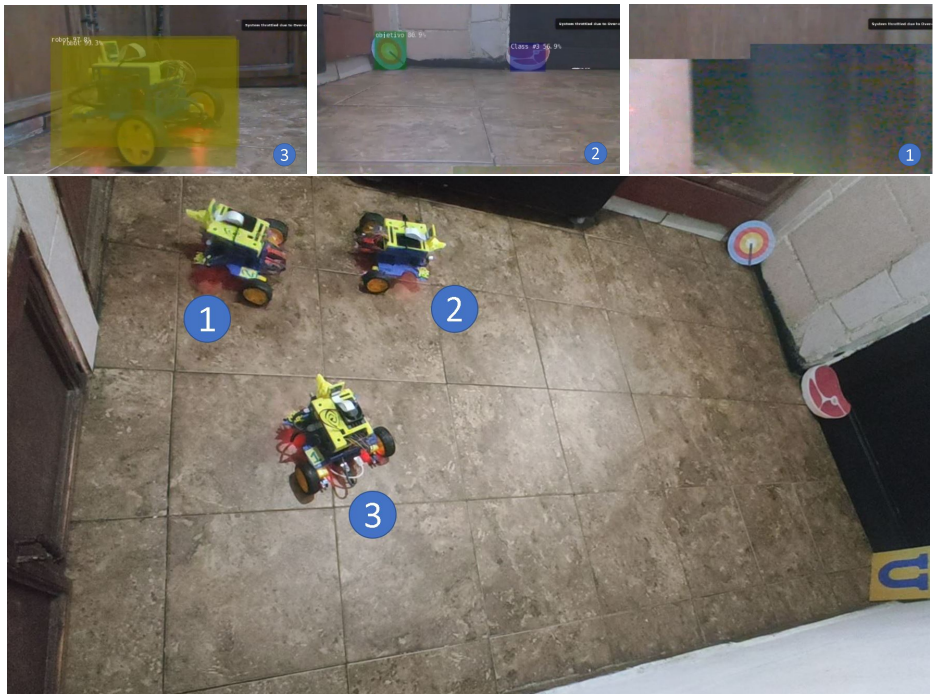


Figura 4.24: Experimento 3, frame 3 WolfBots

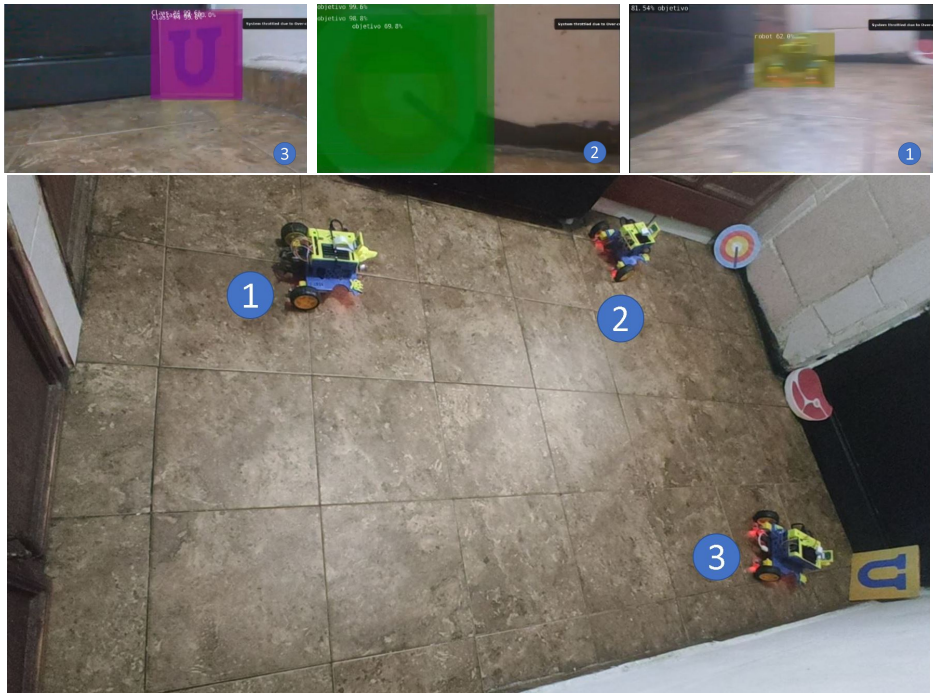


Figura 4.25: Experimento 3, frame 4 WolfBots

Figura 4.26: Experimento 3, frame 5 WolfBots

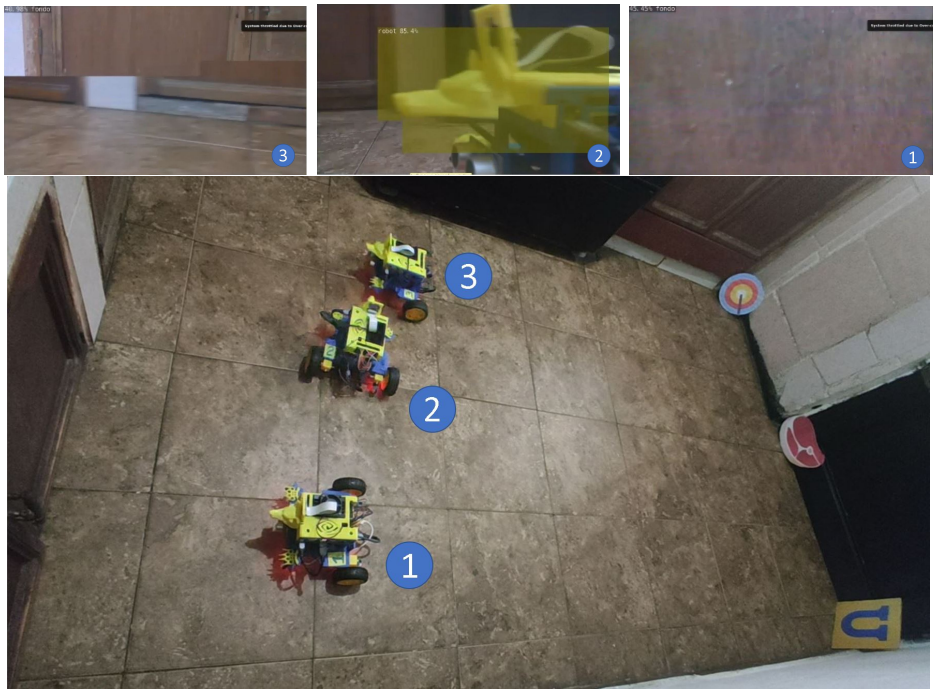


Figura 4.27: Experimento 4, frame 1 WolfBots

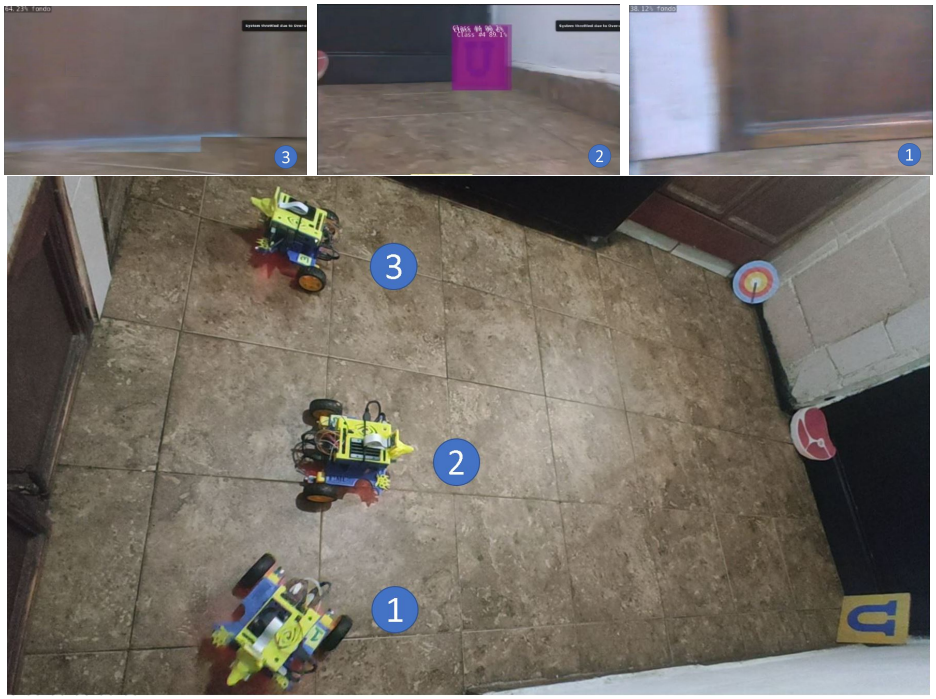


Figura 4.28: Experimento 4, frame 2 WolfBots

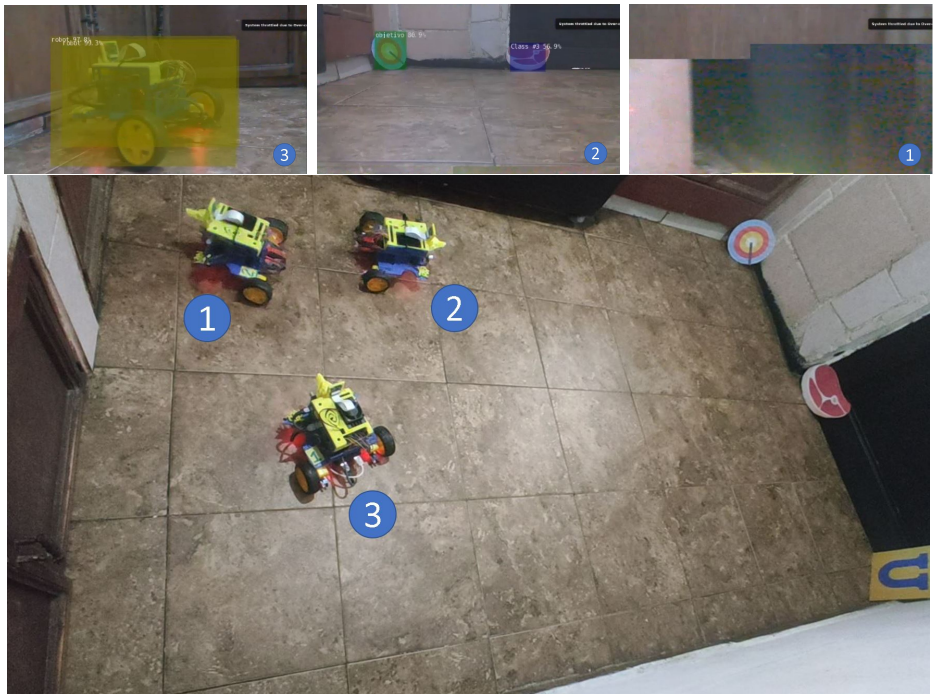


Figura 4.29: Experimento 3, frame 3 WolfBots

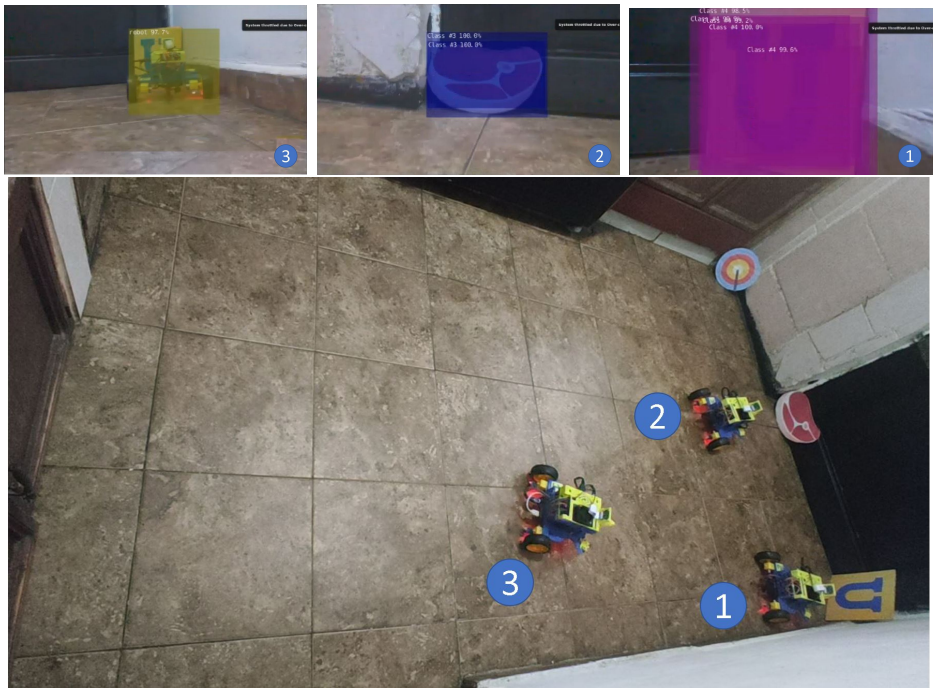


Figura 4.30: Experimento 4, frame 4 WolfBots

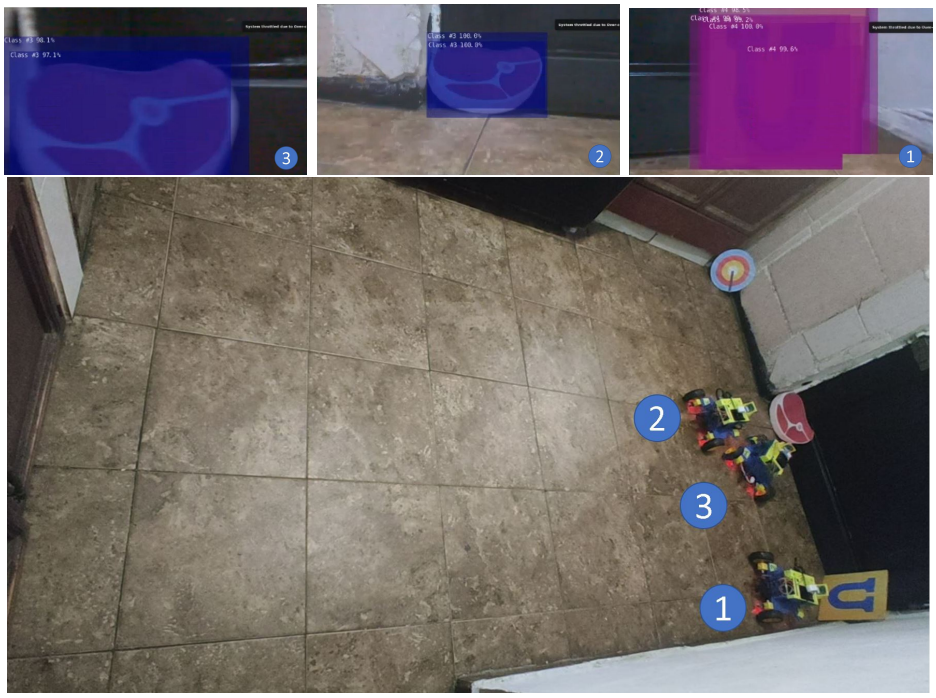


Figura 4.31: Experimento 4, frame 5 WolfBots

obstáculo más que la delimitación de la zona de trabajo y solo se encuentra un

Experimento	Robot	Tiempo
1	1	00:11
	3	01:10
	2	02:15
2	3	00:37
	1	01:18
	2	01:18
3	1	00:18
	2	00:19
	3	01:48
4	2	00:28
	3	01:43
	1	01:56

Tabla 4.1: Tiempos de llegada a estímulos de influencia.

estímulo de influencia, a continuación se da una breve descripción de cada frame del experimento:

- Frame 1: Se puede apreciar el inicio de experimento, con una aproximación de las posiciones iniciales de los robots.
- Frame 2: Un robot ha encontrado un estímulo de influencia y dos siguen en exploración.
- Frame 3: El robot ha podido aproximarse al estímulo de influencia mientras en los dos robots restantes se puede apreciar que realizan interacciones entre ellos, permitiéndoles cambiar la rutina de exploración y abarcar diferentes áreas.
- Frame 4: Se puede apreciar como los robots se perciben entre ellos, donde uno observa el robot que ya ha encontrado el estímulo de influencia y a su vez este es observado por otro vecino.
- Frame 5: Finalmente Los tres robots encuentran el estímulo de influencia.

En la figura 4.21 se puede apreciar el segundo experimento donde existe un obstáculo y solo se encuentra un estímulo de influencia, a continuación se da una breve descripción de cada frame del experimento:

- Frame 1: Se puede apreciar el inicio de experimento, con una aproximación de las posiciones iniciales de los robots.
- Frame 2: Los robots siguen en exploración y se puede apreciar el funcionamiento de la red de detección entre ellos.
- Frame 3: Un robot a encontrado el estímulo de influencia, mientras que en los otros dos se puede observar como uno trata de seguir al otro vecino en exploración realizando una especie de fila.
- Frame 4 y Frame 5: Finalmente los dos últimos robots localizan y se aproximan hacia el estímulo de influencia.

En la figura 4.26 se muestra el tercer experimento donde no existe un obstáculo y se encuentran múltiples estímulos de influencia, a continuación se da una breve descripción de cada frame del experimento:

- Frame 1: Se puede apreciar el inicio del experimento, con una aproximación de las posiciones iniciales de los robots.
- Frame 2 y 3: Se observa el primer robot localizando un estímulo de influencia, mientras que los restantes siguen en estado de búsqueda interactuando entre ellos.
- Frame 4 y 5: Los robots restantes encuentran distintos estímulos de influencia cada uno.

En la figura 4.31 se muestra el cuarto experimento donde no existe un obstáculo y se encuentran múltiples estímulos de influencia, este es una réplica del experimento

3 sin embargo se puede apreciar que dos robots han encontrado el mismo estímulo de influencia.

En la tabla 4.1 se puede apreciar los tiempos de llegada de cada robot.

Capítulo 5

Conclusiones y Trabajo a Futuro

En el presente trabajo se ha adaptado un modelo de percepción utilizando aprendizaje profundo a un enjambre de robots basado en el modelo propuesto por Iain Couzin, esto con la idea de aumentar el nivel de percepción que presentan los enjambres actuales que han adaptado este modelo, donde estos enjambres solo tienen una percepción local de su entorno proporcionada por sensores que registran magnitudes básicas de su entorno como iluminaciones, distancias, entre otras más.

Con el fin de cumplir los objetivos de detección se dividió el modelo de aprendizaje profundo en dos redes, una de clasificación y una de detección. En cuanto a la red de clasificación se tuvieron algunos problemas para la generalización del concepto de fondo al aumentar el número de objetos que se requieren clasificar, debido a esta dificultad se utilizó un umbral en cuanto la probabilidad de clase detectada, con esto se puede decir que la generalización de conceptos dentro de una red de clasificación se vuelve más compleja a medida que aumenta el número de clases a detectar.

Una posible solución a la problemática anterior es la utilización de redes en paralelo encargadas de clasificar diferentes conceptos, en el caso anterior se podría implementar una red clasificadora de objetos y una red de clasificación de existencia

de un objeto de interés.

En los comportamientos se obtuvieron los resultados esperados al poder cumplir con la tarea de exploración y búsqueda, sin embargo, el aumentar el número y tipo de estímulos de influencia también aumenta la complejidad de la elaboración de las rutinas de exploración y del comportamiento en general de los enjambres.

Al momento de cambiar los estímulos de influencia, por ejemplo, de una fuente de luz a un objeto físico como es el caso del presente trabajo, se encontró con dificultades para finalizar con la tarea de búsqueda con el último robot de cada experimento, debido a que al ser un estímulo de influencia con un área limitada este era cubierto por los dos robots que lograban encontrar el estímulo, quitándole visibilidad al último robot y dificultando su tarea de búsqueda.

Se implementó dentro de la rutina de exploración la condición de continuar en estado de búsqueda en caso de estar a cierta distancia de un robot vecino, si se quita esta regla de la rutina de exploración se podrá observar una sincronización de los robots realizando filas, donde el robot que encabeza la fila sigue la rutina de exploración mientras los demás tratan de seguirlo, esto es de importancia en enjambres donde se necesite un robot líder con capacidades superiores a los demás integrantes.

Sin embargo, al momento de eliminar esta condición se presenta el problema de estancamiento entre los miembros del enjambre, ya que si no existe al menos un robot que siga en estado de búsqueda estos se mantendrán estacionados detectándose unos a otros impidiendo el cumplimiento de las tareas del enjambre.

Para trabajos futuros, la utilización de estos sistemas de percepción se puede convertir en vías de comunicación entre miembros del enjambre, utilizando colores o símbolos que puedan ser reconocidos por el sistema de percepción, obteniendo información acerca del estado en el que se encuentren sus vecinos y así lograr realizar una mejor toma de decisiones, esto mismo puede resolver las últimas dos problemáticas planteadas en este capítulo.

Apéndice A

Reporte técnico de los robots

A.1 Hardware

A.1.1 Tarjetas de procesamiento

A.1.1.1 Raspberry Pi 3 modelo B

Para implementar el sistema de percepción del prototipo 1 se utilizó la tarjeta Raspberry Pi 3 modelo B, la cual posee una CPU Quad-Core a 1.20GHz y 1GB de RAM, incluyendo adaptadores Wi-Fi y bluetooth.

Chipset	Broadcom BCM2837 a 1.2 GHz
Procesador	ARM Cortex-A53 de 64 bits
Bluetooth	4.1
LAN inalámbrica	802.11 b/g/n
Pines digitales I/O	20
Memoria	LPDDR2
Memoria RAM	1 GB

Tabla A.1: Características Raspberry Pi 3 modelo B

A.1.1.2 NVIDIA® Jetson Nano™

Para implementar el sistema de percepción del prototipo 1 se utilizó la tarjeta NVIDIA® Jetson Nano™, la cual es especializada en ejecutar algoritmos de aprendizaje profundo haciendo énfasis en visión artificial, teniendo las siguientes características.

Chipset	Broadcom BCM2837 a 1.2 GHz
GPU	128-core Maxwell™ GPU
CPU	Quad-core ARM A57 @1.43 GHz
Memoria	2 GB 64-bit LPDDR4 — 25.6 GB/s
Wi-Fi	802.11ac wireless
Pines digitales I/O	40
Memoria RAM	2 GB

Tabla A.2: Características NVIDIA® Jetson Nano™

A.1.2 Sensores y Actuadores

Cada uno de los WolfBots consta de un sensor ultrasónico HC-SR04, cuyas características se describen a continuación.

Ángulo efectivo	<15 grados
Rango de distancia	2 cm - 400 cm
Dimensiones	45mm x 20mm
Resolución	0.3 cm

Tabla A.3: Características HC-SR04

Cada WolfBot consta además de 4 sensores infrarrojos colocados en cada esquina del chasis como se aprecia en la figura A.4, a continuación se presentan las características principales de este sensor.

Figura A.1: Sensor HC-SR04

Voltaje de operación	3.3V - 5V
Rango de distancia	2 cm - 30 cm
Ángulo de detección	35 grados
Dimensiones	31 mm x 15 mm

Tabla A.4: Características sensor infrarrojo

Figura A.2: Sensor infrarrojo

Para el movimiento de los prototipos se utilizó un motorreductor 48-1, cuyas características se muestran en la tabla A.6.

Voltaje de operación	3V - 12V
Corriente de operación	100 mA
Relación de reducción	48:1
Par	800 gf/cm
Velocidades aproximadas	3 V = 80 rpm / 5 V = 120 rpm / 9 V = 300 rpm

Tabla A.5: Características motorreductores

Para el manejo de los motores se utilizó el módulo L298, sus características se muestran en la tabla A.3.

Voltaje de operación	6V - 35V
Voltaje de operación	5V
Corriente máxima	2A
Señales pwm	2
Dimensiones	43mm x 26mm x 43mm
Motores	2

Tabla A.6: Características sensor infrarrojo

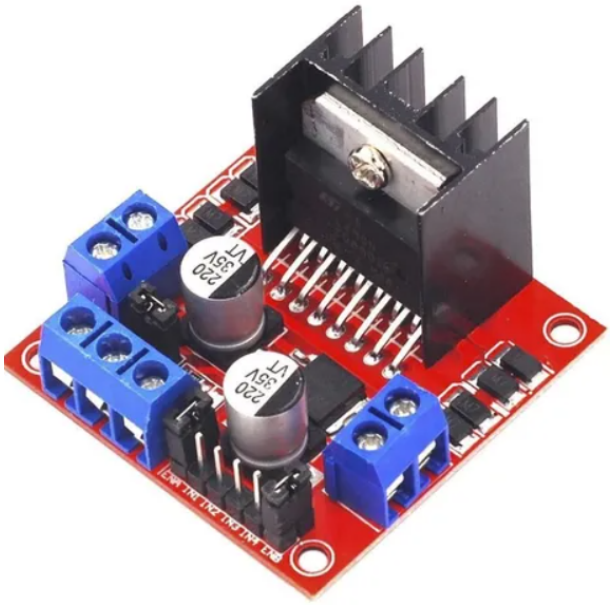


Figura A.3: Puente H L298

A.2 Diseño

El chasis de los WolfBots y la carcasa de la tarjeta Jetson Nano™ fueron impresos en 3D en PLA, como se puede ver en la figura A.4 la estructura de estos robots cuenta con dos plataformas, el chasis y la carcasa, siendo esta última exclusiva de la

tarjeta Jetson Nano™, en el chasis se encontraran los sensores infrarrojos, el sensor ultrasónico y el puente H de los motores.

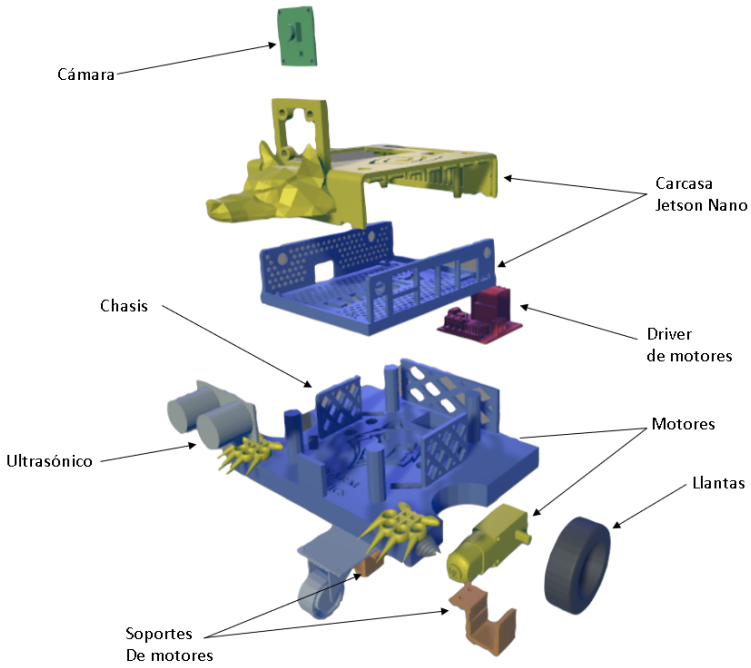


Figura A.4: Vista explosionada de la estructura de robot WolfBot

Apéndice B

Código en python

```
#*************************************************************
# Sistema de percepci n minimo para enjambres de robots RAOI
# basado en aprendizaje profundo
# MCIE Mecatr onica
# UANL, FIME
# Erik Ricardo Palacios Garza
#*************************************************************


#Declaracion de librerias

import jetson.inference
import jetson.utils
import time
import RPi.GPIO as GPIO
import numpy as np
import random
import argparse
import sys


#PINES PWM

output_pins = {
        'JETSON_XAVIER': 18,
        'JETSON_NANO': 32,
        'JETSON_NX': 33,
        'CLARA_AGX_XAVIER': 18,
}
output_pin = output_pins.get(GPIO.model, None)
if output_pin is None:
        raise Exception('PWM not supported on this board')

enable1=32
enable2=33

GPIO.setmode(GPIO.BOARD)
GPIO.setup(enable1, GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(enable2, GPIO.OUT, initial=GPIO.HIGH)
```

```
Enable1 = GPIO.PWM(enable1, 100)
Enable2 = GPIO.PWM(enable2, 100)

#PINES O/I

# Pin Definitons:
motor1a = 7
motor2a = 11
motor1b = 12
motor2b = 13


# sensor HC-SR04.
TRIG = 24
ECHO = 23
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

#Pines de motores
GPIO.setup(motor1a, GPIO.OUT)
GPIO.setup(motor2a, GPIO.OUT)
GPIO.setup(motor1b, GPIO.OUT)
GPIO.setup(motor2b, GPIO.OUT)

GPIO.setwarnings(False)


# Funciones de direccion
#####################################
def Atras():
        #Direccion motor A
        GPIO.output(motor1a,False)
        GPIO.output(motor2a,True)
        #Direccion motor B
        GPIO.output(motor1b,True)
        GPIO.output(motor2b,False)
#####################################
def Adelante():
        #Direccion motor A
        GPIO.output(motor1a,True)
        GPIO.output(motor2a,False)
        #Direccion motor B
        GPIO.output(motor1b,False)
        GPIO.output(motor2b,True)
#####################################
def Izquierda():
#Direccion motor A
        GPIO.output(motor1a,True)
        GPIO.output(motor2a,False)
        #Direccion motor B
        GPIO.output(motor1b,True)
        GPIO.output(motor2b,False)
#####################################
def Derecha():
        #Direccion motor A
        GPIO.output(motor1a,False)
        GPIO.output(motor2a,True)
        #Direccion motor B
        GPIO.output(motor1b,False)
        GPIO.output(motor2b,True)
#####################################
def Alto():
```

```
        #Direccion motor A
        GPIO.output(motor1a,False)
        GPIO.output(motor2a,False)
        #Direccion motor B
        GPIO.output(motor1b,False)
        GPIO.output(motor2b,False)


#PINES DE INFRARROJOS

INFRA1=16
INFRA2=15
INFRA3=18
INFRA4=19

GPIO.setup(INFRA1,GPIO.IN)
GPIO.setup(INFRA2,GPIO.IN)
GPIO.setup(INFRA3,GPIO.IN)
GPIO.setup(INFRA4,GPIO.IN)

Alto()

#Funciones importantes
class Machine:

    #Orientacion del robot
        def Orienta(self, output):
                m1=abs(output[1])
                dir=output[1]
                if m1>100:
                        m1=100
                if m1<28 and m1>10:
                        m1=28
                if m1<10:
                        m1=0

                Enable1.start(m1)
                Enable2.start(m1)
                if dir>0:
                        Izquierda()
                else:
                        Derecha()

    #Aproximacion del robot
        def Orienta_dist(self, output):
                m1=abs(output[1])
                dir=output[1]
                if m1>40:
                        m1=40
                if dir>0:
                        Enable1.start(m1)
                        Enable2.start(m1)
                        Adelante()
                else:
                        Enable1.start(m1)
                        Enable2.start(m1)
                    Atras()


    #Controladores
    def PD(self, input1, setpoint, previous_time, last_error, controlador,...
    cum_error, last_error_abs):
```

```
                if controlador==0:
                        kd=np.array([[0.003, 0], [0, 0.003]])
                        kp=np.array([[0.09, 0], [0, 0.09]])
                else:
                        kd=np.array([[0.005, 0], [0, 0.005]])
                        kp=np.array([[0.2, 0], [0, 0.2]])


                actual_time = time.time()
                elapsed_time= actual_time - previous_time

                error=setpoint-input1

                error_abs=abs(error)

        rate_error = (error - last_error) / elapsed_time
        last_error=error
                last_error_abs=error_abs
                previous_time= actual_time
                if controlador==0:
                        output= np.dot(kp,error) + np.dot(kd,rate_error)
                else:
                        output= np.dot(kp,error) + np.dot(kd,rate_error)
                return output, previous_time, last_error, last_error_abs, cum_error



#Par metros de redes neuronales
parser = argparse.ArgumentParser(description="Locate objects in a live camera stream using an object...
detection DNN.",
                                 formatter_class=argparse.RawTextHelpFormatter, epilog=...
                                 jetson.inference.detectNet.Usage() +
                                 jetson.utils.videoSource.Usage() + jetson.utils.videoOutput.Usage() +...
                                 jetson.utils.logUsage())

parser.add_argument("input_URI", type=str, default="", nargs='?', help="URI of the input stream")
parser.add_argument("output_URI", type=str, default="", nargs='?', help="URI of the output stream")
parser.add_argument("--network", type=str, default="ssd-mobilenet-v2", help="pre-trained model to load (see...
below for options)")
parser.add_argument("--overlay", type=str, default="box,labels,conf", help="detection overlay flags...
(e.g. --overlay=box,labels,conf)\nvalid combinations are:  'box', 'labels', 'conf', 'none'")
parser.add_argument("--threshold", type=float, default=0.5, help="minimum detection threshold to use")

is_headless = ["--headless"] if sys.argv[0].find('console.py') != -1 else [""]



new_data_detection=['detectnet2.py', '--model=models/experimento_final_ssd/ssd-mobilenet.onnx',...
'--labels=models/experimento3_ssd/labels.txt', '--input-blob=input_0', '--output-cvg=scores', '--output-...
bbox=boxes', 'csi://0']

new_data_classification=['imagenet.py', '--model=models/experimento_final_resnet/resnet18.onnx',...
'--input_blob=input_0', '--output_blob=output_0', '--labels=data/experimento_final/labels.txt', 'csi://0']


try:
        opt = parser.parse_known_args()[0]
except:
        parser.print_help()
        sys.exit(0)

# Cargando redes neuronales
```

```
net = jetson.inference.detectNet(opt.network, new_data_detection, opt.threshold)
net2 = jetson.inference.imageNet(opt.network, new_data_classification)

#Iniciando camara
input = jetson.utils.videoSource(opt.input_URI, argv=sys.argv)
output = jetson.utils.videoOutput(opt.output_URI, argv=sys.argv+is_headless)
font = jetson.utils.cudaFont()


rate_error=0
previous_time=time.time()
last_error=0
last_error_abs=0
bandera=0
bandera_datos=True
ultra=False
itera=0
etiqueta_estado=0

potencia_busqueda=35

ObjectsList=""

etiqueta="None"
c=0
c_2=0
c_control=0

distancia_ultra=15
Enable1.start(potencia_busqueda)
Enable2.start(potencia_busqueda)


#Rutina de lectura de sensor ultras nico
def ultrasonico():

        GPIO.output(TRIG, GPIO.HIGH)
        time.sleep(0.00001)
        GPIO.output(TRIG, GPIO.LOW)
        print ("Iniciando eco")
        t=0
        while True:
                if t==5:
                        break
                pulso_inicio = time.time()
                t=t+1
                if GPIO.input(ECHO) == GPIO.HIGH:
                        break
        while True:
                pulso_fin = time.time()
                if GPIO.input(ECHO) == GPIO.LOW:
                        break
        duracion = pulso_fin - pulso_inicio
        distancia = (34300 * duracion) / 2
        print ("Distancia: %.2f cm" %distancia)
        return distancia



#Bucle de comportamiento y captura de imagenes
while True:
```

```
    img = input.Capture()
    img_class = img
#Lectura de sensores infrarrojos
    FD=GPIO.input(INFRA1)
    FI=GPIO.input(INFRA2)
    TI=GPIO.input(INFRA3)
    TD=GPIO.input(INFRA4)

    ultrasonic= ultrasonico()
    if ultrasonic<=5:
            Alto()
            time.sleep(0.5)
            ultrasonic= ultrasonico()


    if not FD or not FI or not TD or not TI or ultrasonic<=10:
            etiqueta="obstaculo"

    #ESTADO DE BUSQUEDA
    if etiqueta=="fondo"or etiqueta=="None":
            etiqueta_estado=0
            g=0
            class_id, confidence = net2.Classify(img_class)
            class_desc = net2.GetClassDesc(class_id)

            if confidence*100 <= 80 or class_desc=="fondo":
                    class_desc2="fondo"
            else:
                    class_desc2=class_desc
            font.OverlayText(img_class, img_class.width, img_class.height, "{:05.2f}% {:s}".format...
            (confidence * 100, class_desc2), 5, 5, font.White, font.Gray40)
            etiqueta= class_desc2

            if etiqueta=="objetivo" or etiqueta=="robot":
                    Alto()
                    time.sleep(1)

    #Rutina de exploracion
            if etiqueta=="fondo":

                    if itera>=0 and itera<=7:
                            Enable1.start(potencia_busqueda)
                            Enable2.start(potencia_busqueda)
                            Derecha()
                    elif itera>7 and itera<=20:
                            Enable1.start(potencia_busqueda)
                            Enable2.start(potencia_busqueda)
                            Izquierda()
                    elif itera>20 and itera<=25:
                            Enable1.start(potencia_busqueda)
                            Enable2.start(potencia_busqueda)
                            Derecha()
                    elif itera>25:
                            Enable1.start(potencia_busqueda)
                            Enable2.start(potencia_busqueda)
                            Adelante()
                            if itera>=50:
                                    itera=0
            itera=itera+1

#Estado de atencion para estimulos de influencia
    if etiqueta=="objetivo" or etiqueta=="u" or etiqueta=="carne" or etiqueta_estado==1:
```

```
Adelante()
ultrasonic= ultrasonico()
if ultrasonic<=5:
        Alto()
        time.sleep(0.5)
        ultrasonic= ultrasonico()


etiqueta_estado=1
bandera_comida=0
centers=[]
classes=[]
rights=[]
lefts=[]
detections = net.Detect(img, overlay=opt.overlay)

for detection in detections:
        classes.append(detection.ClassID)
        rights.append(detection.Right)
        lefts.append(detection.Left)
        centers.append(detection.Center)
        if detection.ClassID==1 or detection.ClassID==3 or detection.ClassID==4:
                bandera_comida=1
                kind_food=detection.ClassID
                break
print(classes)
if bandera_comida==1:
        for i in range(0, len(classes)):
                if classes[i]==kind_food:
                        ObjectsList=centers[i]
                        right=rights[i]
                        left=lefts[i]
                        dist= right-left
                        bandera_comida=0
                        break
elif bandera_comida==0:
        detections=[]



if not detections:
        c_2=c_2+1
        if c_2==100:
                Enable1.start(potencia_busqueda)
                Enable2.start(potencia_busqueda)
                if random.random() > 0.5:
                        Derecha()
                else:
                        Izquierda()
                time.sleep(0.3)
                Alto()
                etiqueta="None"
                c_2=0


elif detections and ultra==False:
        if g==0:
                c_2=0
                c=0
                cum_error=0
                rate_error=0
                previous_time=time.time()
```

```
                                last_error=0
                                last_error_abs=0
                                g=1

                        etiqueta="Control"
                        entrada_control=ObjectsList[0]

                elif detections and ultra==True and bandera_datos==False:
                        g=0
                        c=0
                        c_2=0
                        cum_error=0
                        rate_error=0
                        previous_time=time.time()
                        last_error=0
                        last_error_abs=0
                        bandera_datos=True

                elif detections and ultra==True and bandera_datos==True:
                        c_2=0
                        referencia_angular=ObjectsList[0]
        error_angular=640-referencia_angular
        dist= right-left
                        referencia=900
                        set_point=np.array([0, referencia])
                        input1=np.array([0, dist])
                        output_PD, previous_time, last_error, last_error_abs, cum_error=Machine.PD(0,...
                        input1, set_point, previous_time, last_error,1, cum_error, last_error_abs)
                        Machine.Orienta_dist(0, output_PD)
                        if last_error[1]<100:
                                ultra = False
                                etiqueta="None"
                                Alto()
                                break
                                bandera_datos=False
                        elif abs(error_angular)>200:
                                ultra = False
                                etiqueta_estado=1
                                etiqueta="objetivo"
                                g==0
                                Alto()
                                bandera_datos=False

                else:
                        etiqueta="None"

#Estado de atencion para robots vecinos
    if etiqueta=="robot" or etiqueta_estado==2:
                etiqueta_estado=2
                bandera_lobo=0
                bandera_lobo_comida=0
                centers=[]
                classes=[]
                rights=[]
                lefts=[]
                detections = net.Detect(img, overlay=opt.overlay)

                for detection in detections:
                        classes.append(detection.ClassID)
                        rights.append(detection.Right)
                        lefts.append(detection.Left)
                        centers.append(detection.Center)
```

```
        if detection.ClassID==2:
                bandera_lobo=1
        else:
                bandera_lobo_comida=1


print(classes)
if bandera_lobo==1 and bandera_lobo_comida == 0:
        referencia = 1000
        for i in range(0, len(classes)):
                if classes[i]==2:
                        ObjectsList=centers[i]
                        right=rights[i]
                        left=lefts[i]
                        bandera_lobo=0
                        bandera_lobo_comida=0
                        dist= right-left
                        if dist>=1500:
                                detections=[]
                        break
elif bandera_lobo==0 and bandera_lobo_comida == 0:
        detections=[]

elif bandera_lobo==1 and bandera_lobo_comida == 1:
        referencia = 800
        for i in range(0, len(classes)):
                if classes[i]==1 or classes[i]==3 or classes[i]==4:
                        ObjectsList=centers[i]
                        right=rights[i]
                        left=lefts[i]
                        bandera_lobo=0
                        bandera_lobo_comida=0
                        break

else:
        referencia = 800
        for i in range(0, len(classes)):
                if classes[i]==1 or classes[i]==3 or classes[i]==4:
                        ObjectsList=centers[i]
                        right=rights[i]
                        left=lefts[i]
                        bandera_lobo=0
                        bandera_lobo_comida=0
                        breakif not detections:
        c_2=c_2+1
        if c_2==5:
                Enable1.start(potencia_busqueda)
                Enable2.start(potencia_busqueda)
                if random.random() > 0.5:
                        Derecha()
                else:
                        Izquierda()
                time.sleep(0.3)
                Alto()
                etiqueta="None"
                c_2=0


elif detections and ultra==False:
        if g==0:
                c_2=0
                c=0
```

```
                                cum_error=0
                                rate_error=0
                                previous_time=time.time()
                                last_error=0
                                last_error_abs=0
                                g=1

                        etiqueta="Control"
                        entrada_control=ObjectsList[0]


                elif detections and ultra==True and bandera_datos==False:
                        g=0
                        c=0
                        c_2=0
                        cum_error=0
                        rate_error=0
                        previous_time=time.time()
                        last_error=0
                        last_error_abs=0
                        bandera_datos=True

                elif detections and ultra==True and bandera_datos==True:
                        c_2=0
                        referencia_angular=ObjectsList[0]

                        error_angular=640-referencia_angular

                        dist= right-left
                        referencia=800
                        set_point=np.array([0, referencia])
                        input1=np.array([0, dist])
                        output_PD, previous_time, last_error, last_error_abs, cum_error=Machine.PD(0,...
                        input1, set_point, previous_time, last_error,1, cum_error, last_error_abs)
                        Machine.Orienta_dist(0, output_PD)
                        if last_error[1]<100:
                                ultra = False
                                etiqueta="None"
                                Enable1.start(potencia_busqueda)
                                Enable2.start(potencia_busqueda)
                                if random.random() > 0.5:
                                        Derecha()
                                else:
                                        Izquierda()
                                time.sleep(0.3)
                                bandera_datos=False
                                ultra = False
                                etiqueta_estado=2
                                g==0
                                Alto()
                                bandera_datos=False

                else:
                        etiqueta="None"

#Estado de evacion
        if etiqueta=="obstaculo":
                g=0
                if etiqueta=="robot2" and FD and FI and TD and TI and ultrasonic>20:
                        Enable1.start(potencia_busqueda)
                        Enable2.start(potencia_busqueda)
                        Adelante()
```

```
        etiqueta="fondo"

elif ultrasonic<=distancia_ultra:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Atras()
        time.sleep(1)
        if random.random() > 0.5:
                Derecha()
        else:
                Izquierda()
        time.sleep(0.5)

elif  FI and FD and TI and not TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Adelante()

elif FI and FD and not TI and TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Adelante()

elif FI and FD and not TI and not TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Adelante()

elif FI and not FD and TI and TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Izquierda()

elif FI and not FD and TI and not TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Izquierda()

elif FI and not FD and not TI and TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Izquierda()

elif FI and not FD and not TI and not TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Izquierda()

elif not FI and FD and TI and TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Derecha()

elif not FI and FD and TI and not TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Alto()

elif not FI and FD and not TI and TD:
        Enable1.start(potencia_busqueda)
        Enable2.start(potencia_busqueda)
        Derecha()
```

```
            elif not FI and FD and not TI and not TD:
                    Enable1.start(potencia_busqueda)
                    Enable2.start(potencia_busqueda)
                    Derecha()

            elif not FI and not FD and TI and TD or etiqueta=="Caja":
                    Enable1.start(potencia_busqueda)
                    Enable2.start(potencia_busqueda)
                    Derecha()
                    etiqueta="None"
            else:
                    Alto()
                    etiqueta="None"


#Control de orientaci n
    elif etiqueta=="Control" and ultra==False:
            referencia=640

            input1=np.array([0, entrada_control])
            if etiqueta_estado==1:
                    etiqueta="objetivo"
            elif etiqueta_estado==2:
                    etiqueta="robot"


            set_point=np.array([0, referencia])
            output_PD, previous_time, last_error, last_error_abs, cum_error=Machine.PD(0,...
            input1, set_point, previous_time, last_error, 0, cum_error, last_error_abs)
            Machine.Orienta(0,output_PD)
            if abs(last_error[1]) <150:
                    c_control=c_control+1
            if c_control==2:
                    ultra=True
                    bandera=0
                    c_control=0

    output.Render(img)
    output.SetStatus("{:s} | Network {:.0f} FPS".format(opt.network, net.GetNetworkFPS()))
    net.PrintProfilerTimes()
    if not input.IsStreaming() or not output.IsStreaming():
            Alto()
            break
```

Bibliografía

Alippi, C., S. Disabato y M. Roveri (2018), «Moving convolutional neural networks to embedded systems: the alexnet and VGG-16 case», en *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, IEEE, págs. 212–223.

Couzin, I. D., J. Krause, R. James, G. D. Ruxton y N. R. Franks (2002), «Collective memory and spatial sorting in animal groups», *Journal of theoretical biology*, **218**(1), págs. 1–11.

Giusti, A., J. Nagi, L. Gambardella y G. A. Di Caro (2012), «Cooperative sensing and recognition by a swarm of mobile robots», en *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, págs. 551–558.

He, K., X. Zhang, S. Ren y J. Sun (2016), «Deep residual learning for image recognition», en *Proceedings of the IEEE conference on computer vision and pattern recognition*, págs. 770–778.

Jaworek-Korjakowska, J., P. Kleczek y M. Gorgon (2019), «Melanoma thickness prediction based on convolutional neural network with VGG-19 model transfer learning», en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, págs. 0–0.

Jin, B., Y. Liang, Z. Han y K. Ohkura (2020), «Generating collective foraging behavior for robotic swarm using deep reinforcement learning», *Artificial Life and Robotics*, **25**(4), págs. 588–595.

Mehra, R. *et al.* (2018), «Breast cancer histology images classification: Training from scratch or transfer learning?», *ICT Express*, **4**(4), págs. 247–254.

Nagi, J., G. A. Di Caro, A. Giusti, F. Nagi y L. M. Gambardella (2012), «Convolutional neural support vector machines: hybrid visual pattern classifiers for multi-robot systems», en *2012 11th International Conference on Machine Learning and Applications*, tomo 1, IEEE, págs. 27–32.

Navarro, I. y F. Matía (2013), «An introduction to swarm robotics», *International Scholarly Research Notices*, **2013**.

Ordaz-Rivas, E., A. Rodriguez-Liñan, M. Aguilera-Ruíz y L. Torres-Treviño (2019a), «Collective tasks for a flock of robots using influence factor», *Journal of Intelligent & Robotic Systems*, **94**(2), págs. 439–453.

Ordaz-Rivas, E., A. Rodriguez-Linan y L. Torres-Trevino (2019b), «Collective Behaviors in Swarms of Builder Robots», *Research in Computing Science*, **148**, págs. 103–114.

Ordaz Rivas, E. d. J. (2020), *Colaboración emergente en enjambres de robots, con reglas inspiradas en el comportamiento de animales sociales*, Tesis Doctoral, Universidad Autónoma de Nuevo León.

Ray, S. (2018), «Disease classification within dermascopic images using features extracted by resnet50 and classification through deep forest», *arXiv preprint arXiv:1807.05711*.

Redmon, J., S. Divvala, R. Girshick y A. Farhadi (2016), «You only look once: Unified, real-time object detection», en *Proceedings of the IEEE conference on computer vision and pattern recognition*, págs. 779–788.

Redmon, J. y A. Farhadi (2018), «Yolov3: An incremental improvement», *arXiv preprint arXiv:1804.02767*.

Ren, S., K. He, R. Girshick y J. Sun (2015), «Faster r-cnn: Towards real-time object detection with region proposal networks», *arXiv preprint arXiv:1506.01497*.

Reynolds, C. W. (1987), «Flocks, herds and schools: A distributed behavioral model», en *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, págs. 25–34.

Schmickl, T., C. Möslinger y K. Crailsheim (2006), «Collective perception in a robot swarm», en *International Workshop on Swarm Robotics*, Springer, págs. 144–157.

Zhang, Z. (2016), «Derivation of backpropagation in convolutional neural network (cnn)», *University of Tennessee, Knoxville, TN.*

Zhao, Z.-Q., P. Zheng, S.-t. Xu y X. Wu (2019), «Object detection with deep learning: A review», *IEEE transactions on neural networks and learning systems*, **30**(11), págs. 3212–3232.

Resumen autobiográfico

Erik Ricardo Palacios Garza

Candidato para obtener el grado de
Maestría en Ciencias de la Ingeniería Eléctrica

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Tesis:
Sistemas de percepción mínimos para robots colectivos RAOI basados en Aprendizaje Profundo

Mi nombre es Erik Ricardo Palacios Garza, nací en la ciudad de San Pedro Garza García, Nuevo León, México, el 6 de septiembre de 1995, mis padres son Carmelina de María Garza Perez y Ricardo Palacios Herrera, en el año 2013 entro a la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León, en la carrera de ingeniero en mecatrónica, donde tuve la oportunidad de conocer diversos profesores que me fueron motivando y orientando en el aspecto profesional de mi vida, además de poder conocer la forma de trabajo en algunas empresas del área metropolitana de Monterrey, Nuevo León, llevándome a elegir mi camino actual, el cual trato de orientarlo en la medida de lo posible hacia la investigación, aun me falta mucho camino que recorrer pero voy paso a paso hacia mis objetivos.