UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Evasión de obstáculos con flujo óptico y redes neuronales para vehículos no tripulados

JORGE GÓMEZ VALDERRAMA

Profesor Guía: MATTHEW BARDEEN

Memoria para optar al título de
Ingeniero Civil en Computación

Curicó – Chile
mes, año

UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Evasión de obstáculos con flujo óptico y redes neuronales para vehículos no tripulados

JORGE GÓMEZ VALDERRAMA

Profesor Guía: MATTHEW BARDEEN

Profesor Informante: PROFESOR INFORMANTE 1

Profesor Informante: PROFESOR INFORMANTE 2

Memoria para optar al título de
Ingeniero Civil en Computación

El presente documento fue calificado con nota: _____

Curicó – Chile

mes, año

Dedicado a ...

AGRADECIMIENTOS

Agradecimientos a ...

TABLA DE CONTENIDOS

página

ÍNDICE DE FIGURAS

página

ÍNDICE DE TABLAS

página

RESUMEN

Aquí va el resumen (en Castellano)...

1. Introducción

1.1. Descripción del contexto

1.2. Definición del problema

Los cuadricopteros tiene la capacidad de mantenerse en vuelo sin la intervención de un piloto, ya sea sobre una posición específica o siguiendo una ruta programada, esto es posible gracias al controlador de vuelo que cada multirrotor tiene a bordo. Este controla cada movimiento que realiza y puede llegar a hacer muchas tareas de forma autónoma, como lo es despegar, aterrizar o desplazarse a un punto en específico en el aire.

Las prestaciones mencionadas son realizadas por la adición de dispositivos al controlador de vuelo, estos le entregan información en distintos ámbitos, la que puede ser la posición georeferenciada mediante GPS, la altura por la medición barométrica, los movimientos que realiza el cuadricoptero con sensores de aceleración lineal y angular (acelerómetros y giroscopios), entre otros.

Si bien con toda la información que el controlador de vuelo puede recopilar del entorno, las funciones que este puede cumplir de forma autónoma tiene un paradigma proactivo, es decir, estas deben ser planeadas en detalle de antemano, por ejemplo: si se requiere que el cuadricoptero siga una determinada ruta conformada por puntos de GPS, es necesario diseñar esta de tal forma que no interfiera con algún elemento con el que este puede colisionar.

Es por este motivo que las aplicaciones de conducción autónoma se realizan en espacios abiertos donde exista poca intervención del entorno, pero si se requiere que estos vuelen en espacios más confinados o donde falla la asistencia de georreferencia por ejemplo, se debe hacer un vuelo manual asistido por un piloto.

Entonces la carencia que presentan los multirrotores es la capacidad de ser reactivos ante eventos que ocurran en el entorno, si pudieran reaccionar frente a estos, sin ser previstos de antemano al momento de planificar el vuelo, los cuadricopteros podrían ampliar su rango de aplicaciones.

La falencia está en que no existe un sistema que proporcione una comprensión del entorno donde se realiza el vuelo y que permita tomar decisiones frente a eventualidades que puedan ocurrir durante este.

1.3. Propuesta de solución

Para solventar la falencia mencionada anteriormente se propone implementar un sistema que permita la detección de obstáculos y la evasión de estos, de esta forma agregar al cuadricoptero una herramienta que le permita enfrentar este tipo de problemáticas.

La información de objetos será capturada por medio de una cámara digital, el video entregado por esta se procesara para obtener el flujo óptico presente en las imágenes, véase sección 2.3, éste permite rescatar datos de donde se encuentran los objetos dentro de la imagen.

Una vez que se obtenga el flujo óptico, será entregado a una red neuronal, vease seccion 2.5, la que dara como salida la dirección a la cual se debe hacer un movimiento para evadir un obstáculo.

1.4. Objetivos

1.4.1. Objetivo general

Implementar un sistema que sea capaz de detectar obstáculos y entregar la información necesaria de como evadirlos, mediante flujo óptico y redes neuronales.

1.4.2. Objetivos específicos

- Definir herramientas que permitan la obtención de flujo óptico desde la captura de video.
- Implementación de una red neuronal que permita el procesamiento del flujo óptico.
- Entrenamiento de una red neuronal para que sea capaz de aprender cómo evadir obstáculos.
- Implementación de un robot omnidireccional que permite la obtención de datos para entrenar la red neuronal.
- Realizar pruebas con el robot para evaluar el comportamiento de la red neuronal ante obstáculos.

1.5. Alcances

- El sistema será probado con un robot omnidireccional que solo puede moverse en un plano de dos eje.
- La evasión de obstáculos solo realizará en un eje.
- Solo se evadirán objetos estáticos y de uno a la vez.
- Las pruebas serán realizada en una pista diseñada con este fin bajo condiciones controladas.

2. Marco teórico

En este apartado se describen elementos que son necesarios para la comprensión del problema y la implementación de la solución. Se detalla que es un cuadricoptero, como este se mueve en el aire y por qué es importante para la implementación. Se profundiza en la teoría del software a utilizar, lo que abarca: visión por computador; flujo óptico, su definición y cómo es posible calcularlo; redes neuronales, que son, como funcionan y aprenden. Del hardware se introduce a los robot omnidireccionales y algunos elementos que se utilizan en la construcción de este, que son los controladores de motores y el sensor óptico de movimiento.

2.1. Vehículo aéreo no tripulado

La problemática descrita fue obtenida de los vehículos aéreos no tripulados, por lo que importante describir que son y qué elementos de estos son importantes para a tomar en cuenta en la solución propuesta.

Un vehículo aéreo no tripulado, UAV (por sus siglas del inglés: *unmanned aerial vehicle*) o dron, es cualquier aeronave que no cuente con un piloto o tripulación a bordo, por lo que ésta puede volar de forma autónoma por medio de un computador a bordo o ser controlada remotamente por un humano [21].

Dentro de los vehículos aéreos no tripulados se encuentran los multirrotores o multicópteros, estas son aeronaves que generan sustentación por medio del giro de alas o hélices sobre un eje, siendo este conjunto llamado rotor. La Organización

de Aviación Civil Internacional (CIAO por sus siglas del inglés: *International Civil Aviation Organization*) los define como aeronaves que se mantienen en vuelo por las reacción del aire en uno o más rotores. En el caso de los multirrotores estos están compuestos por dos o más rotores, se diferencian de los helicópteros en la forma que logran el control y la estabilidad, estos tienen mecanismos que varían el paso de las hélices modificando su ángulo de ataque, en cambio los multirotores lo hacen por medio del cambio de velocidad relativa de cada rotor, produciendo cambios en el empuje y torque.

Un tipo de mutirrotor son los cuadricópteros los cuales tiene dos pares de rotores con hélices iguales y paso fijo, dos giran en sentido horario y los otros dos en sentido antihorario, esta configuración permite que el torque de cada rotor se anule con el rotor correspondiente que gira en sentido contrario, de esta forma el cuadricóptero se estabiliza y no gira sobre su propio eje [2].

Se describe este tipo de UAV porque la solución a entregar va a enfocada a este grupo de aeronaves, principalmente a los cuadricópteros que bordean los 250 mm de diámetro.

Como se mencionó anteriormente los movimientos de los multirrotores se logran variando la velocidad de cada rotor independientemente, referencia a la imagen, con esto el quadricóptero puede moverse sobre 3 ejes perpendiculares, por medio de los movimientos de *pitch*, *roll* y *yaw*, además de desplazarse por el eje Z variando su altitud.

Con el movimiento *pitch* el cuadricóptero rota sobre el eje X y se desplaza hacia adelante o atrás por el eje Y, cuando ejecuta el movimiento de *roll* rota sobre el eje Y y se desplaza de izquierda a derecha sobre el eje X y gira sobre el eje Z con el movimiento de *yaw*. Con esto se logra tener control sobre 4 grados de libertad.

Es necesario saber cómo son capaces de moverse los cuadricopteros, ya que en las pruebas a realizar se busca imitar los movimientos que estos realizan. De esta forma al capturar video se tenga una visión muy cercana a la que se tendría al hacerlo desde un cuadricoptero. Con la limitación de movimiento a solo dos ejes sobre una

superficie por medio de un robot omnidireccional, véase sección 2.6.

2.2. Visión por Computador

Computer Vision (visión por computador) es una disciplina que busca que los computadores puedan tener una comprensión de alto nivel sobre imágenes o videos digitales, por ejemplo, con el objetivo de automatizar tareas que el sistema visual humano ya puede hacer[6, 27, 25].

Dentro de las tareas de la visión por computador están: la adquisición, procesamiento, análisis, el entendimiento de imágenes digitales y la extracción de información multi dimensional del mundo real, para producir información numérica o simbólica [17, 23, 18, 13].

Dado que nosotros percibimos el mundo principalmente por medio de la vista, parecería sencillo traspasar nuestra experiencia a la visión por computador, pero en realidad es una tarea muy difícil, debido a la complejidad de cómo funciona nuestro cerebro. Contamos con múltiples sistemas que reciben distinta información segmentada desde el sistema visual e incluso otros sistemas, pero en el caso de la visión por computación solo contamos con una imagen (en el caso de video una secuencia de imágenes) y esto es todo lo el computador puede *"ver"*.

Cuando hablamos de una imagen nos estamos refiriendo a la información que nos es capaz de entregar la luz que es recibida desde el entorno o una escena. Esta información varia dependiendo del contexto en que se hable, si nos referimos a un marco biológico la luz incidirá en un ojo y las células en la retina generaran las señales eléctricas que el cerebro interpretará como imágenes, si hablamos de una cámara la imagen se formará en una película fogatica o en un sensor digital [8].

Dado el ámbito de este documento, es preciso profundizar en las imágenes digitales, esta puede ser definida como la integración y muestreo de información analógica (luz incidiendo en un sensor fotográfico) en un dominio espacial. Esta consiste en un arreglo rectangular de pixeles (x, y, u), cada uno contiene una ubicación $(x, y) \in \mathbb{Z}^2$

y un valor u, muestreado en una ubicación (x, y). $\in$ es conjunto de puntos (x, y) del arreglo rectangular [17]. Entonces podemos definir una imagen como:

$$I = \{(x, y) : 1 \leq x \leq N_{cols} \wedge 1 \leq y \leq N_{rows}\} \subset \mathbb{Z}^2 \tag{2.1}$$

2.3. Flujo óptico

El flujo óptico es uno de los elementos centrales en este documento. Es obtenido por medio del procesamiento de imágenes, con el fin de obtener información de la escena circundante. Esta informacion es la que será utilizada por la red neuronal posteriormente. En esta sección se describe que es el flujo óptico y como es calculado.

El flujo óptico es definido como el cambio de la estructura de la luz en una imagen, por ejemplo, en la retina de un ojo o en el sensor de una cámara, debido a movimiento relativo del observador y escena. Cuando los objetos se mueven frente a una cámara o esta se mueve en un entorno fijo, existe un cambio correspondiente a los movimientos en la imagen, estos cambios pueden utilizarse para recuperar información relativa al movimiento de las formas y los objetos.

Podemos definir un campo de movimiento, en el cual asignamos un vector de velocidad a cada punto de la imagen. En un instante en particular, un punto P_i en la imagen corresponde con algún punto P_0 en la superficie de un objeto. Los dos puntos son conectados por la ecuación de proyección. Si consideramos una protección de perspectiva, una línea se extiende de un punto en la imagen, pasa por el centro de la lente (en el caso de tratarse de una cámara), hasta un punto en la superficie de la escena.

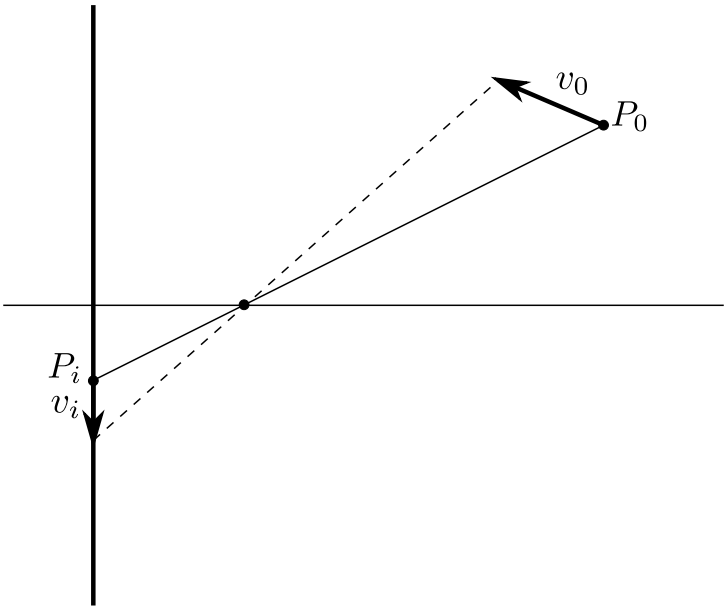


Figura 2.1: Proyección de un punto P_0 con movimoento v_0 en la escena a un punto P_i con movimiento v_i sobre la imagen.

El punto P_0 tiene una velocidad v_0 relativa a la cámara, esto induce un movimiento v_i en el punto P_i correspondiente en la imagen, como se puede ver en la figura 2.1[14].

Los vectores de velocidad producidos por el movimiento aparente son descritos de forma distinta, dependiendo del contexto en que se presenten, de un punto de vista biológico los cambios estructurados en los patrones de la luz en la retina de un ojo dejan la impresión de movimiento. En visión por computación los cambios en la escena son representados por serie de *image frames* (cuadros de imagen), la figura 2.2 muestra una secuencia de tres cuadros, mediante un muestreo espacial y temporal de la luz incidente en la imagen, es decir, como se desplazan los pixeles en la imagen a través del tiempo.

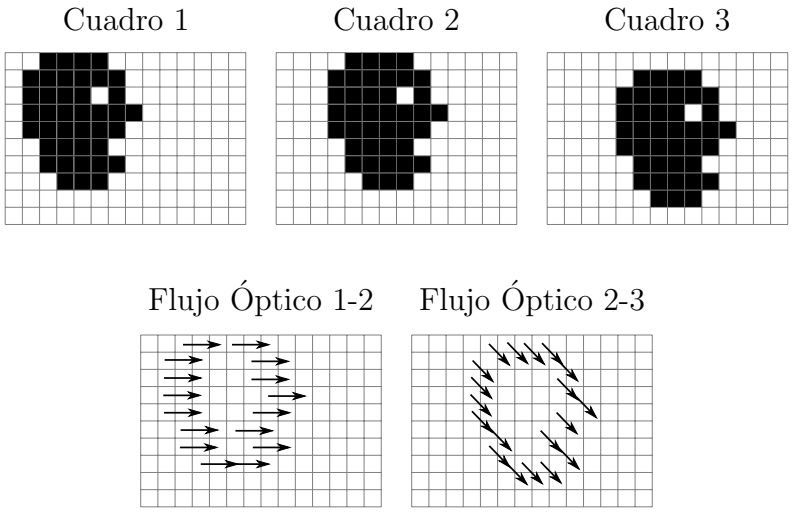


Figura 2.2: Ejemplo de captura del flujo óptico, por medio de cuadros de imagen consecutivos.

Para calcular el flujo óptico por medio el análisis de imágenes se utiliza el método de *Lucas-Kanade*, el cual funciona bajo dos suposiciones:

- La intensidad de un pixel en un determinado objeto no cambia entre cuadros consecutivos, refiriéndose a que un objeto no cambia en la escena, siempre la imagen proyectada de éste en la cámara es la misma.
- La segunda suposición dice que los vecinos cercanos a un pixel tiene un movimiento similar a este, esto permite evaluar una zona en la imagen.

Consideramos un pixel $I(x, y, t)$ en la primera imagen (x e y coordenadas dentro de la imagen y t un instante de tiempo) el cual se mueve una distancia de (dx, dy) en el siguiente cuadro después de un instante de tiempo dt, bajo las suposiciones mencionadas, podemos decir que:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.2}$$

Usando series de taylor aproximamos el lado derecho de la ecuación, removiendo términos semejantes y dividendo por dt obtenemos:

$$f_x u + f_y v + f_t = 0 \tag{2.3}$$

donde:

$$f_x = \frac{\partial f}{\partial x} \; ; \; f_y = \frac{\partial f}{\partial y} \tag{2.4}$$

Esta es la ecuación del flujo óptico, donde f_x y f_y son gradientes de la imagen, como f_t es gradiente del tiempo. Pero (u, v) son desconocidas, por lo que no podemos resolver la ecuación con dos incognitas. Aquí es donde el método *Lucas-Kanade* toma grupos de pixeles de 3x3, considerando las suposiciones mencionadas anteriormente, donde estos tiene el mismo movimiento. Ahora es posible resolver (f_x, f_y, f_t) ya que contamos con nueve ecuaciones y las mismas dos incógnitas, resultando con la siguiente ecuación:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i {f_{x_i}}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i {f_{y_i}}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix} \tag{2.5}$$

De esta forma se toma un grupo de pixeles y calculamos el flujo óptico sobre ellos, pero el método descrito funciona solo con pequeños movimientos, fallando en condiciones cunado los pixeles en la imagen sufren grandes movimientos. Para so-

lucionar este problema utilizamos el algoritmo de pyramid, el cual escala la imagen removiendo los pequeños movimientos y reduciendo los grandes movimientos a pequeños [1].

El algoritmo de *pyrmid* se encarga de realizar *downsampled*, este término se refiere a reducir la tasa de muestreo de alguna señal, a una imagen sucesivamente hasta algún límite dado, creando una colección de imágenes llamada pirámide. Existen dos tipos variantes del algoritmo de *pyramid*: la gaussiana y la laplaciana, en el método de *Lucas-Kanade* se utiliza la variante gaussiana, por lo que esta será descrita en detalle.

Para agregar a la pirámide una nueva capa G_{i+1}, debemos aplicar a la capa G_i un filtro gaussiano y luego eliminar las columnas y filas pares, esto nos genera una nueva imagen con un cuarto del área de la imagen en la capa G_i, iterando este proceso desde la imagen original G_0 producimos la pirámide entera [8].

Cuando se realiza un proceso de muestreo en una señal, en este caso de una imagen, puede producirse *alising*, el cual consiste en la distorsión de la señal orinal una vez que esta es muestreada. Para evitar esto el teorema de muestreo *Nyquist-Shannon* dice que es necesario muestrear una señal al doble de su frecuencia [12].

Es por esta razón que *pyramid* aplica un filtro gaussiano para la capa G_i de la pirámide antes de producir la capa G_{i+1}, esto reduce la frecuencia de la imagen en la capa G_i dando como resultado una imagen con menor distorsión en la capa G_{i+1}. Permitiendo tener una imagen de menor tamaño en una capa G_j, al final de la iteración, la cual tiene una baja distorsión frente a la imagen original en la capa G_0.

2.4. OpenCV

Como se mencionó anteriormente, la obtención del flujo óptico se hace por el procesamiento de imágenes, para esto se utiliza como herramienta a *OpenCV* que permitirá realizar la captura de imágenes desde una cámara digital, para luego calcular de estas el flujo óptico.

OpenCV (Open Source Computer Vision Library) es una librería open source (código abierto) de *computer visión*(visión por computador), distribuida bajo la licencia BSD, la cual está escrita en C y C++, siendo además multiplataforma. También existen interfaces para Python, Ruby, Matlab y otros lenguajes.

OpenCV está diseñado para ser eficiente computacionalmente y enfocado en ser utilizado en aplicaciones de procesamiento en tiempo real. Para esto, fue escrito en C optimizado y toma ventaja de los procesadores multi núcleo.

El proyecto fue iniciado por *Intel Research Lablets* en 1990 con el propósito de hacer la infraestructura de la visión por computador universalmente disponible y de esta forma ampliar a su vez OpenCV fue pasado a código abierto [8].

2.5. Redes neuronales

Otro elemento importante en este documento al igual que el flujo óptico son las redes neuronales. Una red neuronal es la encargada de tomar como entrada el flujo óptico de una determinada imagen y con respecto a esto la red toma una decisión, esta decisión indica la dirección en la cual debe moverse según lo que haya en la escena en ese momento. Esta sección da la definición de que es una red neuronal, como estas funcionan y son capaces de aprender.

Una red neuronal tiene la finalidad de ser un modelo de una red neuronal biológica, que recibe un número determinado de entradas, las que son procesadas dentro de la red y producen una o más salidas. Una red neuronal biológica está conformada por la interconexión de neuronas, las cuales son células nerviosas que reciben estímulos y conducen el impulso nervioso, por medio de un potencial de acción, a otras neuronas [9]. De la misma forma una red neuronal está formada por neuronas, a las que nos referimos por ahora como perceptrón, este funciona tomando varias entradas $x_1, x_2, \ldots$, para producir una sola salida binaria, como se muestra en la figura 2.3.

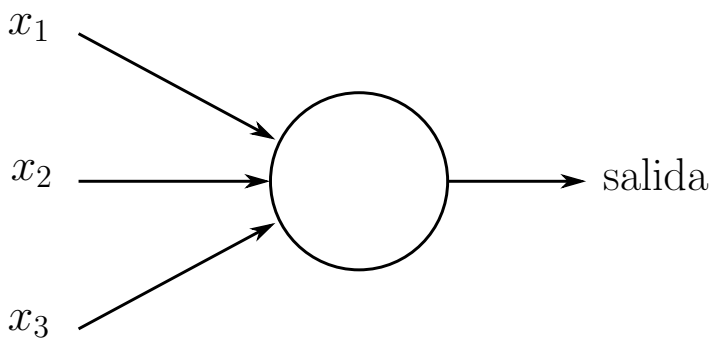


Figura 2.3: Perceptrón con tres entradas: x_1, x_2 y x_3 que genera un salida binaria, con valor 0 o 1.

En este caso se utilizan tres entradas, para computar la salida se introducen *weights* (pesos) $w1, w2, w3$, números reales que representan la importancia de cada entrada con respecto a la salida. Finalmente, la salida de la neurona es calculada como la sumatoria $\sum_j w_j x_j$, la salida binaria es 1 o 0 dependiendo de si el resultado de la sumatoria es menor o mayor al *threshold value* (valor de activación), el cual es un número real, parámetro de la neurona.

$$\text{salida} = \begin{cases} 0 & \text{si } \sum_j w_j x_j \leq \text{ activación} \\ 1 & \text{si } \sum_j w_j x_j > \text{ activación} \end{cases} \tag{2.6}$$

La red neuronal es conformada por la interconexión de perceptrones, los que se agrupan en *layers* (capas), de tal modo que las salidas de los perceptrones en la primera capa son a su vez las entradas para los perceptrones de las segunda capa, así sucesivamente por la totalidad de las capas que constituyan la red neuronal.

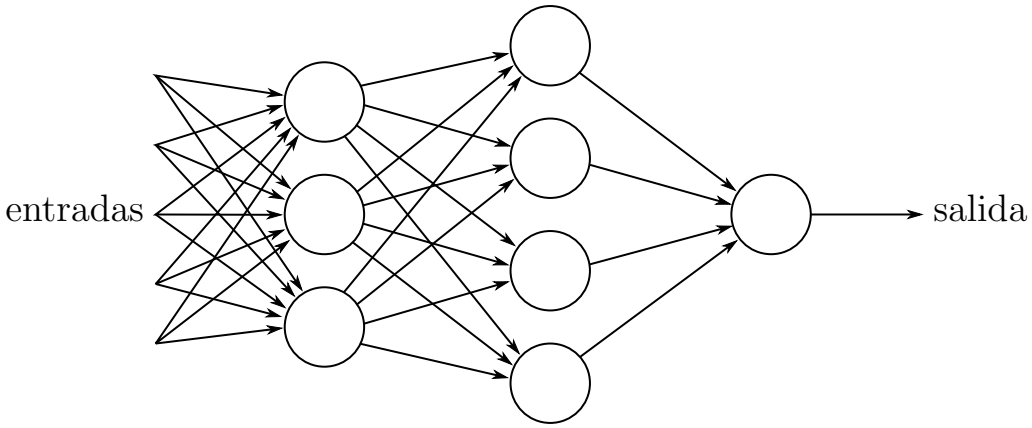


Figura 2.4: Red neuronal conformada por 3 capas, la primera con 3 perceptrones, la segunda con 4 y la tercera y última con 1 perceptrón.

En la definición de perceptrón se menciona que este solo cuenta con una salida, en la figura 2.4 se muestra que un perceptrón tiene múltiples salidas, pero de hecho es la misma única salida que se comparte cómo entrada para los perceptrones de la siguiente capa.

A la hora de implementar la red neuronal es posible realizar algunas simplificaciones, redefiniendo el comportamiento de los perceptrones, podemos suponer que las entradas y los pesos de estos están definidos como dos vectores, con lo que ahora para producir la salida se calcula el producto punto entre la entrada y los pesos, $w \cdot x \equiv \sum_j w_j x_j$, también se mueve el valor de activación al otro lado de la inecuación y remplazándolo por el termino de *bias* (sesgo del perceptrón), $b \equiv -activación$, ahora el comportamiento del perceptrón se define como:

$$\text{salida} = \begin{cases} 0 & \text{si } w \cdot x + b \leq 0 \\ 1 & \text{si } w \cdot x + b > 0 \end{cases} \tag{2.7}$$

El sesgo puede definirse cómo que tan fácil el perceptrón puede producir una

salida de valor 1, mientras más grande el sesgo más fácil producir una salida que sea 1.

Una de las principales características de las redes neuronales es su capacidad de aprender, para lograr esto la red debe variar tanto el valor de los pesos tanto como de los sesgos, para que de esta forma podamos tener la salida esperada a partir de una entrada determinada.

Para lograr este entrenamiento pequeños cambios en los parámetros de la red, pesos y sesgos, deben a su vez generar pequeños cambios en la salida, pero esto no sucede en las redes conformadas por perceptrones, caundo se aplica un pequeño cambio en los parámetros de un perceptrón produce que este cambie completamente su salida de 0 a 1, por ejemplo [20].

Este problema se soluciona remplazando los perceptrones por *sigmoid neuron* (neurona sigmoide), en este tipo de neurona se pueden aplicar pequeños cambios en sus parámetros y esos se verán reflejados también en pequeños cambios en la salida. Una neurona sigmoide funciona de forma similar a un perceptrón donde la salida es calculada por $\sigma(w \cdot x + b)$, σ es una función sigmoide, definida cómo:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{2.8}$$

Remplazando el parámetro de la función por los de la neurona:

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}. \tag{2.9}$$

Al ser sigma una función continua, ahora la salida de la red no es binaria, si no que valor real entre 0 y 1, como se muestra en al figura 2.5.

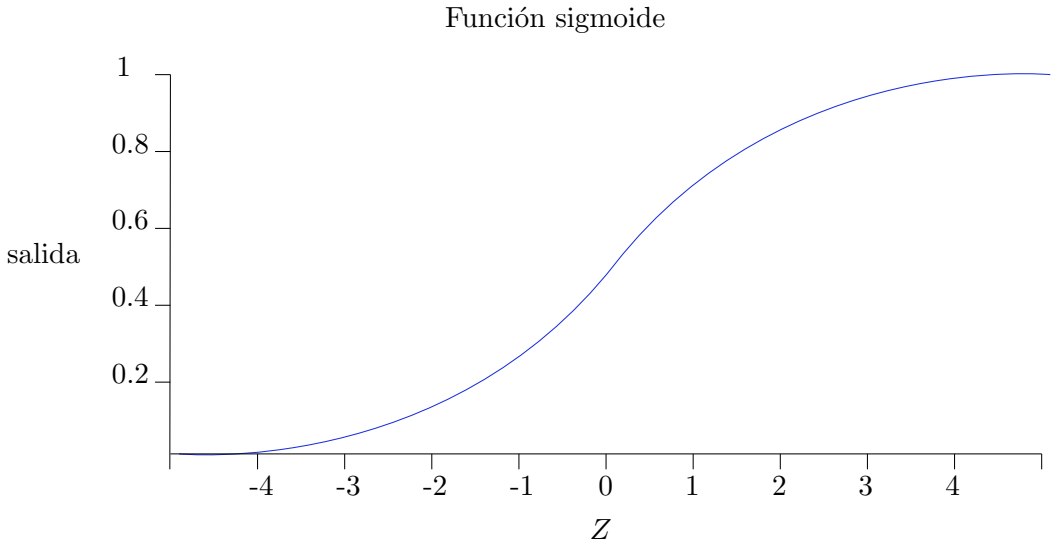


Figura 2.5: Función que utiliza una neurona sigmoide, como se puede ver la salida de la neurona será un valor continuo, entre 0 y 1.

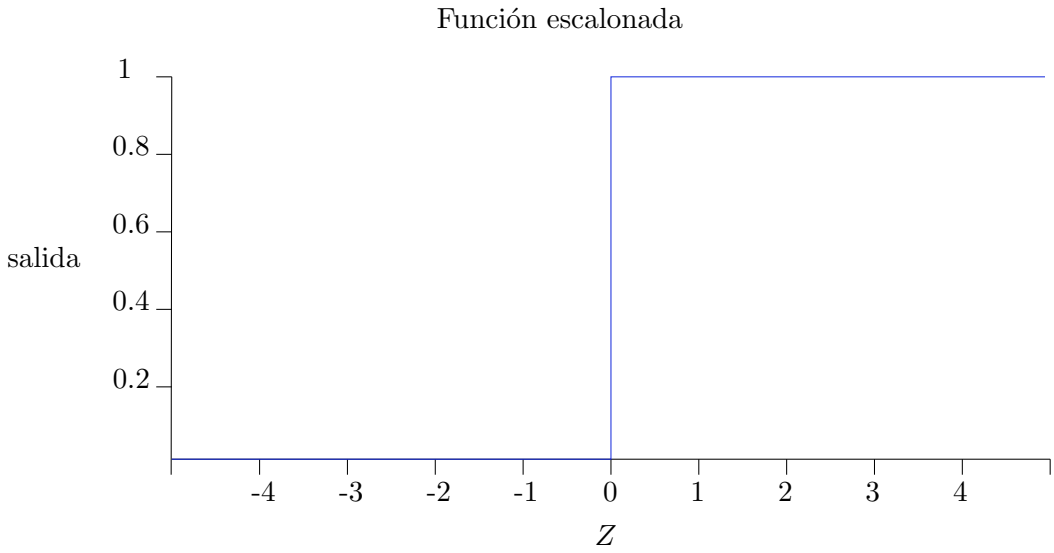


Figura 2.6: Función escalonada que utiliza un perceptrón, el que tiene una salida binaria, con valores 0 y 1.

Ya se mencionó anteriormente que la forma en que la red neuronal aprende es ajustando los valores de los pesos y sesgos de las neuronas que la conforman, para

conseguir este objetivo se utiliza el algoritmo de *backpropagation*. Primero se le entrega un vector de entrada a la red neuronal, se conoce de antemano la salida que esta debe generar con estos valores, la red computa la salida y esta se compara con la esperada mediante una función de costo. El algoritmo tiene el objetivo de determinar qué cambios hay que hacer a los parámetros de la red para reducir la función de costo.

Si en una neurona j de la capa l introducimos algún cambio en los parámetros Δz_j^l, entonces la salida de esta será $\sigma(z_j^l + \Delta z_j^l)$. Este cambio se propagara por la red para producir un cambio en la función de costo C en la medida de $\frac{\partial C}{\partial z_j^l}\Delta z_j^l$, la finalidad es encontrar un Δz_j^l que reduzca C. Si $\frac{\partial C}{\partial z_j^l}$ tiene una valor alto, entonces C se puede reducir eligiendo Δz_j^l que contrarreste $\frac{\partial C}{\partial z_j^l}$. En cambio sí $\frac{\partial C}{\partial z_j^l}$ tiene un valor cercano a cero, Δz_j^l no tendrá mucha influencia a C por lo que esa neurona cuenta con parámetros cercanos a los óptimos.

Denotaremos el error en la neurona j en la capa l como δ_j^l y para una neurona en la salida de la red δ_j^L donde:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{2.10}$$

El término $\frac{\partial C}{\partial a_j^L}$ determina qué tan rápido cambia C en función de la salida de la neurona j en al capa L. $\sigma'(z_j^L)$ expresa que tan rápido cambia σ en función de z_j^L.

La función de error cuadrático es la función de coste, pero esta calcula el promedio de error sobre una muestra. Para aplicar *backpropagation* solo se utiliza un valor de error y no una muestra de estos, por que la función C será:

$$C = \frac{1}{2}(y - a^L)^2 \tag{2.11}$$

Ahora podemos reescribir el error de la salida de la red δ_j^L utilizando la función de coste como:

$$\delta^L = (a^L - y) \odot \sigma'(z^L) \tag{2.12}$$

Generalizando para el resto de las capas de la red, δ^l en términos del error de la siguiente capa δ^{l+1}:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{2.13}$$

Con el error obtenido y propagado por la red solo falta modificar los pesos y sesgos, la tasa de cambio de C respecto a las sesgos de red se define como:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.14}$$

Y con respecto a los pesos:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.15}$$

2.6. Robot omnidireccional

En la realización de las pruebas es necesaria la captura de video, esta debe ser lo más similar a lo que se obtendría si se utilizara una cámara sobre un cuadricoptero.

Esa es la razón por qué se utiliza un robot omnidireccional, este permite imitar en cierto grado la forma en la cual puede moverse un cuadricoptero, con algunas limitantes. La siguiente sección explica cómo logra un robot omnidireccional hacer esto.

Un robot omnidireccional (referido en este documento inherentemente como robot o robot omnidireccional) es un robot que tiene la capacidad de moverse en cualquier dirección sin tener que rotar antes de moverse. Esto es posible por las ruedas omnidireccionales con que está construido el robot, estas tiene la capacidad de moverse libremente en dos direcciones, pueden rodar como una rueda normal o desplazarce lateralmente con las ruedas ubicadas perpendicularmente a lo largo de la circunferencia de la rueda, como se muestra en la figura 2.7.

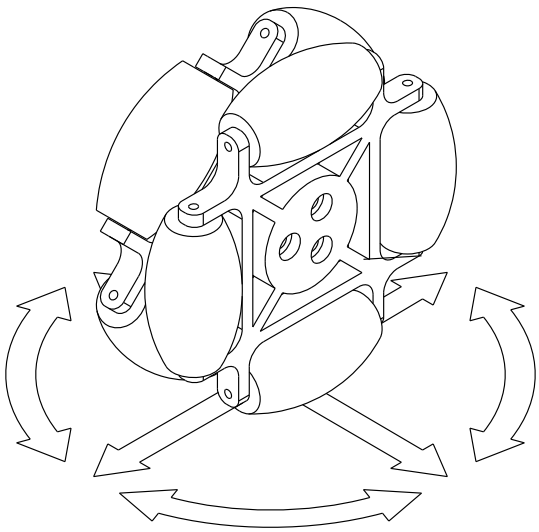
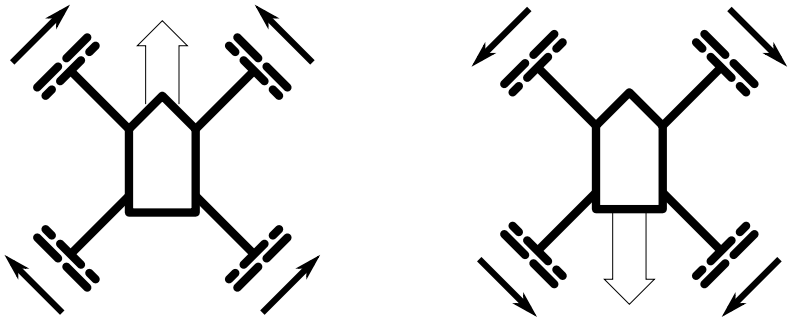


Figura 2.7: Direcciones en las cuales puede moverse una rueda omnidireccional.

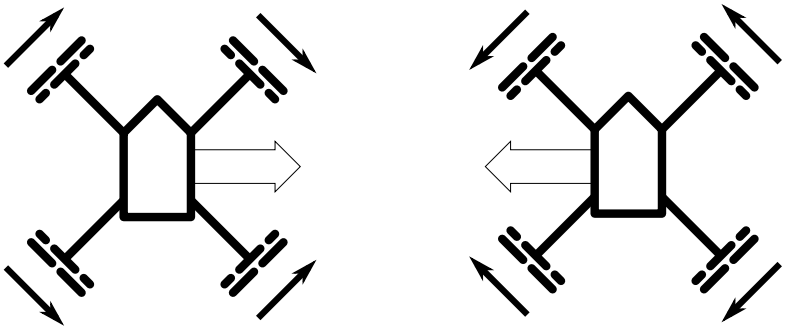
Esto permite que el robot puede trazar un camino en línea recta, en cualquier dirección, sin tener que hacer giros, además de poder combinar el trayecto con algún tipo de rotación para cambiar la dirección a la que apunta el robot, si es necesario. Esto es definido como un robot holonómico, en comparación con un robot normal no-holonómico el cual tiene 1,5 grados de libertad que puede desplazarse a través de los ejes coordenados X e Y, pero requiere movimientos complejos para poder moverse en alguno de los dos ejes. Un robot holonómico que tiene 2 grados de libertad puede moverse libremente por ambos ejes X e Y [16].

Para lograr que el robot se desplace en cualquier dirección es necesario calcular la velocidad y dirección de giro para cada rueda de forma independiente. En la figura 2.8 se muestra la dirección en la que deben girar las ruedas para mover el robot hacia adelante y atrás, en la figura 2.9 hacia la derecha e izquierda.



Movimiento hacia adelante Movimeinto hacia atras

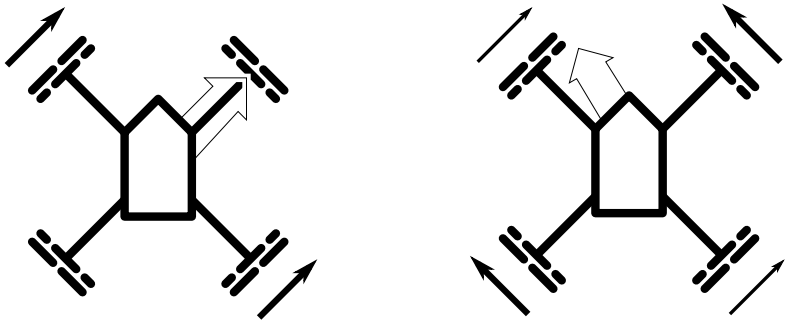
Figura 2.8: Las flechas muestran la dirección en la que las ruedas deben girar para lograr los movimientos hacia adelante y atrás.



Moviento hacia la derecha Movimiento hacia la izquierda

Figura 2.9: Las flechas muestran la dirección en la que las ruedas deben girar para lograr los movimientos hacia la derecha e izquierda.

En estos movimientos la velocidad de las ruedas siempre es la misma, solo cambia su dirección de giro, al modificar la velocidad el robot puede desplazarse en una dirección que combina los movimientos de adelante y atrás con los de derecha e izquierda, como muestra la figura 2.10.



Movimiento hacia adelante en 45° Moviemnto hacia adelante en en 120°

Figura 2.10: Izquierda movimiento del robot en 45° donde solo se mueve un par de ruedas. Derecha movimiento del robot en 120° donde un par de ruedas se mueve más rápido que otro, representado por el grosor la fecha que acompaña a las ruedas.

2.7. Microcontrolador

Para el control y comunicación del robot es necesario contar con alguna unidad de procesamiento, en este caso se hace uso de un microcontrolador. Este se encarga recibir las señales de radio control e interpretarlas, comunicarse vía puerto serial con el computador externo y controlar la dirección y velocidad de los motores.

Un microcontrolador es un pequeño computador fabricado en un solo circuito integrado, lo que es llamado sistema en un chip (SoC por sus siglas del inglés: *system on a chip*). Un microcontrolador consta de uno o más CPUs junto con memoria, periféricos de entradas y salidas programables y almacenamiento destinado al alojamiento del programa que ejecuta el microcontrolador.

Como se puede ver un microcontrolador está compuesto de distintos elementos embebidos que permiten su funcionamiento de forma autónoma con la adición poco o ningún componente, evitando la dependencia de elementos periféricos.

El microcontrolador utilizado es un ATmega328 y dentro de los elementos que lo conforman se encuentran:

CPU: Se encarga de la ejecución de programas, teniendo acceso la memoria, ejecutando cálculos, controlando periféricos y manejando interrupciones. La CPU utiliza

una arquitectura Harvard que se caracteriza en tener buses y memoria separados para los programas y los datos. Además de hacer uso de un set de instrucciones RISC (del inglés *reduced instruction set computer*) funcionando a una frecuencia de 16 MHz.

Registros: Espacios dentro de la CPU donde se almacenan datos que se utilizan para realizar las operaciones dentros de los demás componentes de la CPU, además se guardan los resultados de la ejecución de instrucciones. El ATmega328 cuenta con 32 registros de 8 bit cada uno.

Memoria: Se cuentan con tres tipos de memoria:

- SRAM: 2 KBytes de memoria interna compartida con los registros de entrada y salida.
- Flash Program Memory: Esta memoria reprogramable es compartida para almacenar los programas y el *Boot Loader*, con una capacidad de 32 KBytes.
- EEPROM: Memoria para almacenar datos con una capacidad de 1 KBytes.

Entradas y salida: Todos los pines del ATmega328 tiene la funcionalidad *Read-Modify-Write* lo que significa que la dirección de cada pin puede ser cambiada. Se cuenta con 23 pines de los que por sus funciones especiales quedan disponibles para libre uso 20 puertos. Los pines PB6 y PB7 son usados para el cristal externo y PC6 para reiniciar el microcontrolador. Estos pines también comparten funcionalidades especializadas como lo son: SPI, PWM, puerto serial, conversor análogo digital y ISCP [5].

El microcontrolador es utilizado bajo la plataforma de desarrollo *Arduino*, la que se define como una plataforma de código abierto basada en hardware y software fácil de usar. Esta proporciona el hardware por medio de las *Arduino Board* y el software en *Arduino programming language* y el IDE de desarrollo *Arduino Software* [4].

En específico se utiliza la placa *Arduino* UNO Rev 2, especificaciones en el anexo B, y el IDE en su versión 1.8.2.

2.8. Controlador para motores

Un elemento necesario para el funcionamiento del robot son los controladores para motores, estos componentes son un intermediario entre el microcontrolador y los motores. Estos permiten que las señales de control puedan actuar sobre los motores.

Un controlador para motores es un dispositivo que maneja el desempeño de un motor eléctrico, este incluye mecanismos para arrancar y detener el motor, seleccionar la dirección de rotación y regular la velocidad [24]. La mayoría de los controladores utilizan un circuito eléctrico llamado puente H (también referido como *full-bridge*, este permite aplicar corriente sobre una carga en cualquier dirección, con esto un motor de corriente continua puede girar en ambas direcciones [28].

El nombre es derivado de su representación gráfica, como se muestra en la figura 2.11.A, que tiene una forma de H. El circuito cuenta con cuatro interruptores, los que son accionados de apares, según la figura 2.11.B, S1 y S4 pueden estar en estado cerrado, mientras S2 y S3 en abierto, en esta configuración la corriente circula en una dirección a través del motor M, si en cambio S2 y S3 están cerrados y S1 y S4 abiertos como en la figura 2.11.C la corriente sobre el motor M circulara en la dirección opuesta. Los interruptores S1 y S2, al igual que S3 y S4 nunca son activados al mismo tiempo, ya que esto genera un corto circuito sobre la fuente de voltaje.

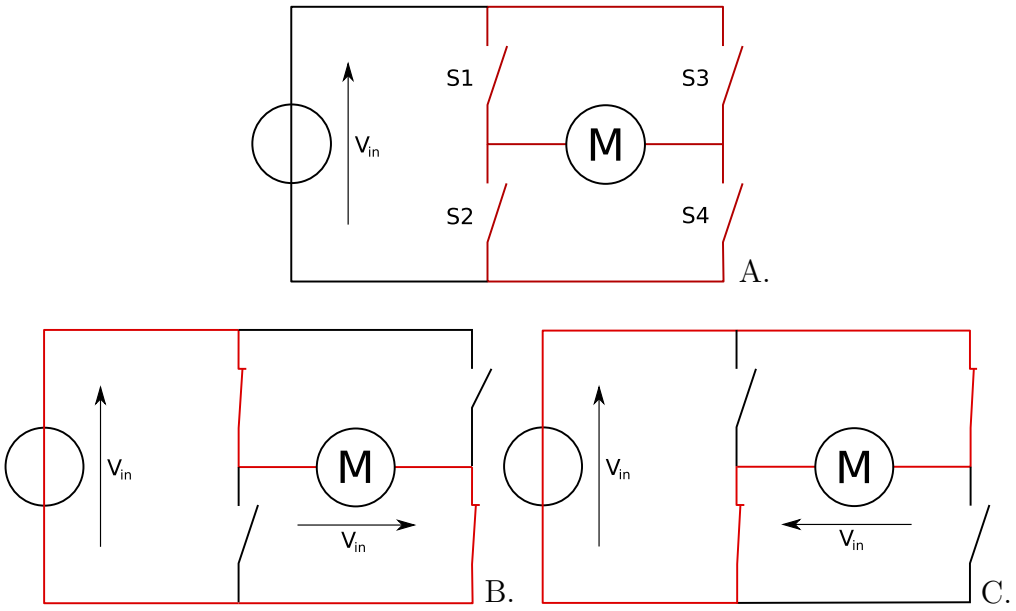


Figura 2.11: Funcionamiento de un puente H. A estructura del puente H. B interruptores S1 y S4 cerrado. C interruptores S3 y S2 cerrados.

El controlador utilizado es el L298, cual contiene dos puentes H compatibles con nivel lógico estándar TTL y puede manejar cargas inductivas, como motores de corriente directa. Cuenta con dos entradas de control para cada motor además de una para la habilitación del puente H. En la descripción de un puente H se mencionó el control sobre 4 interruptores, pero el controlador solo cuenta dos entradas de control por cada puente H, esto se debe a que dos interruptores nunca son activados al mismo tiempo, por ejemplo: S1 y S2. Esto permite que el control sobre estos interruptores puede ser compartida negando la señal de entrada a uno de estos, es decir, que cuando un interruptor está activado el otro siempre esta desactivado. Esta misma situación se da para los interruptores S3 y S4. En la figura 2.12 se muestra el control lógico que realiza el L298 sobre el puente H y la utilización de transistores de canal-N como interruptores.

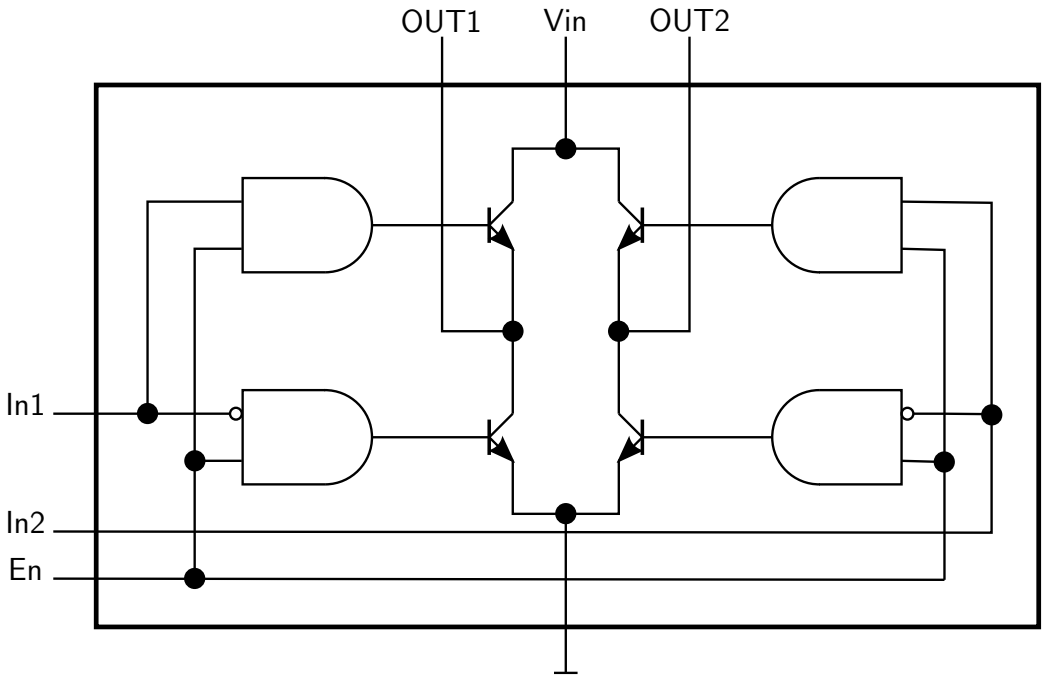


Figura 2.12: Control lógico dentro del L298 sobre uno de sus puente H.

Además se puede ver que el controlador recibe una tercera entrada En (sigla del inglés: *Enable*), está activa o desactiva el puente H dependiendo del estado lógico que presente: Si el estado es alto el puente H funcionara según corresponda con las entradas In1 e In2, pero si está en estado bajo desconecta ambas salidas OUT1 y OUT2 [26].

2.9. Sensor óptico de movimiento

Este es otro componente que es parte del robot omnidireccional, permite registrar los movimientos realizados sobre la superficie en la que se ha desplazado. Estos datos son necesarios ya que forman una estrecha relación con la salida que entrega la red neuronal y son son utilizados tanto para realizar el entrenamiento de la red como comprobar si este entrenamiento ha sido efectivo o no. A continuación se muestra el funcionamiento del sensor y cómo entrega la información de posicionamiento.

El sensor óptico de movimiento acoplado al robot que tiene la función registrar las trayectorias que este sigue, proviene de un *mouse* óptico (referido en este documento

indistintamente como *mouse* o *mouse* optico). Este utiliza una fuente de luz, en este caso un LED rojo y detector de luz como una matriz de fotodiodos, los que forman una pequeña cámara digital, para detectar el movimiento relativo a una superficie.

El sensor del *mouse* forma una imagen capturando las texturas sobre la superficie en que se encuentra, estas texturas son resaltadas por el ángulo de incidencia de la luz proveniente del LED. Se toma una continua sucesión de imágenes de la superficie, las que son comparadas unas con otras para determinar cuánto se ha movido el *mouse*.

Para determinar esto el *mouse* evalúa dos imágenes consecutivas y busca coincidencias de la primera imagen en la segunda, cuantificando cuanto se desplazado la segunda imagen sobre la primera, así se puede cuantificar el movimiento del *mouse*.

El *mouse* toma cientos de imágenes por segundo, dependiendo de qué tan rápido se esté moviendo cada imagen va a estar desplazada con la imagen anterior por una fracción de pixel o tanto como varios pixeles. El *mouse* procesa matemáticamente las imágenes usando correlación cruzada para calcular cuánto cada imagen, de la sucesión de imágenes, se ha desplazado desde la imagen anterior.

El protocolo de comunicación utilizado es PS/2 (*IBM Personal System 2*), el cual es serial, síncrono y bireccional. Utiliza dos líneas de comunicación DATA por la que se envían los datos y CLOCK que proporciona la señal de reloj, el dispositivo que en este caso es el *mouse* siempre es el que genera la señal de reloj, cuando las dos líneas están en alto el dispositivo puede enviar datos, pero el *host* tiene el control sobre la línea de datos, por lo que puede inhibir la comunicación enviando la señal de comunicación a low [11].

El *mouse* envía la información de estado de los botones, derecho, central e izquierdo, además del desplazamiento en el eje X e Y en un paquete de tres *byte*, que se muestra en la tabla 2.1. El movimiento es un entero de 9 *bite* a complemento 2, el *bit* más significativo es Y sign y X sign que aparece en el primer *byte*. Este valor representa la diferencia relativa a la posición del *mouse* desde la última vez que se envió el paquete de datos. El rango va de -255 a +255, si el rango es excedido el

correspondiente *bit* de *overflow* es activado [10].

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign	X sign	always 1	Btn central	Btn derecho	Btn izquierdo
Byte 2	Movimiento X							
Byte 3	Movimiento Y							

Tabla 2.1: Paquete de datos del movimiento del *mouse*.

3. Implementación

En este capítulo se detallan los elementos que fueron necesarios implementar para poder cumplir con el objetivo principal. Dentro de estos esta la forma como se obtiene el flujo óptico desde la captura de video; el funcionamiento de la red neuronal, el procesamiento de una entrada para producir una salida y el entrenamiento de esta; el diseño y construcción del robot omnidireccional.

3.1. Flujo óptico

El flujo óptico es la informacion que se obtiene desde la captura de imagenes y la que posteriormente sera entregada a la red neuroral. Para procesar las imagenes y obtener como resultado el flujo óptico se utiliza la librería *OpenCV* en un programa escrito en *Python*, esta entrega el metodo *calcOpticalFlowPyrLK* que implementa el algoritmo de *Lucas-Kanade*, más detalle en la sección 2.3, la cual recive como paramatros dos imagenes img_0 e img_1 y los puntos p_0 donde calcular el flujo optico.

Los puntos p_0 son una susecion de pares coordenados dentro de img_0, los que tiene una posicion predefinida a un intervalo regular de pixeles. El algoritmo usa los pixeles de img_0 correspondientes a los puntos definidos y calcula donde se han desplazado en img_1, dando como resultado una nueva susecion de puntos, estos nuevos puntos p_1 indican la nueva posicion qué tiene p_0 en img_1.

Una vez obtenido p_1 estos son procesados para dejar solo los puntos que han podido ser calculados correctamente. Esto se hace analizando la distancia de desplazamiento y las banderas de estado y error que acompañan a cada punto, si el

desplazamiento es mayor a 50 pixeles o una de las banderas esta como *False* ese punto es descartado y reemplazado por su valor correspondiente en p_0.

Las imagenes img_0 e img_1 son obtenidas secuencialmente *frame* por *frame* desde la funte de video, la cual puede ser un archivo de video o directamente de una camara digital. img_1 es el siguiente *frame* despues de img_0, cuando se procesa img_0 se copia el *frame* a img_1, para luego obtener el siguiente *frame* para img_0.

El flujo optico es representado por el desplazamiento que tiene P_0 en P_1, el que es en el plano de imagen tanto en eje X como en el Y.

A continuacion en el algoritmo 1 se describe el pseudocodigo del calculo del flujo óptico.

Algorithm 1 Calculo Flujo Óptico

```
1: function CalOpticalFlow
2:     frame ← cam.read()          ▷ Obtiene el siguiente frame de la fuente de video
3:     img1 ← gray(frame)          ▷ Combierte el frame a escala de grices
4:     p1, st, err ← calcOpticalFlowPyrLK(img0, img1, p0)
5:     img0 ← img1
6:     p1 ← goodPoints(p1, st, err)                                              ▷
7:     return p1
8: end function
```

3.1.1. Captura de flujo óptico

Para obtener el flujo óptico se utiliza una cámara de digital conectada por USB con una resolución de 640x480 píxeles, Con formato de pixel YUV 4:2:2, velocidad de obturación de hasta 1/30 seg

De las imágenes obtenidas desde la cámara se calcula el flujo óptico en una matriz de 21x5 puntos, cada punto está separado por 30 píxeles uno del otro. La matriz se extiende a todo el ancho la imagen y desde los 200 hasta 330 píxeles de alto.

Los puntos se ubican en esta posición para así obtener la información que entrega

la pista de pruebas.

El plano de la cámara está ubicado perpendicularmente al del desplazamiento del robot, por lo que solo se entrega a la red neuronal el flujo óptico que es calculado en el eje X de la imagen, es decir, la cuantificación del desplazamiento lateral de los pixeles en la imagen.

Por cada *frame* en el que se calcula el flujo óptico se obtiene un arreglo con un tamaño de 105 elementos que proporciona el desplazamiento de los puntos con respecto a la matriz antes descrita. Además del desplazamiento que tuvo el robot sobre el eje X de la pista, en la figura 3.1 se muestra un ejemplo de esto. Esta es la información que posteriormente se le entrega a la red neuronal.

```
[15.1345, 45.7654, 75.4358,... 615.5472, 645.8256, 675.1234], [0.75]
```

Figura 3.1: Ejemplo del flujo óptico obtenido de un *frame* más el desplazamiento del robot.

El algoritmo 2 muestra el pseudocodigo de como se realiza la captura del flujo óptico.

Algorithm 2 Captura de datos

```
1: function DataCapture
2:     data_input ← [ ]
3:     data_output ← [ ]
4:     data_mouse ← [ ]
5:     while video_capture do
6:         optical_flow_calc ← calcOpticalFlow()
7:         optical_flow_data ← getXComponent(optical_flow_calc)   ▷ Obtiene el
   compoenete X del flujo óptico
8:         x, y ← getMovement()          ▷ Obtiene el desplazamiento del robot
9:         data_input.append(optical_flow_data)
10:        data_output.append(normalize(x))
11:        data_mouse.append([x, y])
12:    end while
13:    saveData(data_input, data_output, data_mouse)
14: end function
```

La información obtenida de cada *frame* es almacenada en tres arreglos distintos, para el flujo óptico, el movimiento normalizado del robot en el eje X y la información completa del movimiento tanto en el eje X como en el Y. Cuando termina la captura de video se almacenan los tres arreglos en un archivo serializado. Por cada recorrido del robot por la pista se tiene una captura continua de video, entonces también se tiene un archivo con el flujo óptico y el desplazamiento realizado.

3.2. Red neuronal

La red neuronal es una adaptación de la implementada por *Michael Nielsen* en del libro *Neural Networks and Deep Learning* escrita en *Python*[19], está tiene la finalidad de recibir la información visual obtenida por el cálculo del flujo óptico y direccionar correctamente el robot por la pista. Aquí se documenta el funcionamiento de la red y el método que se utiliza para hacer que esta aprenda.

3.2.1. Funcionamiento

Para representar la estructura de la red, la que cuenta con *weights* y *biases* (pesos y sesgos) por cada neurona, se utiliza un arreglo multidimensional por cada parámetro. Para los sesgos se utiliza una matriz de dos dimensiones, así obtenemos b_{ij} que representa el sesgo para la *j-ésima* neurona en la *i-ésima* capa. La representación de los pesos se hace con una matriz de tres dimensiones, dando w_{ijk} que se traduce como el peso de la *j-ésima* neurona de la *i-ésima* capa con la *k-ésima* neurona de la i+1-*ésima* capa.

La obtención de la salida de red neuronal, también llamado *feedforward*, se hace ejecutando para cada capa la siguiente función:

$$a' = \sigma(z) \tag{3.1}$$

donde z:

$$z = (wa + b) \tag{3.2}$$

Si aplicamos la función σ a la *i-ésima* capa usamos la función sigmoide σ en cada elemento del arreglo z, z contiene el resultado del producto punto entre los pesos y el arreglo a más el sesgo de cada neurona. Donde a es la salida de la i-1-*ésima* capa o si es la primera la entrada de la red. Dando como resultado la salida a' la cual es el resultado *i-ésima* capa. El pseudocodigo se mustra en el algoritmo 3.

Algorithm 3 Red neuronal

```
1: function FEEDFORWARD(a)
2:     for b, w ← biases, weights do
3:         a ← σ((w · a) + b)
4:     end for
5:     return a
6: end function
```

La implementación del algoritmo de *backpropagation* consta de realizar el proceso de *feedforward* para una determinada entrada y poder calcular con su salida esperada y la siguiente ecuación de error:

$$\delta^L = (a^L - y) \odot \sigma'(z^L) \tag{3.3}$$

Donde a^L y z^L pertenecen a la última capa L, el arreglo δ^L representa la influencia de la salida de la red en la funcion de costo. Ahora se obtiene la difencia de los sesgos y pesos de la capa L:

$$\begin{aligned} b^L &= \delta^L \\ w^L &= \delta^L \sigma(z^L)^t \end{aligned} \tag{3.4}$$

Este procedimiento se propaga hacia atrás en la red por las capas $L-1, L-2, L-3...$ hasta alcanzar la primera capa. Para luego actualizar los parametros de red aplicando las diferencias obtenidas bajo un coeficiente de aprendizaje, este determina cuanto se deben modificar los parametros existentes en la red con respecto a las diferencias obtenidas del algoritmo de *backpropagation*.

3.2.2. Entrenamiento

Para entrenar la red neuronal se utiliza como entrada el flujo óptico obtenido de la captura de video de la pista de pruebas y como salida esperada la dirección en la cual se mueve el robot sobre el eje X de la pista. Para modificar los parametros al interior de la red (pesos y sesgos), que se traduce como el aprendizaje de la red, se utiliza la técnica de *backpropagation*, véase sección 2.5.

La implementación de red reorganiza de forma aleatoria los datos de entrenamiento e itera el entrenamiento un número determinado de veces. Al final de cada iteración se valida el entrenamiento con los datos de prueba, así se evalúa el transcurso del entrenamiento. Al finalizar la iteracion se guarda la red que tuvo un mejor desempeño en la validacion del entrenamiento.

La generación de set de datos del cual la red aprende se hace por medio de un esquema *on-the-fly* [22], el cual tiene como objetivo que la red neuronal imite el comportamiento de un humano al realizar la misma tarea, la cual es desplazar el robot por la pista de pruebas. Por lo que el flujo óptico y la dirección del robot se obtiene de la conducción manual del robot.

La salida esperada de la red neuronal, que representa la direccion en la cual debe moverse el robot mientras se desplaza por la pista y que es utilizada como parámetro en el entrenamiento por *backpropagation*, es obtenida por el seguimiento de los movimientos que realice el robot por medio de un sensor óptico de movimiento. La arquitectura de la red neuronal tendrá una sola salida la que estará normalizada entre cero y uno, por lo que la dirección del robot será representada por un solo valor.

3.3. Robot omnidireccional

El diseño del robot está basado en uno del repositorio de diseños 3D thingiverse (https://www.thingiverse.com/thing:167923), de este se tomó el concepto de las ruedas omnidireccionales las que se rediseñaron para este robot en espesifico.

3.3.1. Diseño del robot

La construcción consta de dos ruedas paralelas rotadas sobre el eje de giro 45° una de la otra, unidas por medio de tres tornillos. Cada una de estas ruedas cuanta a su vez con cuatro ruedas o rodillos en su circunferencia colocadas perpendicularmente separadas cada una por 90°, esto permite tener un solapamiento con los rodillos de la otra rueda ubicada perpendicular a esta. Los rodillos vas fijos a un eje de 2mm que los atraviesa y puede girar libremente en el soporte de la rueda. Cada rueda tiene un diámetro de 80mm y un ancho de 40mm, en la figura 3.2 se puede ver la construcción de esta.

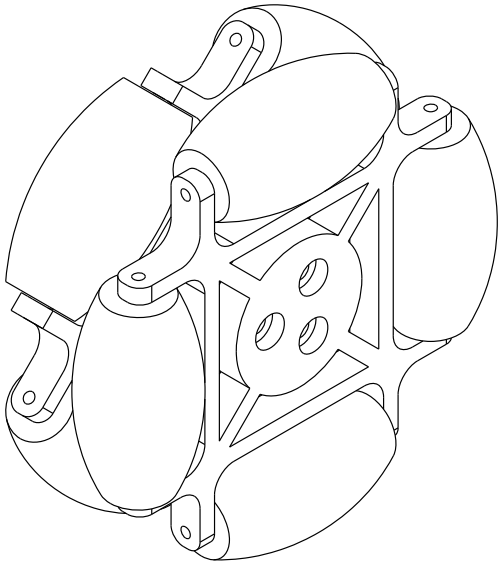


Figura 3.2: Diseño de rueda omnidireccional.

Cada rueda cuenta con un acople, que en un extremo va atornillado a la rueda por tres tornillos y en el otro va sujeto a presión al eje de la caja reductora del motor.

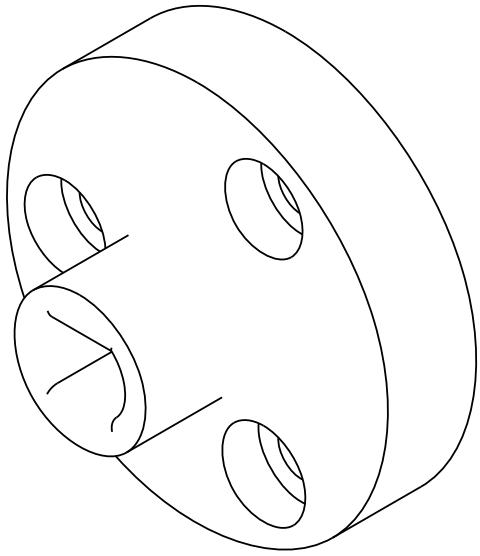


Figura 3.3: Acople para unir las ruedas del robot con la caja reductora de los motores.

A su vez los motores y su respectiva caja reductora están unidos por una extensión que permite unirlos al cuerpo principal, de esta forma se puede modificar el diámetro total del robot sin tener que modificar también el cuerpo principal, 3.4

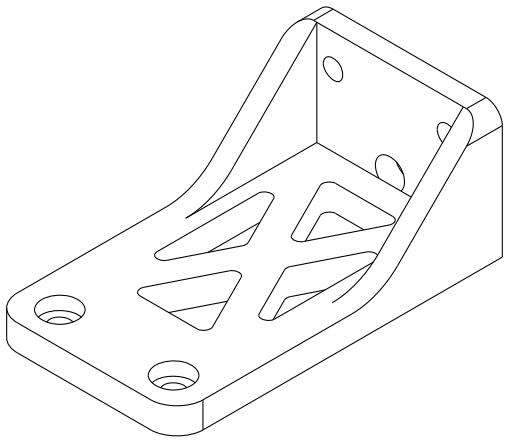


Figura 3.4: Extensión para unir los motores.

En el cuerpo principal se posicionan el resto de los componentes necesarios para el funcionamiento, estos son el microcontrolador, los *driver* para los motores, las

baterías, el receptor de radio, la cámara digital y sensor de movimiento. El cuerpo fue diseñado de acuerdo a las dimensiones de cada componente, de esta manera se pueden colocar de manera precisa, además de usar las fijaciones para las cuales fueron diseñados, que se muestra en la figura 3.5.

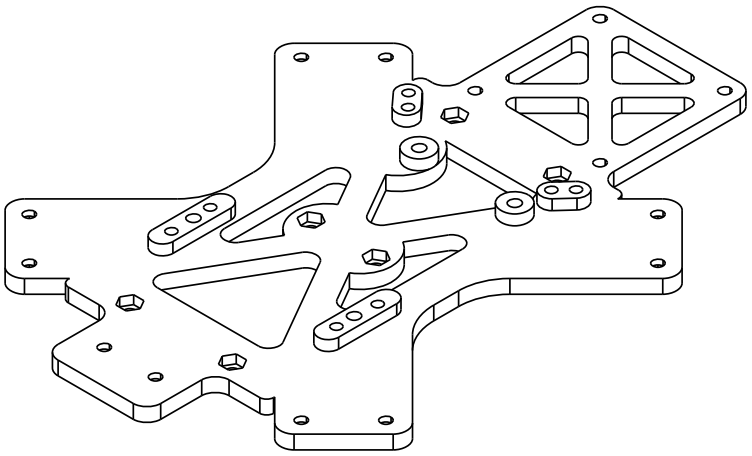


Figura 3.5: Cuerpo principal donde se alojan todos los componentes del robot omnidireccional.

El compartimiento para las baterías es ubicado debajo de cuerpo principal, este permite la fijación de baterías de litio estándar 18650 colocadas en serie, permite colocar un total de 3 baterías. El diseño se puede ver en la figura 3.6.

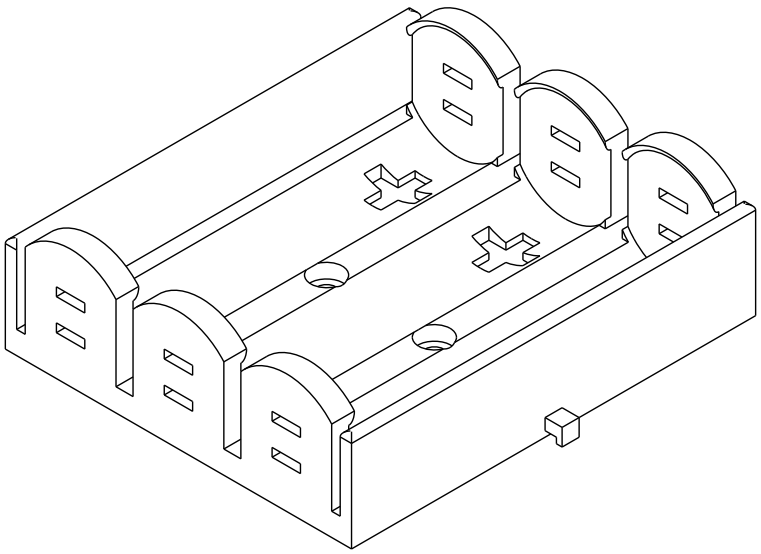


Figura 3.6: Compartimiento para colocar en serie las tres baterías de litio 18650.

Debajo de este, en contacto con la superficie donde se desplaza el robot, va el soporte del sensor óptico de movimiento, en la figura 3.7 se puede ver que el soporte cuenta con perforaciones por las que se atornilla al cuerpo principal por medio de separadores, para ajustar la altura del soporte para que quede posicionado de forma precisa sobre la superficie.

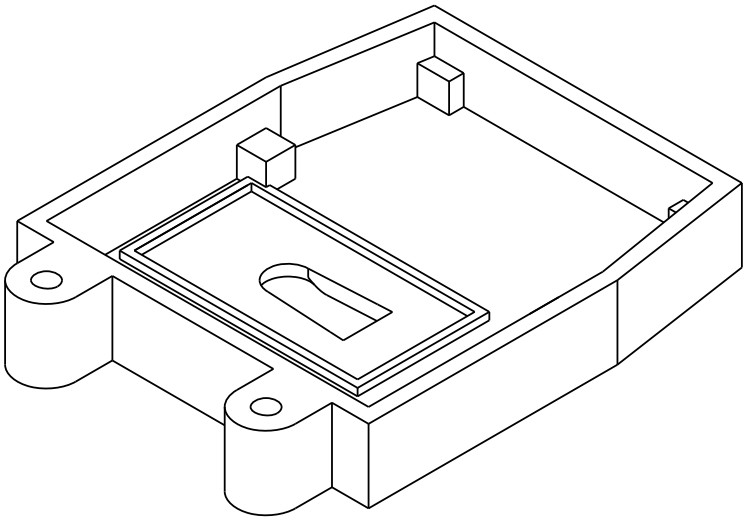


Figura 3.7: Soporte para la instalación del sensor óptico que registra las trayectorias realizadas por el robot.

Al frente del rebot va instalada la cámara digital, esta va sobre un soporte que se ajusta a sus dimensiones, el que a su vez va unido al cuerpo principal, en la figura 3.8 se muestra el diseño, que incluye una ranura para posicionar el cable de la cámara.

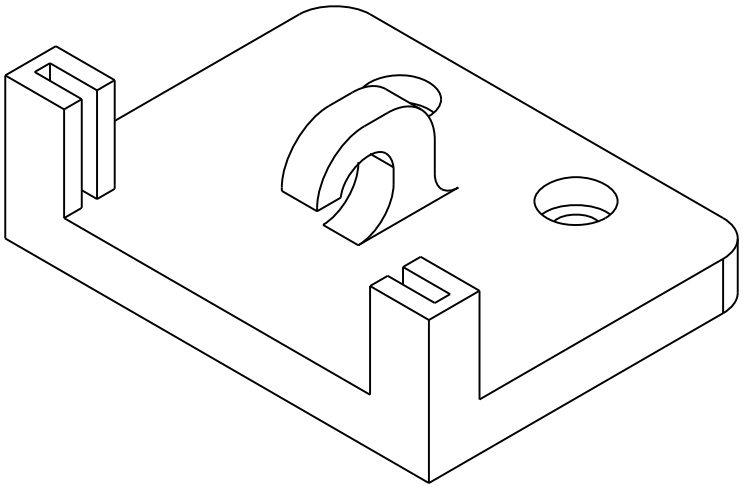


Figura 3.8: Soporte sobre el cual va la cámara digital.

El ensamble completo con todos los componentes diseñados para la construcción

del robot se puede ver en la figura 3.9. Todos los acoples mecánicos fueron realizados con tornillos de métrica 3 (M3), cordatos una medida específica para cada unión.

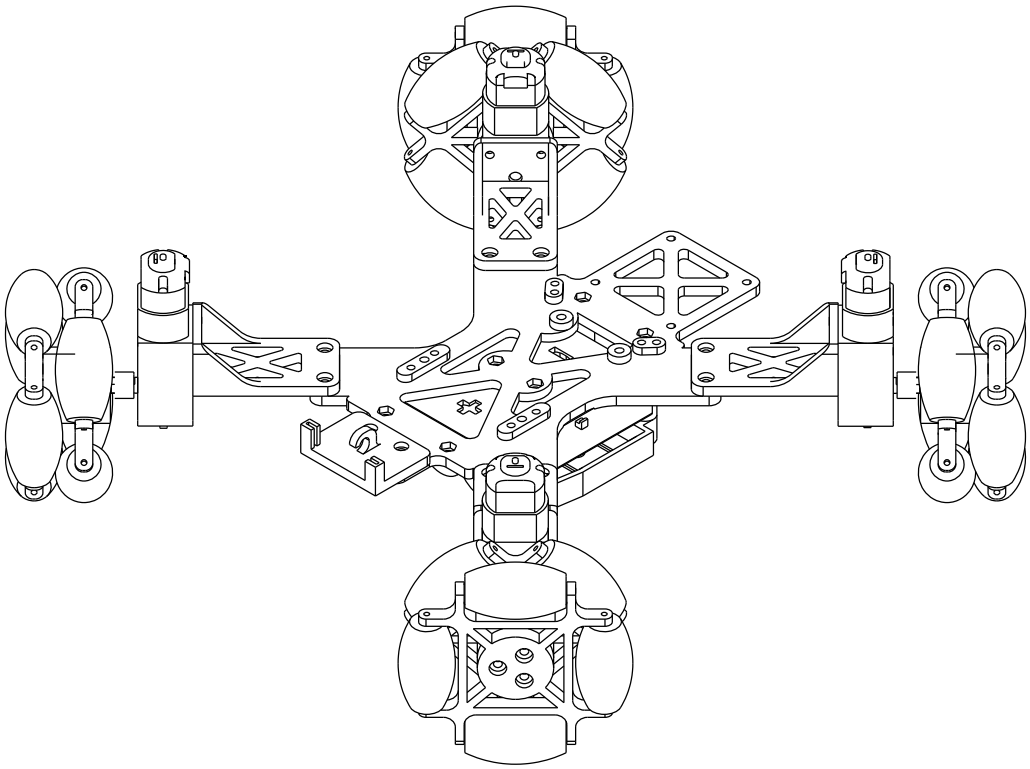


Figura 3.9: Ensamble completo del robot omnidireccional.

Los componentes mencionados y diseñados específicamente para el robot (ruedas omnidireccionales, extensiones para los motores, cuerpo principal, compartimiento para baterías, soporte para la cámara digital y soporte para el sensor óptico) fueron diseñados con la herramienta FreeCAD, un modelador 3D parametrico. Para el proceso de fabricación se utiliza Sli3r, que combierte los modelos 3D a instrucciones para la impresora 3D. Las piezas son construidoas con impresión 3D de filamento fundido (FFF del termino en inglés: *Fused Filament Fabrication*) con las siguientes especificaciones:

- Material: PLA 1.75mm
- Diametro boquilla: 0.4mm

- Altura de capa: 0.2mm
- Temperatura de boquilla: 220°
- Temperatura placa: 60°
- Velocidad impresion: 30mm/s
- Velocidad recorrido: 60mm/s
- Relleno: 15 %
- Patron relleno: rectilíneo

La espesificación complera de la configuracion de la impresora se puede ver en el anexo C

3.3.2. Control de las ruedas

El desplazamiento del robot es producido por cuatro motores, estos no van en conexión directa con las ruedas, si no que por medio de una caja reductora y esta transmite el movimiento del motor a las ruedas. Los motores son de tamaño estándar 130 alimentados con 12v de corriente continua (DC) con una caja reductora TT a 90°, de relación 120:1, como se muestra en la figura 3.10.

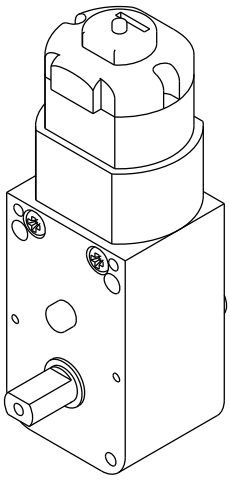


Figura 3.10: Motor con caja reductora del robot omnidireccional.

Los motores son controlados por el *driver* (controlador) l298n, este cuenta con dos puentes H, uno por motor, por lo que el robot utiliza dos controladores. Los controladores son alimentados con 12v y estos son los que suministran la energía a los motores.

La fuente de energía del robot es proporcionada por baterías, se utilizan tres baterías de litio 18650 conectadas en serie, cada batería es de 3,7v, por lo que en total en conjunto se tiene un voltaje nominal de 11.1v, con las baterías cargadas al 100 % de su capacidad su voltaje es de 12,6v.

Los movimientos del robot son realizados posteriormente al cálculo de las velocidades y dirección de giro de cada rueda, esta tarea es realizada por el microcontrolador, el utilizado es un ATmega328P bajo la infraestructura de Arduino UNO. El microcontrolador toma como entrada la velocidad y dirección a la cual debe moverse y realiza los cálculos para obtener los valores asociados a cada motor. Las salidas de control van dirigidas a los controladores de los motores, se utilizan tres líneas por cada motor, figura 3.11, dos para controlar la dirección del motor y la tercera para la velocidad, por lo que se tiene un total de doce salidas de control para todos los motores.

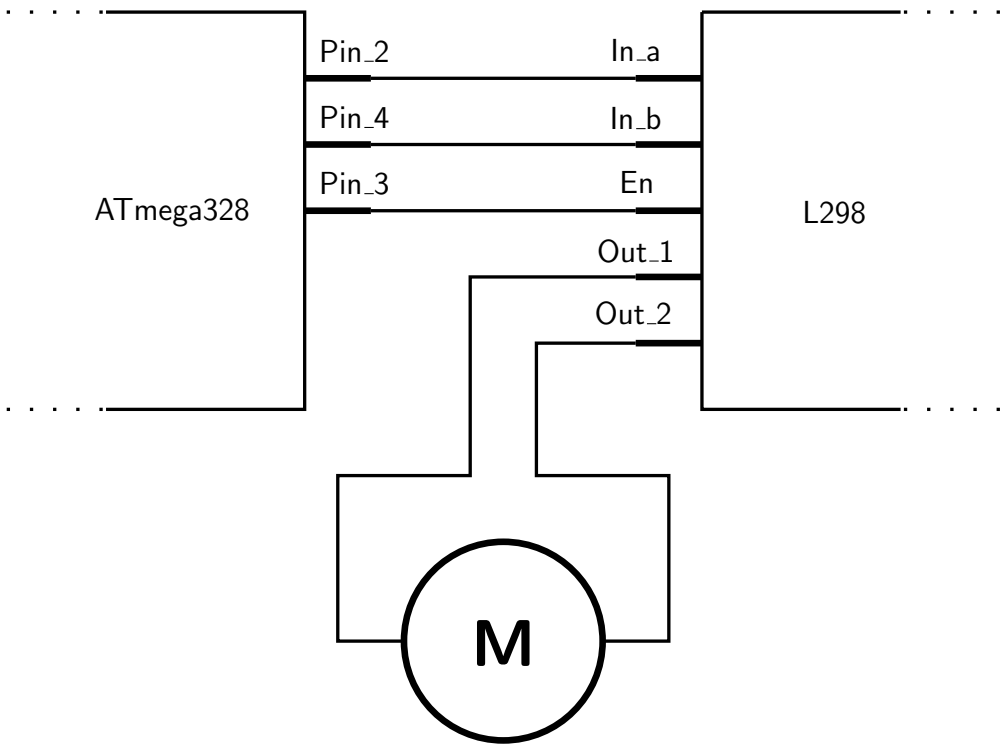


Figura 3.11: Control del microcontrolador ATmega328 sobre uno de los motores, por medio del controlador L298.

El control sobre los motores se hace de la siguiente manera: En el controlador L298N las salidas reflejan el estado en el que se encuentran las entradas, por lo que OUT1 tendrá el mismo estado que In1, lo mismo sucede cuando con OUT2 y In2, siempre y cuando En este en estado lógico alto. La tabla 3.1 muestra el comportamiento del controlador con respecto a las distintas entradas.

Entradas			Salidas	
In1 = H	In2 = L	En = H	OUT1 = H	OUT2 = L
In1 = L	In2 = H	En = H	OUT1 = L	OUT2 = H
In1 = X	In2 = X	En = L	OUT1 = L	OUT2 = L

Tabla 3.1: Comportamiento de las salidas del controlador L298N. L = estado lógico bajo, H = estado lógico alto y X = cualquier estado.

De esta forma variando el estado de las entradas es posible cambiar la dirección

hacia donde giran los motores, ahora para establecer la velocidad a los motores se utiliza la tercera entrada En con una señal *PWM* (sigla del inglés: *Pulse-width modulation*) que mediante la variación del *duty cycle* (ciclo de trabajo) logra establecer la variación de la velocidad. El ciclo de trabajo se define como el tiempo en que la señal se encuentra en estado alto, por ejemplo, en un ciclo de trabajo del 50 % la mitad del tiempo la señal está en alto y la otra mitad en bajo, en un ciclo del 25 % la señal está en alto solo el 25 % y el 75 % restante del tiempo está en estado bajo [7], com muestra la figura 3.13.

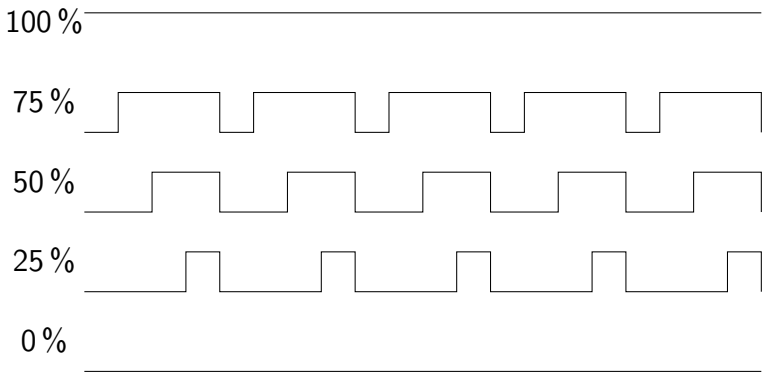


Figura 3.12: Señal *PWM* con distintos ciclos de trabajo.

Esta señal de control es producida por el microcontrolador a una frecuencia de 490 Hz, con una resolución de 8 bit, es decir, el ciclo de trabajo puede ser configurado con un rango de valores entre 0 y 255, siendo 0 un ciclo de trabajo del 0 % y 255 del 100 % [3].

3.3.3. Enlace de radio

Para dar al robot su dirección y velocidad se utiliza un enlace de radio 2.4GHz el que proporciona hasta seis canales de comunicación, para el control de robot solo se utilizan dos. Cada canal se lee desde el receptor de radio por separado por el microcontrolador, utilizando una entrada por canal. La señal se codifica mediante *PWM* donde la duración del pulso determina el valor del canal, cada pulso se produce en serie uno tras otro, este valor va en el rango de 1000μs a 2000μs.

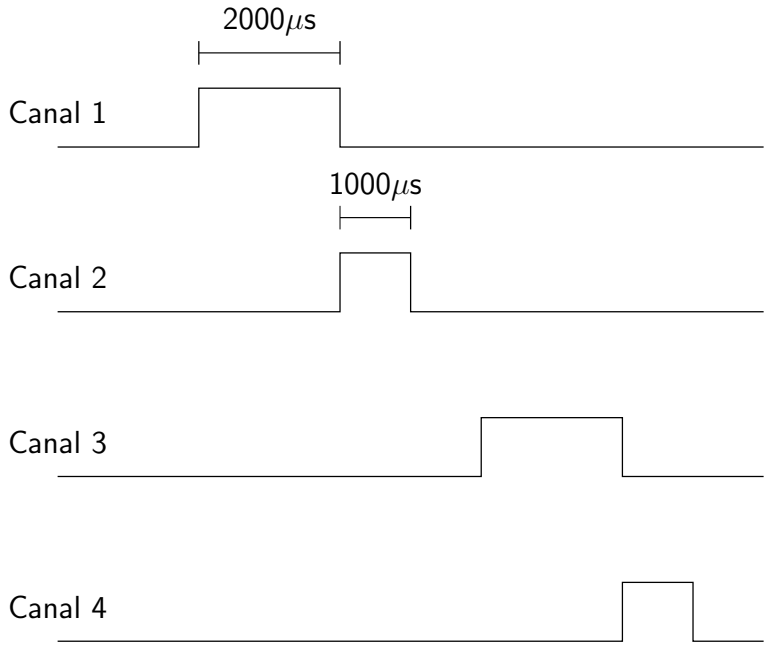


Figura 3.13: Señal PWM de distintos canales, el pulso tiene una tiempo de duracion minimo de 1000μs y máximo de 2000μs.

Para obtener el valor de cada canal desde el receptor con el microcontrolador se lee la señal de un canal y se toma el tiempo, en milisegundos, que esta señal permanece con un valor alto, el tiempo registrado corresponderá al valor de dicho canal, esto se repite para el resto de los canales.

Los canales utilizados corresponden a uno y dos (ch1 y ch2), de estos se normalizan sus valores a un rango entre -500 y 500, con estos se conforma un vector $\vec{d}$ con coordenadas (ch1, ch2), como muetsra la figura 3.14, con este vector se calcula posteriormente el movimiento del robot.

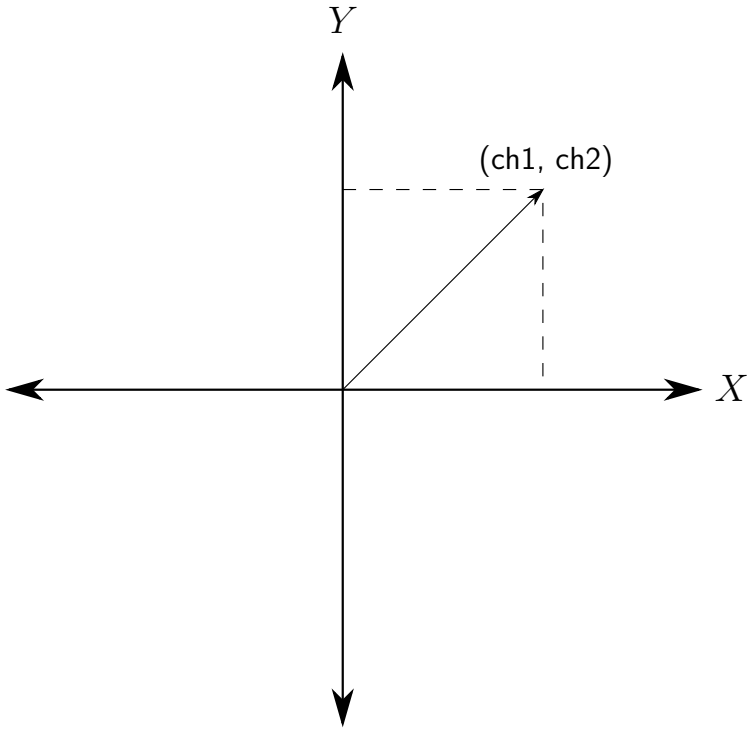


Figura 3.14: Vector formado por los valores de los canales uno y dos obtenidos desde el receptor de radio.

3.3.4. Movimiento

Primero se necesita establecer un marco de referencia, el robot es capaz de desplazarse en dos dimensiones sobre un plano, este está formado por los ejes X e Y. Cuando se menciona que el robot se mueve hacia adelante hace su recorrido avanzando sobre el eje Y, al contrario, cuando se mueve hacia atrás retrocede en el eje Y, como esta reprecentado en la figura 3.15. Ahora cuando se mueve a la derecha, avanza en el eje X y a la izquierda retrocede. Haciendo una comparación con los cuadricopteros, el movimiento hacia adelate y atrás del robot corresponde al movimiento de *pich* en el cadricoptero y los de derecha e izquierda al *roll*. No está dentro de los alcances contemplar el movimiento de *yaw*.

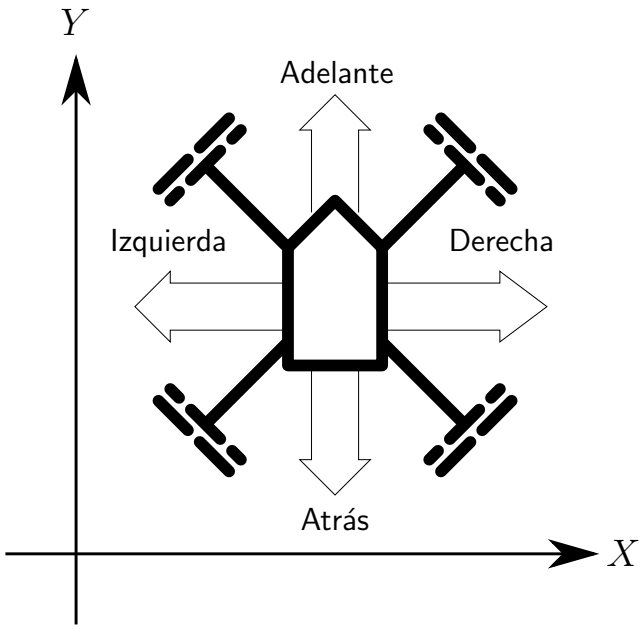


Figura 3.15: Desplazamiento del robot sobre el plano conformado por los ejes X e Y.

El movimiento del robot está determinado por dos parámetros, hacia donde debe moverse y a que velocidad. Con esto se tiene un vector, que llamaremos $\vec{d}$, donde la dirección de este es la misma que debe seguir el robot y la magnitud es la velocidad del movimiento. Dado que las ruedas del robot están dirigidas en distintas orientaciones no es posible moverse directamente en la dirección de $\vec{d}$. Cada rueda se mueve en una dirección distinta con su propio vector de movimiento, por lo que para poder desplazarse en $\vec{d}$ se tiene que realizar la suma de los vectores de cada rueda, de tal forma que el resultado coincida con $\vec{d}$.

Si bien el robot cuenta con cuatro ruedas, con los movimientos que se tiene contemplados hacer (delante, atrás, derecha e izquierda) siempre un par de ruedas se mueven en paralelo, como muestra la figura 3.16, así los movimientos de robot pueden descomponerse en dos vectores: $\vec{r_1}$ y $\vec{r_2}$, la magnitud de estos representa la velocidad a la que gira cada rueda. La suma de estos debe dar $\vec{d}$, como muestra la figura 3.17.

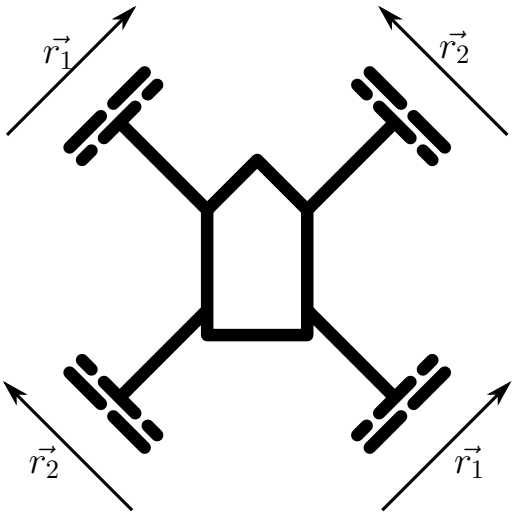


Figura 3.16: $\vec{r_1}$ y $\vec{r_2}$ pares de ruedas que se mueven de forma paralela.

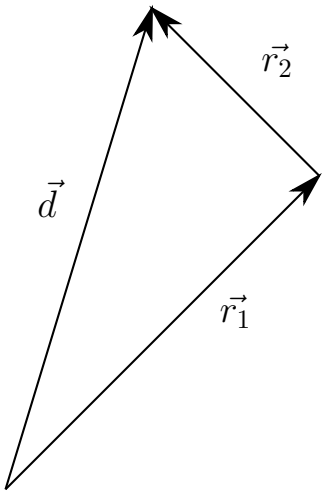


Figura 3.17: Suma de los vectores $\vec{r_1}$ y $\vec{r_2}$ que dan como resultado la dirección del robot.

Para realizar esta suma primero se obtienen las proyecciones de $\vec{d}$ sobre el eje X e Y, dando como resultado $\vec{d_x}$ y $\vec{d_y}$. El ángulo θ se mide entre el $\vec{d}$ y el eje X como muestra la figura 3.18.

$$\begin{aligned} \vec{d_x} &= |\vec{d}| \cos(\theta) \\ \vec{d_y} &= |\vec{d}| \sin(\theta) \end{aligned} \tag{3.5}$$

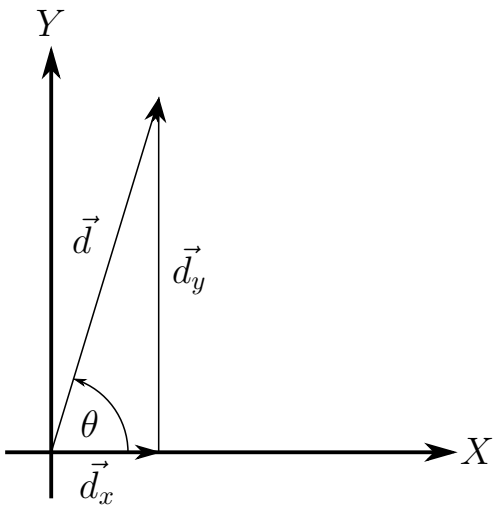


Figura 3.18: Proyección del vector $\vec{d}$ sobre el eje X e Y, dando los vectores $\vec{d_x}$ y $\vec{d_y}$.

Ahora la magnitud de $\vec{r_1}$ y $\vec{r_2}$ se puede definir como:

$$\begin{aligned} \vec{r_1} &= \vec{d_x} \cos(\pi/4) + \vec{d_y} \cos(\pi/4) \\ \vec{r_2} &= -\vec{d_x} \cos(\pi/4) + \vec{d_y} \cos(\pi/4) \end{aligned} \tag{3.6}$$

Donde $\pi/4$ es ángulo de $\vec{r_1}$ y $\vec{r_2}$ sobre el eje X. Si la magnitud de los vectores toma un valor negativo es debe invertir la dirección de las ruedas.

4. Pruebas

Las pruebas que se realizan para comprobar si la red neuronal aprende y es capaz de evadir obstáculos se dividen en dos etapas: La primera consta en mover el robot sobre la pista de pruebas sin colocar obstáculos, sólo con información visual a los costados, la que será captada como flujo óptico, esta permitirá que el robot evite los bordes de la pista y sea capaz de centrarse dentro de esta. En la segunda etapa se mantendrá el mismo esquema de la primera, además de incluir un obstáculo en la pista de pruebas, que el robot tendrá que evadir.

4.1. Comprobación de aprendizaje

Por cada una de las etapas se comprobará de dos formas distintas si la red neuronal está aprendiendo. La primera es evaluando el aprendizaje con un set de datos de prueba, que no fue usado para el entrenamiento, aquí se evalúa cuánto difiere la salida teórica que debería entregar la red, salida esperada, con la que entrega realmente la red. Esta diferencia se considera *test error* (error de prueba) y esta será obtenida por la técnica de *Validation Set Approach* (enfoque de conjunto de validación) la que consiste en dejar parte del conjunto de entrenamiento para validar cuán efectivo fue este [15].

Para la primera etapa se usaron 34755 datos para el entrenamiento y 7367 para la validacion, en la segunda etapa se suman a los datos ya utilizados

[completar cuantos datos de entrenamiento fueron y cuantos se dejaron para validation set approach].

La cuantificación del error se mide de dos formas distintas: La primera es mediante es el error cuadrático medio *MSE* (del ingles: *mean squared error*):

$$MSE = \frac{1}{n}\sum_{x=1}^{n}(y_i - a^L(x_i))^2 \tag{4.1}$$

Donde y_i es la salida esperada y $a^L(x_i)$ es la salida que da la última capa L de la red neuronal con la entrada x_i, con un set de prueba de tamaño n. En el segundo método se mide el porcentaje de efectividad de la red, cuantificando la cantidad de veces que la red entrega una salida que satisface a una respuesta esperada sobre el total de respuestas entregadas. Para saber si la salida cumple esta condición se establece un margen de aceptación de 90 %, donde la salida de la red no puede desviarse más allá de este porcentaje para que se considere salida correcta. El entrenamiento se considera exitoso si la eficacia de la red está sobre el 90 %.

La segunda comprobación de entrenamiento se realizará de forma empírica, comparando el comportamiento de la conducción manual del robot, contra la conducción autónoma guiada por la red neuronal, cada vez que el robot se desplace por la pista de pruebas se deja registro de su desplazamiento, por lo que se puede comparar el recorrido realizado de forma manual y el de forma autónoma. Para considerar la prueba exitosa la desviación de las dos trayectorias no debe ser mayor a [medida de desviación].

4.2. Obtención de datos de prueba

La obtención del conjunto datos, para realizar las pruebas, consta de conducir de forma manual el robot repetidas veces por la pista de pruebas, colocando el robot en posiciones definidas al comienzo de la pista y dirigirlo hasta un punto determinado de esta.

Para la primera etapa, sin importar la posición de inicio, tiene como objetivo mo-

ver el robot al centro de la pista direccionando sobre el eje X mientras se desplaza por la pista en el eje Y, para que de esta forma la red neuronal pueda aprender a direccionar el robot evadiendo los bordes de la pista.

En la segunda etapa se posicionará un obstáculo dentro de la pista, al igual que en la primera etapa, también se deberá centrar el robot dentro de la pista, pero al enfrentarse a un obstáculo este debe ser evadido.

Para la recopilación de datos el recorrido del robot comienza en tres posiciones distintas en el inicio de la pista, las posiciones están a 20 cm del borde derecho, al centro y a 20 cm del borde izquierdo. Por cada una de estas se registran 30 recorridos del robot sobre la pista para los datos de entrenamiento y 10 para los datos de prueba. Obteniendo la información de flujo óptico y el recorrido hecho por este. El recorrido registrado es a una velocidad constante de 60 cm por segundo y una distancia de 2 mt desde el comienzo de la pista.

Para la recopilación de datos con obstáculos, se utilizará el mismo esquema anterior, pero se posiciona un obstáculo dentro de pista en distintas posiciones predefinidas. El obstáculo está construido con tres cilindros de cartón pintados de negro con una franja vertical blanca de 12 mm de ancho (para aumentar el flujo óptico registrado por el robot), cada cilindro tiene un diámetro de 5 cm y 18.5 cm de alto.

El obstáculo es posicionado en tres lugares distintos dentro de la pista, a 25 cm del borde derecho de la pista, al centro de la pista y a 25 cm del borde izquierdo, cada posición está a 1,5 mt del inicio de la pista.

Por cada posición del obstáculo se registran cinco recorridos para los datos de entrenamiento y tres para los da prueba por cada cada posición de inicio, dando un total de quince recorridos para el entrenamiento y nueve para las pruebas por cada posición del obstáculo.

La construcción de la pista de pruebas necesita de un material que proporciona tracción a las ruedas del robot, por lo que se utiliza una alfombra de cubre pisos de 1 metro de ancho y 4 metros de largo, de color *beige* para proporcionar mayor contras-

te frente a los obstáculos. Los bordes de la pista estan construidos de poliestireno expandido de 1 cm de espesor, 25 cm de alto y cubren todo el recorrido de la pista de 4 mt, con un patrón regular líneas verticales blancas y negras de 5 cm de ancho.

5. Resultados

A continuación se muestran los resultados obtenidos al realizar la comprobación del entrenamiento de la red neuronal y las pruebas empíricas de conducción autónoma del robot tanto para la primera etapa, que consta de evitar los bordes de la pista, como para segunda que tiene como objetivo evadir los obstáculos que se le presentan al robot.

5.1. Evasión de bordes

El entrenamiento de la red neuronal en esta etapa se hace con un set de 34755 datos y la validación con un set de 7367 datos. El entrenamiento consta de 30 iteraciones, es decir que la red se entrena por *backpropagation* luego se calcula el *test error* por medio de su efectividad y MSE, luego con red ya entrenada se repite el proceso por un total de 30 veces. El entrenamiento es realizado a dos redes distintas la primera con una sola capa oculta de 50 neuronas (denotada a continuación como Red.1.A), la segunda con 4 capas ocultas de 100, 70, 40, 10 neuronas respectivamente (denotada a continuación como Red.1.B). En la figura 5.1 y 5.2 se muestran los resultados obtenidos con las dos redes neuronales.

Como se puede ver en las figuras no existe mucha diferencia en la efectividad de las dos redes, las dos tiene una efectividad máxima de 0.9370164, en la figura 5.3 se muestra la comparación entre las dos redes. A través de las 30 iteraciones la desviación estándar de la efectividad de Red.1.A es de 0.01846731 y de Red.1.B es 0.01068135.

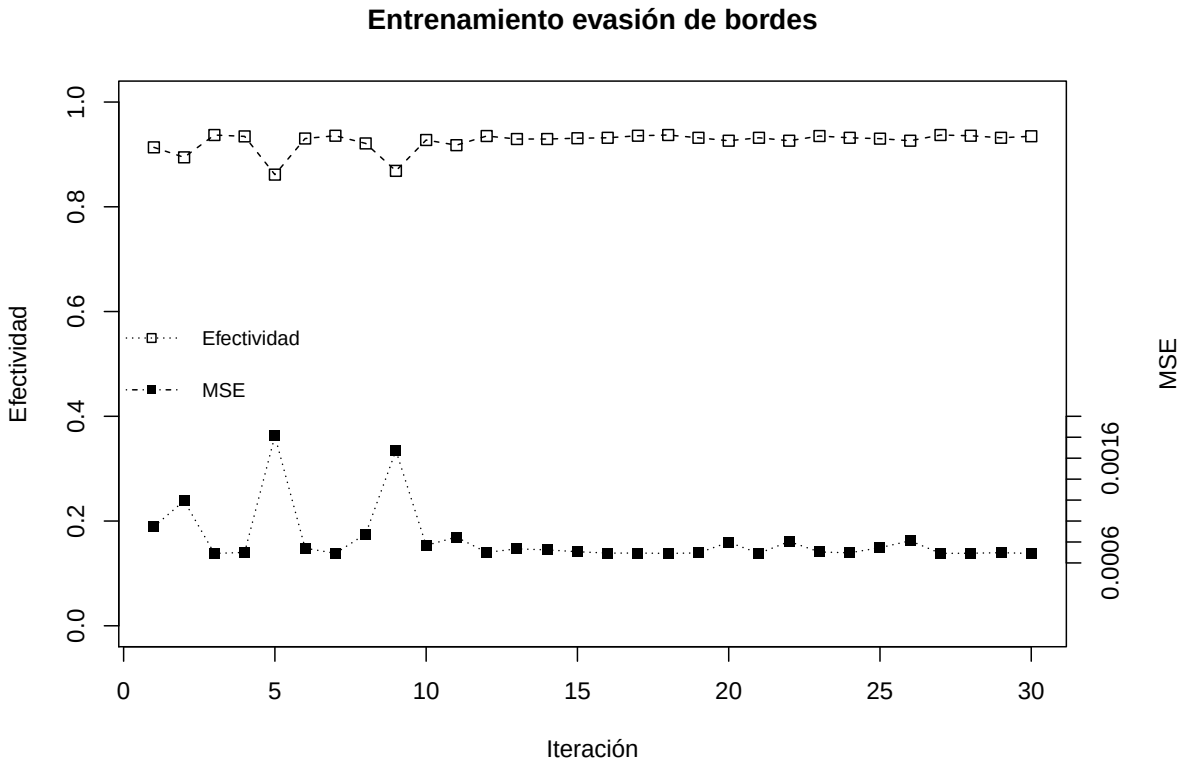


Figura 5.1: Resultado del entrenamiento después de 30 iteraciones de Red.1.A.

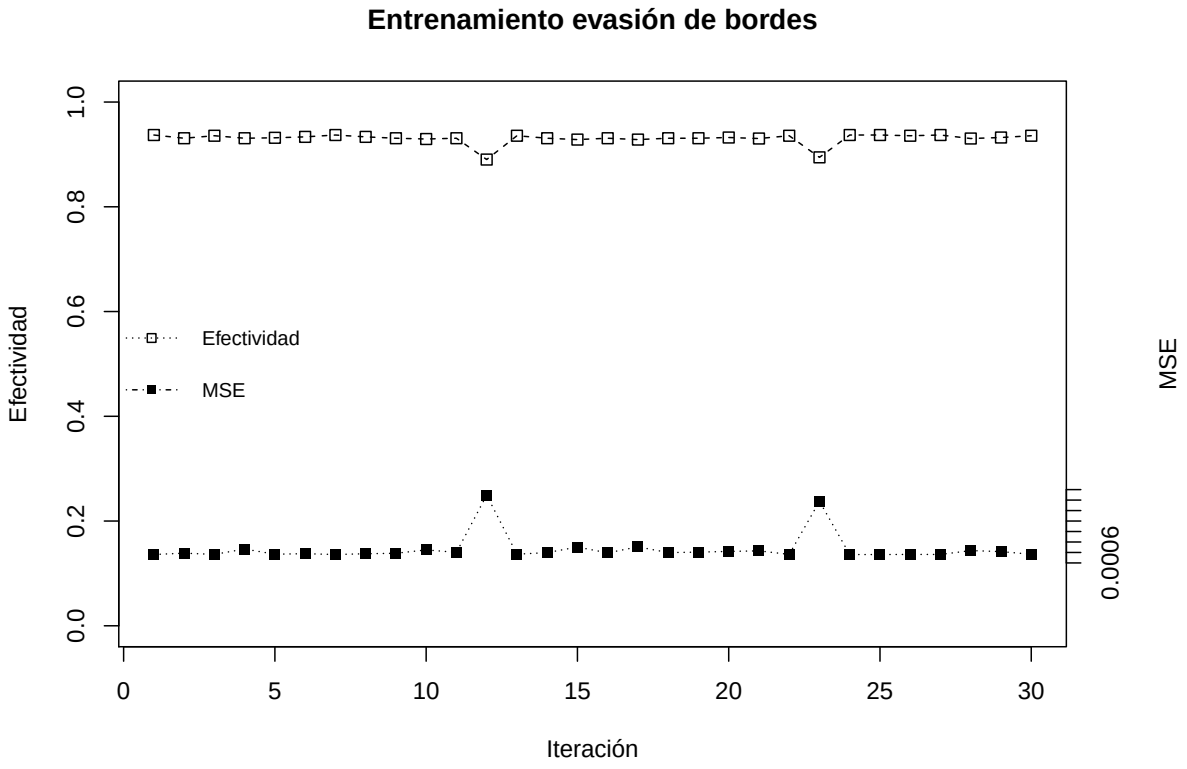


Figura 5.2: Resultado del entrenamiento después de 30 iteraciones de Red.1.B.

Dado que las dos redes tiene un número distinto de neuronas se diferencia en el tiempo que se tarda en computar el entrenamiento, Red.1.A toma 2.7691 minutos en completar las 30 iteraciones y Red.1.B 8.4236 minutos.

El MSE de las redes es bastante bajo, en comparación con un resultado aleatorio que presenta un valor de 0.25, a lo largo del entrenamiento el MSE máximo y mínimo de Red.1.A es de 0.00069 y 0.001812, el de Red.1.B de 0.00067 y 0.00124.

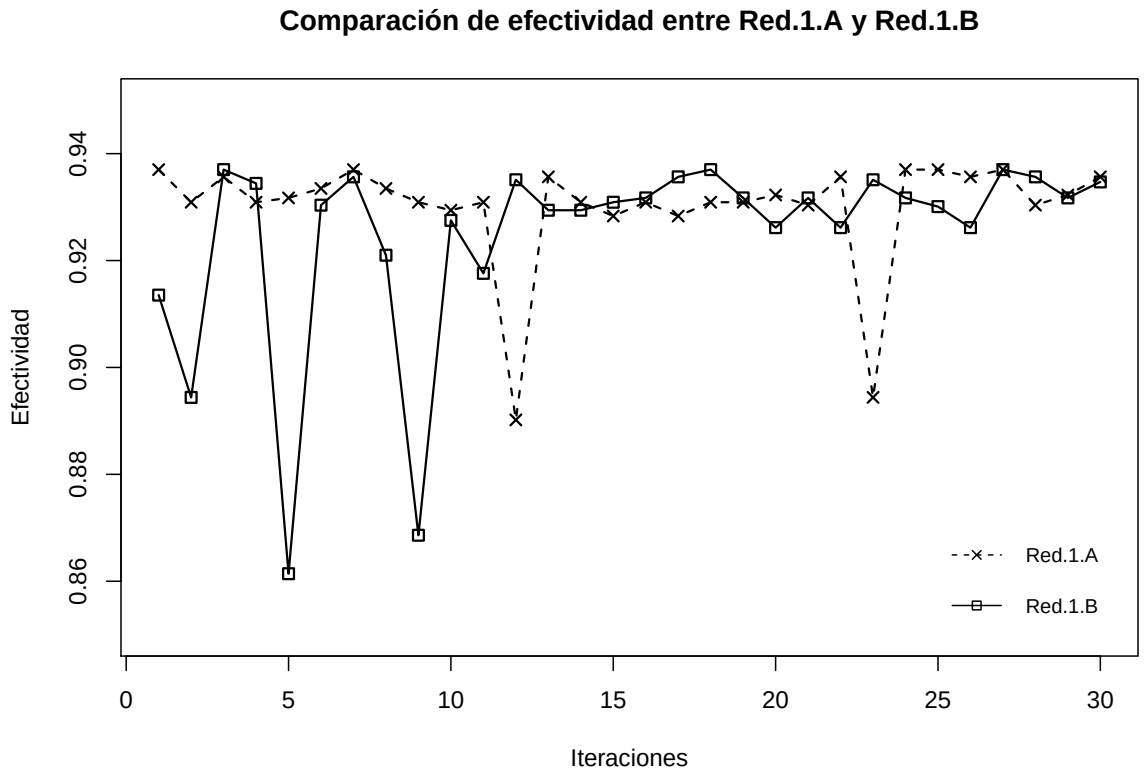


Figura 5.3: Comparación de efectividad entre Red.1.A y Red.1.B.

En la tabla 5.1 muestra una comparativa entre el entrenamiento de las redes neuronales Red.1.A y Red.1.B.

Tabla 5.1: Datos comparativos del entrenamiento en Red.1.A y Red.1.B

	Entrenamiento Red.1.A y Red.1.B				
Red	**Efectividad maxima**	**Desviasion estandar**	**MSE minimo**	**MSE maximo**	**Tiempo de entrenamiento**
Red.1.A	0.9370164	0.01846731	0.0006908158	0.0018120270	2.76913118362
Red.1.B	0.9370164	0.01068135	0.0006799585	0.0012468020	8.42360328436

5.2. Evasión de obstaculos

En esta etapa se utilizan los mismos datos que en la evasión de bordes, pero a estos se le suman 31542 elementos al set de entrenamiento y 13268 al de validación, obtenidos de los recorridos realizados por la conducción del robot evadiendo obstáculos. Con estos nuevos conjuntos de datos que tienen un total de 66297 elementos para el entrenamiento y 20635 para la validación, se vuelve a hacer el proceso de entrenamiento, iterar 30 veces para ambas redes neuronales (denotadas como Red.2.A y Red.2.B) y al término de cada entrenamiento hacer la comprobación con los datos de validación. La figura 5.4 y 5.5 muestra los resultados del proceso.

Al igual que en la primera etapa no se muestra mucha diferencia entre la efectividad de las redes neuronales Red.2.A y Red.2.B, pero esta vez las red que cuenta con más neuronas, Red.2.B, logró una efectividad de 0.932348 mientras que Red.2.A 0.9311849, la figura 5.6 muestra una comparativa entre las dos. La desviación estándar a lo largo del entrenamiento fue de 0.01630602 y 0.001632309, para la Red.2.A y Red.2.B respectivamente. Además ambas redes disminuyeron su resultados en comparación con el primera etapa, pero destaca la Red.2.B por tener una disminución notoria frente a las demás redes.

Debido al aumento tanto del set de datos de entrenamiento como el de validación los tiempos para completar el entrenamiento también aumentaron siendo de 5.5678 minutos para Red.2.A y 17.6041 para Red.2.B. Nuevamente debido al número de neuronas el tiempo de entrenamiento es mayor para la segunda red.

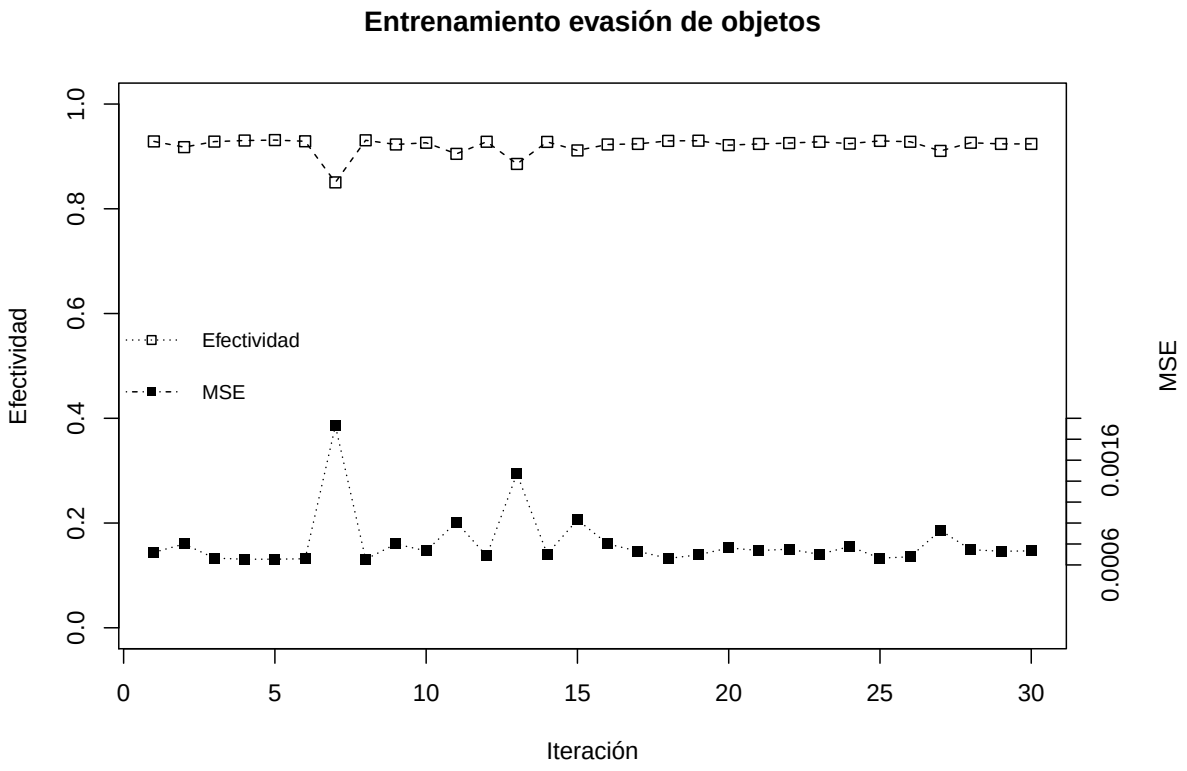


Figura 5.4: Resultado del entrenamiento después de 30 iteraciones de Red.2.A.

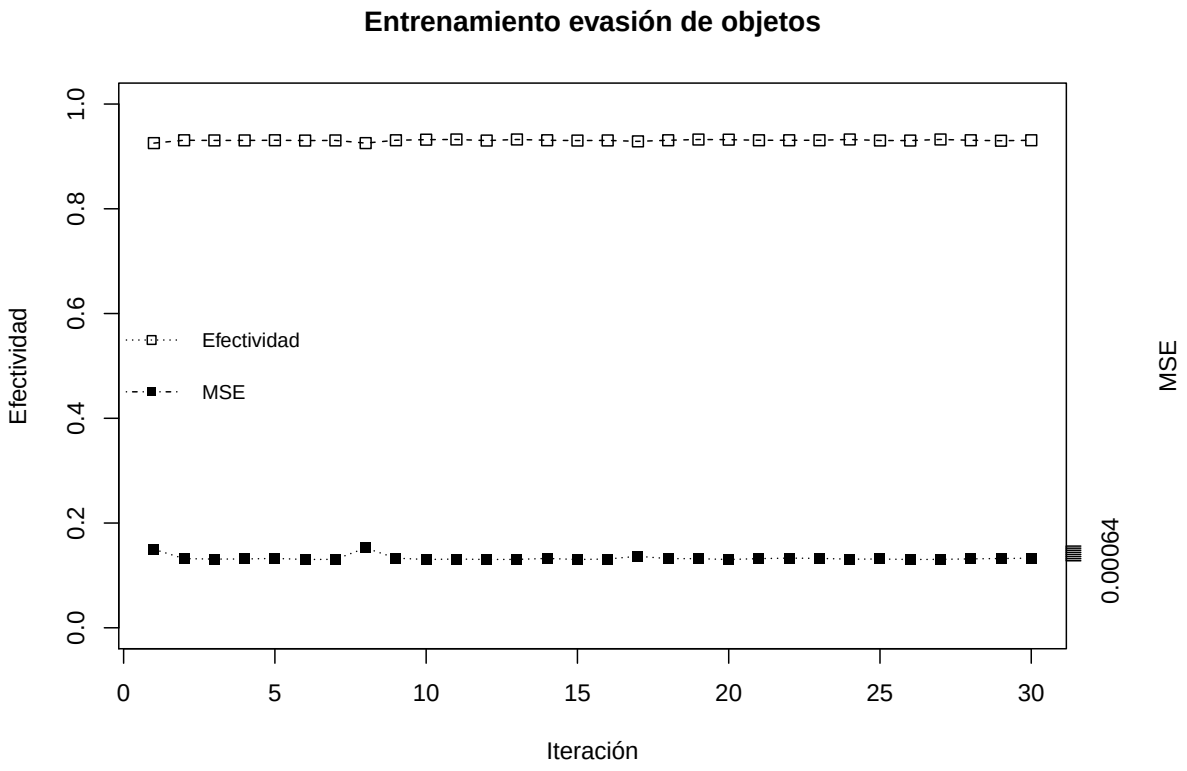


Figura 5.5: Resultado del entrenamiento después de 30 iteraciones de Red.2.b.

El MSE en esta etapa también se mantuvo con valores bajos, pero al igual que la desviación estándar de la Red.2.B el rango sufrió una reducción notoria, siendo de 0.0006519474 y 0.0007619559 mientras que para la Red.2.A el mínimo y máximo fue de 0.000653724 y 0.001927609 respectivamente.

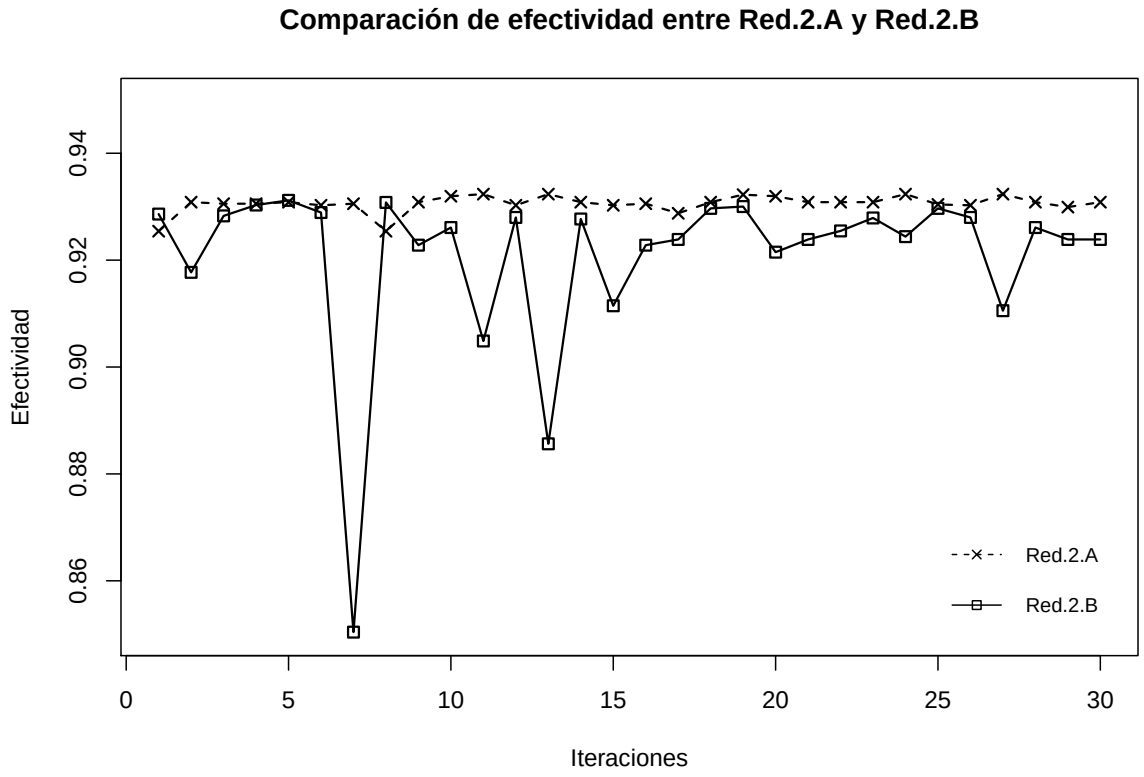


Figura 5.6: Comparación de efectividad entre Red.2.A y Red.2.B.

La tabla 5.2 muestra la comparativa entre la Red.2.A y Red.2.B de los resultados obtenidos en la segunda etapa de entrenamiento.

Tabla 5.2: Datos comparativos del entrenamiento en Red.2.A y Red.2.B

	Entrenamiento Red.2.A y Red.2.B				
Red	**Efectividad maxima**	**Desviasion estandar**	**MSE minimo**	**MSE maximo**	**Tiempo de entrenamiento**
Red.2.A	0.9311849	0.01630602	0.000653724	0.001927609	5.56788144906
Red.2.B	0.932348	0.001632309	0.0006519474	0.0007619559	17.604199481

En el anexo A se muestran los resultados completos del entrenamiento realizado a las distintas redes neuronales, se listan las iteraciones y en cada una la efectividad y MSE conseguidos.

Glosario

Ángulo de ataque:

Caja reductora:

Circuito en serie:

Filtro gaussiano:

Grado de libertad:

Holonómico:

Impresión 3D:

LED:

Modelado Paramétrico:

Motor DC:

Open source:

Overflow:

Paso de hélice:

Puente h:

Robot:

Test Error:

Validation Set Approach:

Voltaje nominal:

Bibliografía

[1] Opencv: Optical flow, 2017.

[2] Clay Allen. Quadcopters, 2014.

[3] Arduino. Arduino reference analogwrite, 2017.

[4] Arduino. What is arduino?, 2017.

[5] Atemel Corporation. *ATmega48A/PA/88A/PA/168A/PA/328/P*, 11 2015.

[6] Dana Ballard. *Computer vision*. Prentice-Hall, Englewood Cliffs, N.J, 1982.

[7] Steven Barrett. *Microcontrollers fundamentals for engineers and scientists*. Morgan & Claypool Publishers, San Rafael, Calif, 2006.

[8] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. .ºReilly Media, Inc.", 2008.

[9] Myriam Cayre, Jordane Malaterre, Sophie Scotto-Lomassese, Colette Strambi, and Alain Strambi. The common properties of neurogenesis in the adult brain: from invertebrates to vertebrates. *Comparative Biochemistry and Physiology Part B: Biochemistry and Molecular Biology*, 132(1):1–15, 2002.

[10] Adam Chapweske. Ps/2 mouse interfacing, 2003.

[11] Adam Chapweske. The ps/2 mouse/keyboard protocol, 2003.

[12] Sanja Filder. Image pyramid.

[13] David Forsyth. *Computer vision : a modern approach*. Prentice Hall, Upper Saddle River, N.J. London, 2003.

[14] Berthold Horn. *Robot vision*. MIT press, 1986.

[15] Gareth James, Daniela Witten, and Trevor Hastie. An introduction to statistical learning: With applications in r., 2014.

[16] Munasinghe Muhandiram Premarathna Jayakody, Munagamage and Rajakaruna. Design and development of an omni-directional wheeled mobile robot. *International Research Symposium on Engineering Advancements 2015 (RSEA 2015)*, 2015.

[17] Reinhard Klette. *Concise computer vision : an introduction into theory and algorithms*. Springer, London, 2014.

[18] Tim Morris. *Computer vision and image processing*. Palgrave Macmillan, Basingstoke, 2004.

[19] Michael Nielsen. Neural networks and deep learning, 2017.

[20] Michael Nielsen. Using neural nets to recognize handwritten digits, 2017.

[21] International Civil Aviation Organization. Unmanned aircraft systems (uas), 2011.

[22] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. pages 305–313, 1989.

[23] Linda Shapiro. *Computer vision*. Prentice Hall, Upper Saddle River, NJ, 2001.

[24] Charles Siskind. *Electrical control systems in industry*. McGraw-Hill, New York, 1963.

[25] Milan Sonka. *Image processing, analysis, and machine vision*. Thompson Learning, Toronto, 2008.

[26] STMicroelectronics. *L298*, 2000.

[27] C. E. Vandoni. *Proceedings : 1996 CERN School of Computing : Egmond aan Zee, the Netherlands, 8 September-21 September 1996*. CERN, Geneva, 1996.

[28] Al Williams. *Microcontroller projects using the Basic Stamp*. CMP Books, Lawrence, Kan, 2002.

ANEXOS

A. Resultados entrenamiento de redes neuronales

A.1. Red.1.A

Tabla A.1: Datos del entrenamiento realizado a la Red.1.A.

Entrenamiento Red.1.A		
Iteración	**Efectividad**	**MSE**
1	0.913533324284	0.000945908778483
2	0.894393918827	0.00119303279299
3	0.937016424596	0.00069081575127
4	0.934437355776	0.00070042173591
5	0.861408986019	0.00181202704469
6	0.930365141849	0.000741045642054
7	0.935659019954	0.000691926551444
8	0.920999049817	0.000876352966901
9	0.868603230623	0.00167283443075
10	0.9275145921	0.000767064584306
11	0.917605538211	0.000846716155162
12	0.935116058097	0.000700496521621
13	0.929414958599	0.000734796869545
14	0.929414958599	0.000724355116221
15	0.930908103706	0.000707206147845

16	0.931722546491	0.000693620418216
17	0.935659019954	0.000694472906729
18	0.937016424596	0.000690996410918
19	0.931722546491	0.000695499916258
20	0.926157187458	0.000797349211209
21	0.931722546491	0.000691992846241
22	0.926157187458	0.000804536187667
23	0.935116058097	0.000704533339926
24	0.931722546491	0.000696808227387
25	0.93009366092	0.000745401931822
26	0.926157187458	0.000811712662915
27	0.937016424596	0.000691168580695
28	0.935659019954	0.000692704302926
29	0.931722546491	0.000697733637806
30	0.934708836704	0.000691069343532

A.2. Red.1.B

Tabla A.2: Datos del entrenamiento realizado a la Red.1.B.

Entrenamiento Red.1.B		
Iteración	**Efectividad**	**MSE**
1	0.937016424596	0.000680479158646
2	0.930908103706	0.000693363879534
3	0.935659019954	0.000681495122213
4	0.930908103706	0.000732845982048
5	0.931722546491	0.000683404847922
6	0.933487172526	0.000688381137309
7	0.937016424596	0.000680450324945
8	0.933487172526	0.000687662533276
9	0.930908103706	0.000692330783035
10	0.929414958599	0.000724415835138
11	0.930908103706	0.000701182301275

12	0.890185964436	0.00124680202199
13	0.935659019954	0.000683446893253
14	0.930908103706	0.000698575037316
15	0.928329034885	0.000751828245677
16	0.930908103706	0.000696074099689
17	0.928329034885	0.000753517327056
18	0.930908103706	0.000698791474565
19	0.930908103706	0.000703277030309
20	0.932265508348	0.00070997250449
21	0.930365141849	0.000715367567923
22	0.935659019954	0.000681680637419
23	0.894393918827	0.00118822859392
24	0.937016424596	0.000679987713934
25	0.937016424596	0.000679958482945
26	0.935659019954	0.000681375297971
27	0.937016424596	0.000680985065225
28	0.930365141849	0.000717947488892
29	0.932265508348	0.000708196541166
30	0.935659019954	0.00068277419994

A.3. Red.1.B

Tabla A.3: Datos del entrenamiento realizado a la Red.2.A.

Entrenamiento Red.2.A		
Iteración	**Efectividad**	**MSE**
1	0.928616428398	0.000719893721006
2	0.917712624182	0.000800126689279
3	0.928277198934	0.000664979822336
4	0.930312575721	0.000654552914366
5	0.931184880058	0.000654863604206
6	0.928907196511	0.000658793172797
7	0.850399806155	0.00192760902722

8	0.930797189242	0.000653724038728
9	0.92280106615	0.000800747839431
10	0.926096438091	0.000733538849396
11	0.904870365883	0.0010089428019
12	0.927986430821	0.000685578890759
13	0.885631209111	0.00147211690003
14	0.927695662709	0.000703759297938
15	0.911461109765	0.00103087829926
16	0.92280106615	0.000801065965672
17	0.923867215895	0.000730485527153
18	0.929682578144	0.000662293709483
19	0.930021807608	0.000696563117277
20	0.921492609644	0.000762692770483
21	0.923867215895	0.000737694338211
22	0.925466440514	0.000750068796091
23	0.927889508117	0.000701192146117
24	0.924400290768	0.000776208040826
25	0.929682578144	0.00066460162863
26	0.927986430821	0.000678062613667
27	0.910540344076	0.000930106000753
28	0.926096438091	0.000745957305927
29	0.923867215895	0.000730502893394
30	0.923867215895	0.000734406848618

Tabla A.4: Datos del entrenamiento realizado a la Red.2.B.

Entrenamiento Red.2.B		
Iteración	**Efectividad**	**MSE**
1	0.925417979162	0.000749432750139
2	0.930845650594	0.000660165282152
3	0.930554882481	0.000655959851093
4	0.930554882481	0.000656857864731
5	0.930845650594	0.000662295925217

6	0.930264114369	0.000652311348451
7	0.930554882481	0.000653654554964
8	0.925417979162	0.000761955863816
9	0.930845650594	0.000664555387331
10	0.931960261691	0.000652296703772
11	0.932347952508	0.000656277069034
12	0.930264114369	0.000652385361621
13	0.932347952508	0.000652944033472
14	0.930845650594	0.000661843223622
15	0.930264114369	0.000652187111804
16	0.930554882481	0.000656488882683
17	0.928761812455	0.000683020599665
18	0.930845650594	0.00066003563414
19	0.932251029804	0.00065971120924
20	0.931960261691	0.000651947403756
21	0.930845650594	0.000660280012452
22	0.930845650594	0.000665047738241
23	0.930845650594	0.000662555843386
24	0.932347952508	0.000654873962048
25	0.930409498425	0.00065689395976
26	0.930264114369	0.000652052591494
27	0.932347952508	0.0006534034983
28	0.930845650594	0.000657837034339
29	0.929924884904	0.000661217162169
30	0.930845650594	0.000664652371238

B. Arduino UNO rev 2

Espesificaciones de la placa *Arduino* UNO rev 2.

Microcontrolador	ATmega328P
Voltaje de operación	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (limite)	6-20V
Pin digitales	14 (6 proporcionan salida PWM)
Pin digitales PWM	6
Pin de entrada analogica	6
Corriente por pin	20 mA
Corriente por pin 3.3V	50 mA
Memoria Flash	32 KBytes de los cuales 0.5 KBytes son usados por el bootloader
SRAM	2 KBytes
EEPROM	1 KBytes
Velocidad de reloj	16 MHz
LED_BUILTIN	13
Largo	68.6 mm
Ancho	53.4 mm
Peso	25 g

Tabla B.1: Especificaciones Arduino UNO.

C. Configuración impresora 3D

Configuración completa de la impresora 3D al momento de construir las piezas para el robot omnidireccional.

```
# generated by Slic3r 1.2.9 on Tue Jun 20 23:36:24 2017
avoid_crossing_perimeters = 0
bed_shape = 0x0,200x0,200x145,0x145
bed_temperature = 60
before_layer_gcode =
bottom_solid_layers = 3
bridge_acceleration = 0
bridge_fan_speed = 100
bridge_flow_ratio = 1
bridge_speed = 30
brim_width = 0
complete_objects = 0
cooling = 1
default_acceleration = 0
disable_fan_first_layers = 3
dont_support_bridges = 1
duplicate_distance = 6
end_gcode = M104 S0 ; turn off temperature\nG28 X0
; home X axis\nG0 Y149; \nM84     ; disable motors\n
external_fill_pattern = rectilinear
external_perimeter_extrusion_width = 0
```

```
external_perimeter_speed = 80%
external_perimeters_first = 0
extra_perimeters = 1
extruder_clearance_height = 20
extruder_clearance_radius = 20
extruder_offset = 0x0
extrusion_axis = E
extrusion_multiplier = 1
extrusion_width = 0
fan_always_on = 1
fan_below_layer_time = 0
filament_colour = #000000
filament_diameter = 1.75
fill_angle = 45
fill_density = 15%
fill_pattern = rectilinear
first_layer_acceleration = 0
first_layer_bed_temperature = 60
first_layer_extrusion_width = 250%
first_layer_height = 0.4
first_layer_speed = 50%
first_layer_temperature = 215
gap_fill_speed = 15
gcode_arcs = 0
gcode_comments = 0
gcode_flavor = reprap
infill_acceleration = 0
infill_every_layers = 1
infill_extruder = 1
infill_extrusion_width = 0
infill_first = 0
infill_only_where_needed = 0
infill_overlap = 15%
infill_speed = 40
```

```
interface_shells = 0
layer_gcode =
layer_height = 0.2
max_fan_speed = 100
max_print_speed = 80
max_volumetric_speed = 0
min_fan_speed = 70
min_print_speed = 10
min_skirt_length = 0
notes =
nozzle_diameter = 0.4
octoprint_apikey =
octoprint_host =
only_retract_when_crossing_perimeters = 1
ooze_prevention = 0
output_filename_format = [input_filename_base].gcode
overhangs = 1
perimeter_acceleration = 0
perimeter_extruder = 1
perimeter_extrusion_width = 0
perimeter_speed = 30
perimeters = 3
post_process =
pressure_advance = 0
raft_layers = 0
resolution = 0
retract_before_travel = 2
retract_layer_change = 1
retract_length = 5
retract_length_toolchange = 10
retract_lift = 0
retract_restart_extra = 0
retract_restart_extra_toolchange = 0
retract_speed = 40
```

```
seam_position = aligned
skirt_distance = 6
skirt_height = 1
skirts = 2
slowdown_below_layer_time = 20
small_perimeter_speed = 15
solid_infill_below_area = 70
solid_infill_every_layers = 0
solid_infill_extruder = 1
solid_infill_extrusion_width = 0
solid_infill_speed = 80%
spiral_vase = 0
standby_temperature_delta = -5
start_gcode = G28 ; home all axes\n;G1 Z5 F5000 ; lift nozzle\n
support_material = 0
support_material_angle = 0
support_material_contact_distance = 0.2
support_material_enforce_layers = 0
support_material_extruder = 1
support_material_extrusion_width = 0
support_material_interface_extruder = 1
support_material_interface_layers = 3
support_material_interface_spacing = 0
support_material_interface_speed = 100%
support_material_pattern = pillars
support_material_spacing = 2.5
support_material_speed = 40
support_material_threshold = 0
temperature = 210
thin_walls = 1
threads = 2
toolchange_gcode =
top_infill_extrusion_width = 0
top_solid_infill_speed = 50%
```

```
top_solid_layers = 3
travel_speed = 60
use_firmware_retraction = 0
use_relative_e_distances = 0
use_volumetric_e = 0
vibration_limit = 0
wipe = 0
xy_size_compensation = 0
z_offset = 0
```

C.1. La primera sección del primer anexo

Aquí va el texto de la primera sección del primer anexo...

C.2. La segunda sección del primer anexo

Aquí va el texto de la segunda sección del primer anexo...

C.2.1. La primera subsección de la segunda sección del primer anexo

D. El segundo Anexo

Aquí va el texto del segundo anexo...

D.1. La primera sección del segundo anexo

Aquí va el texto de la primera sección del segundo anexo...