UNIVERSIDAD DE PIURA

FACULTAD DE INGENIERÍA

Prototipo de robot cuadrúpedo basado en inteligencia artificial con aprendizaje profundo por refuerzo y ROS, para realizar actividades de búsqueda y rescate en la región de Piura

Tesis para optar el Título de
Ingeniero Mecánico - Eléctrico

Jesús Joel Izquierdo Arcaya

Asesor:
Mgtr. Ing. Juan Carlos Soto Bohórquez

Piura, diciembre de 2024

UNIVERSIDAD DE PIURA

Declaración Jurada de Originalidad del Trabajo Final

Yo, Jesús Joel Izquierdo Arcaya, egresado del Programa Académico de Ingeniería Mecánico-Eléctrica de la Facultad de Ingeniería de la Universidad de Piura, identificado(a) con DNI: 74947807, declaro que:

Soy autor del trabajo final titulado:
"**Prototipo de robot cuadrúpedo basado en inteligencia artificial con aprendizaje profundo por refuerzo y ROS, para realizar actividades de búsqueda y rescate en la región de Piura.**"
El mismo que presento bajo la modalidad de Tesis para optar el Título profesional de Ingeniero Mecánico-Eléctrico.

El texto de mi trabajo final es original y no vulnera los derechos de terceros o, de ser el caso, derechos de los coautores, incluidos los derechos de propiedad intelectual, datos personales, entre otros. En tal sentido, el texto de mi trabajo final no ha sido plagiado total ni parcialmente, para lo cual, he respetado las normas internacionales de citas y referencias de las fuentes consultadas. Asimismo, el texto del trabajo final que presento no ha sido publicado ni presentado antes en cualquier medio electrónico o físico; y que la investigación, los resultados, datos, conclusiones y demás información presentada que atribuyo a mi autoría son veraces.
En caso de detectarse el incumplimiento de lo declarado asumo frente a terceros, la Universidad de Piura y/o la Administración Pública toda responsabilidad que pueda derivarse por el trabajo final presentado. Lo señalado incluye responsabilidad pecuniaria incluido el pago de multas u otros por los daños y perjuicios que se ocasionen.

La asesoría del trabajo estuvo a cargo de los siguientes docentes de la Universidad de Piura:

- Mgtr. Ing. Juan Carlos Soto Bohórquez, identificado con DNI: 40107278

Declaro (declaramos) que:
Luego de haber empleado el software de coincidencia Turnitin, revisado las fuentes de información señaladas por el autor, y en razón de mi (nuestra) experiencia como investigador(es), declaro (declaramos) que las ideas expuestas en el trabajo final alcanzan las condiciones de calidad, integridad y originalidad acorde a los objetivos institucionales y estándares en materia de investigación. Finalmente, no asumo (asumimos) responsabilidad por la posible vulneración de derechos de autor en el trabajo final referido, pues tal responsabilidad es exclusiva del autor.

Fecha: 04/12/2024.

.....
Firma del asesor[1]

.....
Firma del autor[1]

.....
Firma del co-asesor[1]

.....
Firma del co-asesor[1]

[1] Firma idéntica al DNI. No se admite digital, salvo certificado.

Dedicatoria:

A Dios por estar presente conmigo
A mis amados padres

Armando y María Vicenta

A mi hermana Julexy

Y a las personas que han estado

presente por su constante apoyo.

Agradecimientos

Agradezco en primer lugar a Dios por guiarme en el camino profesional y brindarme salud a mí y a mis seres queridos. Aplicar mis conocimientos profesionales para poder ayudar a las personas que lo necesiten.

A mis padres por estar constantemente apoyándome, a mi hermana por ser mi fuente de inspiración para seguir siendo un profesional y siempre estar apuntando hacia lo más alto. A mis amistades, por estar presente en cada uno de los logros que he ido obteniendo en el transcurso de esta investigación. A mi asesor por seguir perseverante en el desarrollo de loa investigación y brindarme todo el apoyo necesario.

Resumen

El presente trabajo de investigación tiene como objetivo principal implementar un prototipo de robot cuadrúpedo basado en inteligencia artificial con aprendizaje profundo por refuerzo y ROS, para realizar actividades de búsqueda y rescate en la región de Piura. Para cumplirlo se ha optado por el diseño y ensamblaje de un robot cuadrúpedo, con doce grados de libertad, con el fin de poder realizar un control a través de un microprocesador Raspberry Pi 4 y con una placa de extensión Navio2. La eficiencia de este proyecto se verá reflejado en la capacidad que tenga el agente "el prototipo" para lograr obtener un valor de recompensa alto. Se diseñará diferentes entornos para el agente, en donde para cada entorno la recompensa será diferente por lo que se podrá comparar con el dos modelos de aprendizaje como es el *Deep Reforcement Learning* y la Qtable, siendo esta última un modelo sencillo el cual sirva como partida para tener que comprar los resultados con el primer modelo. Todo esto bajo el apoyo de un algoritmo llamado ROS el cual permite trabajar con diferentes nodos y entrelazar comunicación entre ellos.

Tabla de contenido

Lista de tablas

Lista de figuras

Introducción

En el presente trabajo de investigación de tesis, denominado "Prototipo de robot cuadrúpedo basado en inteligencia artificial con aprendizaje profundo por refuerzo y ROS, para realizar actividades de búsqueda y rescate en la región de Piura", pone en práctica uno de los conceptos revolucionados en el ámbito de la inteligencia artificial. Se implementa los conceptos de aprendizaje profundo por refuerzo profundo de un agente (quadrobot) y el Sistema Operativo Robótico (ROS). El sistema ROS cuenta con bibliotecas de software y herramientas para aplicaciones de robots y es de código abierto.

Este proyecto consta de cinco capítulos. En el primer capítulo, "Inteligencia Artificial", describe los conceptos de inteligencia artificial, tipos de aprendizaje supervisado, no supervisado y por refuerzo; así como las ecuaciones fundamentales que los abordan. Este capítulo describe principalmente el aprendizaje por refuerzo, por ser el modelo que se desarrolla en esta investigación.

En el segundo capítulo, "Robot Operating System (ROS)", se muestra la instalación y configuración de las bibliotecas en un dispositivo Raspberry Pi 4, los diferentes conceptos del sistema ROS, así como también el diseño de nodos y tópicos de comunicación como Publisher y Subscriber.

En el tercer capítulo, "Ensamblaje e instalación de componentes", se muestra los componentes principales que se han requerido, desde la impresión de las piezas, hasta los dispositivos electrónicos que se han tomado en cuenta con una breve descripción de su uso en el prototipo de forma general. Se describe las partes del robot, su construcción, ensamblaje y programación.

En el cuarto capítulo, "Desarrollo del entorno y el modelo de entrenamiento", se diseña el entorno creado para que el agente (quadrobot) pueda interactuar en él. En este capítulo, se ven los conceptos de los ángulos de navegación obtenidos del sensor inercial IMU, se describe la cinemática inversa para el cambio de posición en los servomotores; así como también la configuración y topologías referentes al modelo Qtable y el DQN con la estructura de la red neuronal.

En el quinto capítulo, "Análisis de resultados de los modelos", se muestran los resultados obtenidos por los modelos diseñados, se hace una comparación entre modelos y propuestas de mejoras. Posteriormente, se dan las conclusiones obtenidas a través de la experimentación y el cumplimiento de los objetivos planteados en esta investigación.

En la sección de Apéndices se incluyen los programas y planos que se han utilizado en el proyecto de tesis.

El sistema de control basado en *Machine Learning* y *Deep Learning* junto con ROS sirve de base para futuras investigaciones.

Capítulo 1
Inteligencia Artificial

1.1 Introducción a la Inteligencia Artificial

La inteligencia artificial (IA) tiene como objetivo imitar algunas de las actividades que la mente humana puede llevar a cabo. Se desarrolla principalmente en áreas como la percepción, asociación, predicción, planificación y control motriz.

La IA, tal como lo describe Boden en su libro "Inteligencia Artificial" (Boden, 2017), plantea tener dos usos principales. El primero es tecnológico, es decir, utilizar ordenadores con el fin de realizar actividades como control de vehículos autónomos, realidad aumentada o realidad virtual. El otro es científico, basándose en modelos de inteligencia artificial para tratar de resolver dilemas de los seres humanos y otros seres vivos, así como para la toma de decisiones en salud y transporte.

Se clasifica en representaciones simbólicas y sub-simbolicas; las simbólicas consiste en un número finito de reglas que se utiliza para la predicción y que forman parte del sistema, mientras que las sub-simbolicas se caracterizan por crear sistemas con capacidad de aprendizaje (Julio Cesar Ponce Gallegos et al., 2014).

Asimismo, bajo estos conceptos y con los avances en informática para almacenar grandes cantidades de información, han surgido la idea de "aprendizaje", que se caracteriza por la optimización de parámetros de un modelo mediante el uso datos previos o datos de entrenamiento.

Actualmente, en el campo del aprendizaje abarca una amplia gama de enfoques y técnicas, cada uno con sus propias características y aplicaciones distintivas. Entre los principales tipos de aprendizaje se encuentran tres características fundamentales: el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo.

1.2 Aprendizaje supervisado

El aprendizaje supervisado tiene como finalidad la predicción. Se espera que a partir de un dato X, resulte una predicción $\hat{Y}$. Este tipo de aprendizaje es especialmente relevante en clasificación de datos, donde se utilizan conjuntos de datos de entrada previamente etiquetados. Luego, se procede a entrenar un modelo con la expectativa de que aprenda a identificar y clasificar correctamente estos datos, y sea con la misma base de datos o una nueva.

En el aprendizaje supervisado, se emplea un conjunto de entrenamiento para instruir a los modelos sobre como generar el resultado deseado. Esto se logra al presentar ejemplos de entradas junto con sus salidas esperadas correspondientes, lo que permite al modelo aprender y mejorar con los datos. Durante el proceso de entramiento, el algoritmo evalúa su precisión utilizando una función de pérdida, ajustándose continuamente hasta que el error se haya minimizado lo suficiente para satisfacer los criterios de rendimiento establecidos.

En la extracción de datos, la inteligencia artificial por aprendizaje supervisado se enfrentan dos problemas principales; clasificación y regresión. Estos problemas pueden ser abordados por el aprendizaje supervisado de la siguiente manera:

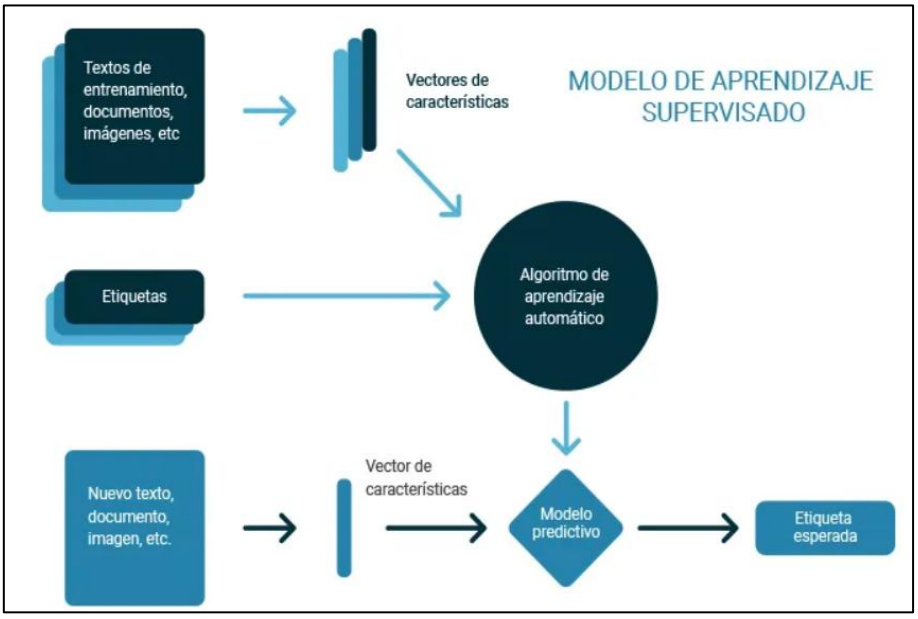
En la clasificación, el aprendizaje supervisado utiliza algoritmos para asignar con precisión los datos de prueba en categorías específicas, reconocer entidades especificas dentro del conjunto de datos e intenta extraer conclusiones sobre cómo deben ser etiquetados.

Mientras que en la regresión se usa para entender las variables dependientes e independientes, haciendo uso de proyecciones, regresiones lineales, regresiones logísticas o regresiones polinómicas, siendo esta última la más utilizada.

En la siguiente figura se observa un diagrama de flujo donde expresa el funcionamiento de la inteligencia artificial por aprendizaje supervisado.

Figura 1

Diagrama de flujo de aprendizaje supervisado



Nota: tomado de (Gonzalez, 2020)

Los modelos de aprendizaje supervisados se pueden utilizar para crear y promover una serie de aplicaciones empresariales, incluidas las siguientes:

- Reconocimiento de imágenes y objetos: los modelos basados en el algoritmo de aprendizaje supervisado se pueden utilizar para localizar, categorizar objetos y aislar a partir de cualquier medio audiovisual, siendo estos útiles para aplicaciones tales como visión artificial y análisis de imágenes.

- Análisis predictivo: un caso de uso generalizado para modelos de aprendizaje supervisado consiste en crear sistemas de análisis predictivos para proporcionar información detallada sobre diversos puntos de datos empresariales.

• Detección de Spam: Se emplea un algoritmo de clasificación supervisado; las organizaciones pueden entrenar bases de datos para identificar patrones o anomalías en nuevos datos para organizar de manera efectiva el spam (IBM, 2024).

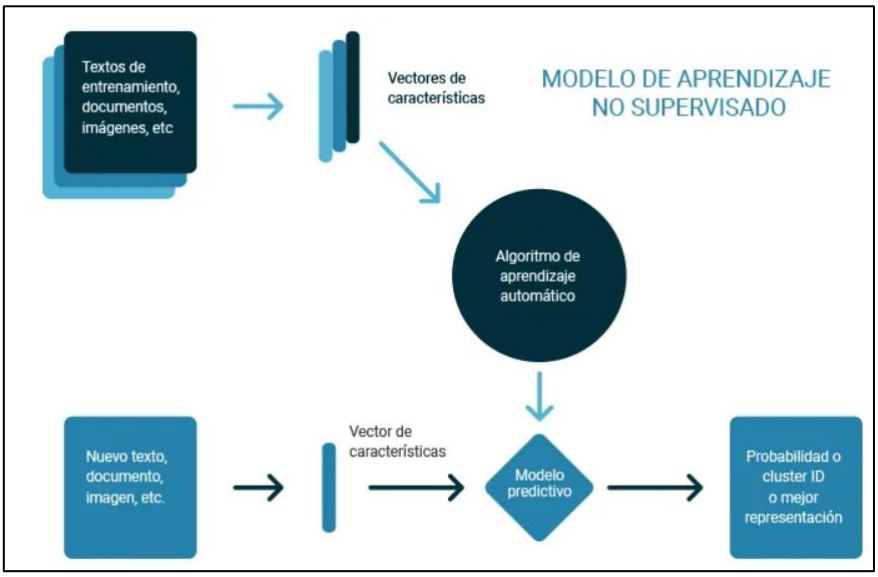
1.3 Aprendizaje no supervisado

En el aprendizaje no supervisado no se realiza un etiquetado previo de los datos, por lo que el modelo debe "aprender" a clasificarlos, buscará una forma de agrupar los datos que comparten similitudes. Este tipo de aprendizaje no requiere de intervención humana (LASSE ROUHIAINEN, 2018).

El aprendizaje no supervisado es un enfoque crucial en el campo del aprendizaje automático. Se basan en algoritmos como la agrupación en clústeres, asociación y reducción de dimensionalidad para extraer patrones y estructuras ocultas en conjuntos de datos sin etiquetar. Esta capacidad de descubrir similitudes y diferencias en la información lo convierten en la solución ideal para el análisis de datos exploratorios, la segmentación de clientes, reconocimiento de imágenes y la estrategia de venta cruzada.

Figura 2

Diagrama de flujo de aprendizaje no supervisado



Nota: tomado de (Gonzalez, 2020)

La técnica de agrupación en clústeres se usa en la minería de datos que agrupa datos no etiquetados en función de sus similitudes o diferencias. Los algoritmos de agrupación de clústeres se pueden clasificar en exclusivos y superpuestos. Los exclusivos estipulan que un punto de datos solo puede existir en un clúster, mientras que los superpuestos permiten que un punto de datos pertenezca a múltiples clústeres. Los algoritmos de agrupación también pueden ser jerárquicos, donde los puntos de datos se aíslan inicialmente como agrupaciones separadas y luego se funcionan iterativamente sobre la base de similitud hasta que se logra un clúster.

Además, existen algoritmos probabilísticos que ayudan a resolver problemas de clústeres de estimación de densidad, los puntos de datos se agregan en función de la probabilidad que pertenezca a una distribución determinada.

La técnica de reglas de asociación es un método de aprendizaje no supervisado basado en reglas para detectar relaciones entre variables en un conjunto de datos determinado. Este método se utiliza para análisis de compra, lo que permite a las empresas comprender mejor la relación entre diferentes productos. El objetivo principal es entender los hábitos de consumo de los clientes para mejorar y desarrollar estrategias de venta más efectivas. Mediante la identificación de patrones de compra frecuente, las empresas pueden ofrecer recomendaciones personalizadas, optimizar la disposición de productos en tiendas físicas o en línea, y diseñar campañas de marketing más dirigidas hacia los consumidores (IBM, 2023).

1.4 Aprendizaje por refuerzo

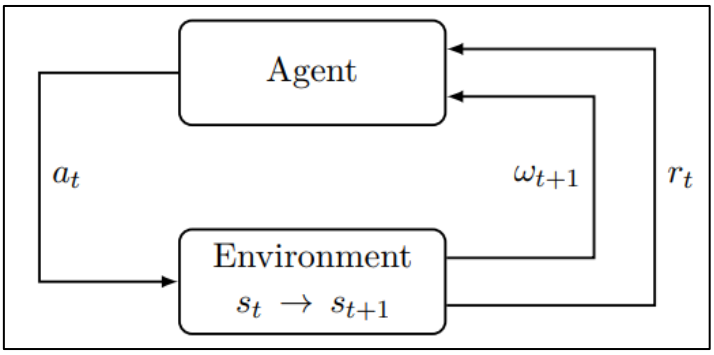
La inteligencia artificial basada en aprendizaje por refuerzo (RL) es el estudio de como un agente interactúa en un entorno y aprende en base a políticas cuyo objetivo es obtener la mayor recompensa frente a una tarea. (Henderson et al., 2019) Recientemente, el aprendizaje por refuerzo ha tenido un gran avance, abarcando temas como control continuo en sistemas de robótica, juegos de Atari, o cualquier juego competitivo. Uno de los aspectos claves del aprendizaje por refuerzo es que un agente aprenda un “buen comportamiento”; esto significa que modifica su comportamiento a fin de que obtenga la mayor recompensa posible. Para ello, debe hacer uso de la experiencia, es decir, a base de prueba y error, por lo que el agente no requiere tener un conocimiento o control completo del entorno; solo requiere que colecte información (Francois-Lavet et al., 2018).

1.4.1 *El entorno del aprendizaje por refuerzo*

Generalmente el RL es formalizado como un proceso de control estocástico en tiempo discreto (Francois-Lavet et al., 2018). El cual el agente, inicialmente, interactúa con su entorno de la siguiente manera: parte de un estado determinado dentro del entorno $s_0 \in \mathcal{S}$, dando consigue una observación $\omega_0 \in \Omega$, y por cada paso de tiempo t, el agente tomará una acción $a_t \in \mathcal{A}$, esto se puede observar de forma gráfica en la siguiente figura.

Figura 3

Interacción del agente en el entorno



En este punto el agente va a obtener lo siguiente: una recompensa $r_t \in \mathcal{R}$, un estado en transición $s_{t+1} \in \mathcal{S}$ y una nueva observación $\omega_{t+1} \in \Omega$. Partiendo de ello, Francois-Lavet, plantea que el futuro del proceso solo depende de la observación actual, y el agente no está interesando en conocer el historial completo del entorno. Esto lo define como una propiedad de Markov; que está compuesto por una tupla de dimensión (5,) expresada de la siguiente manera $(\mathcal{S}, \mathcal{A}, T, \mathcal{R}, \gamma)$ donde:

- $\mathcal{S}$ es el espacio de estado,
- $\mathcal{A}$ es el espacio de acciones,
- $T: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ es la función de transición,
- $\mathcal{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$ es la función de recompensa la cual es continua y pertenece al rango $R_{max} \in \mathbb{R}^+$,
- $\gamma \in [0,1)$ es el factor de descuento.

1.4.2 *Categorías de políticas*

Las políticas en aprendizaje por refuerzo definen como el agente selecciona las acciones ante su entorno (Francois-Lavet et al., 2018); estas políticas pueden ser estacionarias o no-estacionarias, es decir, depende del paso del tiempo. Por su parte, las no-estacionarias pueden ser utilizadas para un horizonte finito donde la recompensa que recibe el agente puede ser implementada para optimizar un número finito de paso de tiempo futuros. De igual forma, las políticas pueden categorizarse como estocásticas o determinísticas.

Las determinísticas hace referencia de que por cada estado existe una acción determinada que el agente tiene que realizar, $\pi(s): \mathcal{S} \rightarrow \mathcal{A}$. Mientras que las estocásticas hacen referencia a cuan probable es que el agente realice una determinada acción partiendo de su estado, $\pi(\mathcal{S}, \mathcal{A}): \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$.

Se espera que el agente puede encontrar una política $\pi(s, a) \in \Pi$, que pueda ser lo óptimo posible dando un rendimiento esperado $V^{\pi}(s): \mathcal{S} \rightarrow \mathbb{R}$, denominado también como función de valor, la cual viene expresada en la siguiente función.

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \, r_{t+k} | s_t = s, \pi\right] \qquad \text{Ecuación 1}$$

Donde:

- $\mathrm{r_t} = \mathbb{E}_{\mathrm{a} \sim \pi(\mathrm{s_t}, \cdot)} \mathcal{R}(\mathrm{s_t}, \mathrm{a}, \mathrm{s_{t+1}})$, representa los valores que se obtienen de la recompensa.
- $\mathbb{P}(\mathrm{s_{t+1}} | \mathrm{s_t}, \mathrm{a_t}) = \mathrm{T}(\mathrm{s_t}, \mathrm{a_t}, \mathrm{s_{t+1}})$ con $\mathrm{a_t} \sim \pi(\mathrm{s_t}, .)$, lo que significa que las observaciones nuevas que se han obtenido son producto de una nueva acción.

Teniendo en cuenta esto, el valor máximo de la función de valor viene expresado de la siguiente manera.

$$V^*(s) = \max_{\pi \in \Pi} V^{\pi}(s) \qquad \text{Ecuación 2}$$

Otra función que se debe añadir es la Q-value expresada como $Q^{\pi}(s,a): \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, está función se define de la siguiente manera:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} |\, s_t = s, a_t = a, \pi\right] \qquad \text{Ecuación 3}$$

Sin embargo, esta ecuación también se puede reescribir por medio de la ecuación de Bellman:

$$Q^{\pi}(s,a) = \sum_{s' \in S} T(s,a,s')\left(\mathcal{R}(s,a,s') + \gamma Q^{\pi}\left(s', a = \pi(s')\right)\right) \qquad \text{Ecuación 4}$$

De forma similar al determinar el valor máximo de la función Value, se puede determinar el valor máximo de la función Q-value.

$$Q^*(s,a) = \max_{\pi \in \Pi} Q^{\pi}(s,a) \qquad \text{Ecuación 5}$$

Y, partiendo de la función Ecuación 1 y Ecuación 3 se puede obtener la función de ventaja la cuales se expresa como restas de las funciones antes mencionadas.

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \qquad \text{Ecuación 6}$$

Cualquiera de estas ecuaciones se puede utilizar para el desarrollo del RL, este método se le denomina Monte Carlo. Cabe resalta que la ecuación Ecuación 4**¡Error! No se encuentra el origen de la referencia.** se puede reescribir de la siguiente manera:

$$Nuevo\ Q(s,a) = Q(s,a) + \alpha\left[\mathcal{R}(s,a) + \gamma\left(\max\left(Q^*(s,a)\right)\right) - Q(s,a)\right] \qquad \text{Ecuación 7}$$

Donde:

- α, es la tasa de aprendizaje,
- γ, factor de descuento,
- $Q(s,a)$, es el estado actual,
- $\mathcal{R}(s,a)$, la recompensa del estado s y la acción a.

1.5 Aprendizaje por refuerzo profundo (DQN)

1.5.1 *Q-learning*

Basado en el algoritmo que permite crear funciones de valor, con subsecuencia que nos permite definir una política. Principalmente, el aprendizaje por refuerzo profundo se basa en el algoritmo de aprendizaje por Q-learning y en su variante Q-learning ajustado.

Q-learning es una versión básica de la tabla de valores $Q(s,a)$ o comúnmente llamado, Qtable. Por cada valor de entrada da un par de acción-estado acuerdo al aprendizaje óptimo de la función Q-value. Por lo tanto, se va a tener en cuenta con la ecuación de Bellman llegando a la siguiente expresión matemática:

$$Q^*(s,a) = (\mathcal{B}Q^*)(s,a) \qquad \text{Ecuación 8}$$

Por lo que el operador de Bellman tiene la siguiente función $K: S \times A \rightarrow \mathbb{R}$, por lo que al aplicarla $Q(s,a)$ se obtiene lo siguiente.

$$Q^*(s,a) = (\mathcal{B}Q^*)(s,a) \qquad \text{Ecuación 9}$$

Esta ecuación lo que permite es que los valores que tiene en la función $Q(s,a)$ hacen que los nuevos valores a los cuales está operando, tiendan a converger dentro de un mapa creado.

1.5.2 *Ajuste Q-learning*

La experiencia reunida en el dataset "D" en una tupla $< s, a, r, s' >$ donde el estado en el siguiente tiempo denominado s' está representada por $T(s,a,\cdot)$ y cuya recompensa es $R(s,a,s')$. En el entrenamiento Q-learning, el algoritmo comienza con un valor inicial de la Q-value $Q(s,a;\theta_0)$ donde θ_0 refiere a un parámetro inicial. Después de eso se tiene una iteración de los $Q(s,a;\theta_k)$ es decir una iteración hacia el futuro con un valor determinado. Tomando el valor máximo del siguiente futuro como se observa en la siguiente expresión matemática.

$$Y_k^Q = r + \gamma \max_{a' \in A} Q(s', a'; \theta_k) \qquad \text{Ecuación 10}$$

Donde θ_k, representa con parámetros que se definen con la Q-value en el k^{th} iteración. γ hace referencia al factor de descuento. En un entrenamiento neuronal Q-learning (NFQ) el estado puede provenir de una entrada de Q-network y una salida diferente dada una acción posible.

Esto proporciona una estructura eficiente que tiene una ventaja de obtener un valor de computación expresado como $\max_{a' \in A} Q(s', a'; \theta_k)$ en un paso único hacia adelante en la red neuronal dado por s'. El valor de Q-value es parametrizado en la red neuronal como $Q(s,a;\theta_k)$, donde el parámetro θ_k son actualizados por el descenso de gradiente y por la pérdida de mínimos cuadrados.

$$L_{DQN} = \left(Q(s,a;\theta_k) - Y_k^Q\right)^2 \qquad \text{Ecuación 11}$$

La actualización del aprendizaje de Q-learning se actualiza los parámetros basándose en la siguiente ecuación.

$$\theta_{k+1} = \theta_k + \alpha \left(Y_k^Q - Q(s, a; \theta_k) \right) \nabla \theta_k Q(s, a; \theta_k) \qquad \text{Ecuación 12}$$

Donde α representa la tasa de aprendizaje que contiene en cada iteración. La Ecuación 12 es otra forma de expresar a la Ecuación 7**¡Error! No se encuentra el origen de la referencia.** siendo este parámetro el necesario para poder implementar en el algoritmo de DQN.

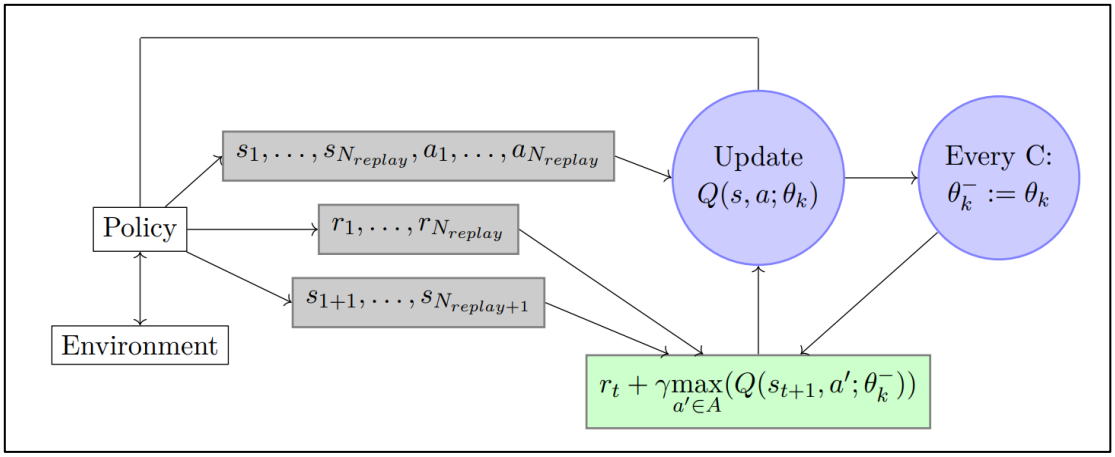
1.5.3 *Deep Q-networks*

El objetivo de una Q-network basada en la Ecuación 10 es reemplazar por $Q(s', a'; \theta_k^-)$ donde cada valor del parámetro θ_k^-, representa la actualización de cada $C \in \mathbb{N}$ iteraciones con la siguiente asignación $\theta_k^- = \theta_k$. Previniendo inestabilidades para propagarse rápidamente y reducir el riesgo de divergencia. La idea de una red puede ser vista en una instancia de entrenamiento con Q-learning, donde cada periodo de actualizaciones de la red de destino corresponde a una única Q-iteración ajustada. La memoria de repetición guarda toda la información del último paso de tiempo, donde la experiencia es recolectada siguiendo una política. Esta actualización está hecha de una tupla $< s, a, r, s' >$.

Esto permite que técnicamente las actualizaciones cubran una gama amplia en un espacio de acciones; y con la ayuda de un *mini-batch*, tiene que comparar cual de todos tiene menos variancia para una actualización de la tupla. Por consiguiente, proporciona la posibilidad de hacer una actualización larga de los parámetros.

Figura 4

Algoritmo DQN



DQN emplea otras heurísticas importantes para mantener los valores objetivos a una escala razonable y para garantizar un aprendizaje adecuado en la práctica, la recompensa se debe mantener en un rango entre -1 y +1.

1.6 Métodos de gradiente de políticas para RL profunda

Este es un método particular en la familia de los algoritmos de aprendizaje por refuerzo, el cual utiliza la política de la gradiente. Estos métodos optimizan un objetivo de rendimiento, gracias a una variante del ascenso de la gradiente estocástica con respecto a la política de los parámetros.

Capítulo 2
Robot Operating System (ROS)

2.1 Introducción a Robot Operating System

Robot Operating System es un marco de trabajo en la escritura de software para robots que incluye librerías, herramientas y convenciones diseñadas para simplificar el desarrollo de comportamientos complejos y robustos en una amplia variedad de plataformas robóticas (Quigley et al., 2015).

ROS es un sistema que se nutre tanto directa o indirectamente de su propia comunidad para su desarrollo. Al ser de código abierto, los programadores comparten la filosofía de colaboración, lo que facilita que la programación se sienta más intuitiva y natural. Además, ROS es compatible con una variedad de entornos como Windows o Mac OS X. Asimismo, hay que tener en cuenta algunos aspectos de ROS que lo convierten en una de las estructuras adecuadas para el trabajo con la robótica.

- *Peer to peer:* El sistema ROS permite que múltiples programas de computación se conecten entre sí continuamente, compartiendo mensajes de forma bidireccional. Esto contribuye a la optimización de líneas de código.

- *Multilingual:* Mucho software requiere de lenguajes de programación más efectivos como Python; sin embargo, es evidente que algunos casos se necesitan lenguajes más rápidos como C++, o por preferencia optan por lenguajes como MATLAB. ROS permite la comunicación entre diferentes tipos de lenguajes, lo que hace que la programación sea más adaptable según las necesidades del desarrollador.

2.1.1 *Conceptos preliminares*

Dentro de la configuración de ROS, existe conceptos fundamentales que son necesarios conocer para su ejecución y su uso adecuado. Estos conceptos se basan en una amplia variedad de programas independientes cuyo objetivo es facilitar la conexión entre nodos y la comunicación entre ellos. Morgan Quigley los explica a detalle en su libro:

- *ROS GRAPH:* Su objetivo es representar los programas que están ejecutando simultáneamente y la comunicación que existen entre ellos mediante el intercambiando de mensajes.

- *ROSCORE:* Es un servicio que facilita la conexión entre nodos. Debe iniciarse previamente para que los programas puedan intercambiar información entre sí. Se inicializa escribiendo en la terminal “roscore”.

- *WORKSPACES:* Es una carpeta que se debe crear previamente y que va a alojar todos los scripts con lo que se va a trabajar.

- *ROSRUN:* Es una forma de ejecutar un código dentro de un paquete ubicado en el workspace.

2.1.2 *Tópicos*

Cada archivo de programa puede ser asignado con un nombre específico, considerado como nodo. El propósito de esto es permitir la identificación de los elementos que se están desarrollando, facilitando su visualización en ROS Graph.

Por otra parte, los tópicos se utilizan para definir el mecanismo de comunicación que cada nodo va a utilizar. Se les asigna un nombre y un tipo de tópico. Entre los más comunes se encuentran los "publisher" y "subscriber", que determinan cómo se van a transferir los datos.

• *Publisher:* Se encarga de enviar información desde un nodo a otro que esté suscrito al mismo nombre de tópico. Los datos son de tipo booleano, Int, float, long, string, arrays, entre otros. Se debe especificar el tipo de dato que se va a publicar, por ejemplo, Int32 o Int64. En Python, la estructura es la siguiente:

```
rospy.Publisher('nombre_del_topico', tipo_de_dato)
```

• *Subscriber:* Se encarga de recibir los datos enviados por un Publisher. Normalmente, se asocia a una función que tiene como argumento una variable msg. Dentro de esta función, se puede acceder a los datos recibidos a través de la línea de código msg.data. Esto permite procesar los datos que se están enviando. En Python, la estructura es:

```
rospy.Subscriber('nombre_del_topico',tipo_de_dato,
nombre_de_funcion)
```

Tanto el nodo *Publisher* como el nodo *Subscriber* deben tener el mismo nombre de tópico para poder comunicarse correctamente. Se pueden asignar varios tópicos en un mismo nodo, lo que significa que un nodo puede recibir información de otro nodo, procesarla y publicarla hacia otro nodo.

2.2 Configuración de ROS en Raspberry Pi

Primero, se debe instalar una versión de ROS en la Raspberry Pi. Los pasos detallados se encuentran en la página web de ROS. En este caso, se trabajará con ROS Noetic. Posteriormente, se realiza una actualización de la Raspberry Pi a través del terminal de la siguiente manera:

```
pi@navio:~ $ sudo apt-get update
```

Cabe resaltar que la versión de ROS con la que se está trabajando es la 1.15.14. El comando anterior permite actualizar e instalar las dependencias necesarias en la Raspberry Pi. Una vez completada la actualización, se procede a verificar si la librería Noetic está instalada utilizando el siguiente comando:

```
pi@navio:/opt/ros/noetic
```

Al ejecutar ese comando, se accede a la carpeta noetic. En caso contrario, se mostrará un mensaje indicando que no se puede acceder a esa librería o que no existe. Esta dirección contiene archivos en los cuales se deben realizar algunas modificaciones. Para visualizar qué archivos se encuentran dentro de un directorio en la Raspberry, se utiliza el comando "ls".

```
pi@navio:/opt/ros/noetic $ ls
bin etc lib local_setup.sh setup.bash _setup_util.py share env.sh
include local_setup.bash local_setup.zsh setup.sh setup.zsh
```

El archivo que se va a modificar es “setup.bash”, con el fin de poder dar permiso para que se pueda activar “roscore”, para realizar el cambio se debe poner el siguiente comando en la consola:

```
pi@navio:~ $ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
pi@navio:~ $ gedit ~/.bashrc
```

Esto hará que se abra el archivo “setup.bash”, el cual contiene un código; y en la parte final de este, se va a proceder quitar el comentario de unas líneas de código (ver anexo A).

```
source /opt/ros/noetic/setup.bash
source ~/catkin_ws/devel/setup.bash
```

Con esta modificación, se podrá ejecutar "roscore" desde cualquier terminal dentro de la Raspberry Pi. Estos son los permisos necesarios que se deben configurar para el correcto funcionamiento de ROS.

2.3 Creación de Workspace

Como se mencionó anteriormente, se debe de crear un workspace donde va a encontrar todos los scripts que se va a estar trabajar. En el terminal se empleará el siguiente comando.

```
pi@navio:~ $ mkdir catkin_ws
```

El comando “mkdir” hace referencia que se va a crear una carpeta o un nuevo directorio y mientras que “catkin_ws” es el nombre que va a tener este directorio. Para acceder a ese directorio se hace por medio del comando “cd” seguido del nombre que hemos creado. Dentro de este se creará otra carpeta llamada “src” se realizando el mismo procedimiento.

```
pi@navio:~/catkin_ws $ mkdir src
```

Para crear la construcción del espacio, se crearán dos carpetas adicionales “build” y “devel”, aparte de la carpeta “src” que se ha creado anteriormente, para ello se inserta el comando:

```
pi@navio:~/catkin_ws $ catkin_make
```

Hasta este punto, se tiene la creación del espacio de trabajo, el cual se deberá acceder y posteriormente agregar los *Package* y scripts requeridos para el proyecto.

2.4 Creación de Package

Dentro del directorio "src" se va a crear un *Package*, el cual va a alojar los scripts donde se va a alojar el código; a este *Package* se le va a asignar un nombre, que por fines de la investigación se llamará "AIDRA" el cual significa *Artificial Intelligence Disaster Response Aid* cuyo significado en español es "ayuda de respuesta a desastres con inteligencia artificial" el cual forma parte del objetivo de la investigación; para la creación como tal se debe utilizar el siguiente comando.

```
pi@navio:~/catkin_ws/src $ catkin_create_pkg aidra rospy turtlesim
```

El comando "catkin_create_pkg", permite la creación de tres archivos "package.xml", "CMakelists.txt" y "src". Mientras que el comando turtlesim es un paquete con el que se trabaja por defecto en ROS.

2.5 Creación de archivos

En el directorio aidra se va a crear otro directorio el cual tendrá como nombre "scripts" el cual permitirá crear los scripts donde va a alojar los nodos.

```
pi@navio:~/catkin_ws/src/aidra $ mkdir scripts
pi@navio:~/catkin_ws/src/aidra $ ls
CMakeLists.txt  package.xml  scripts  src
```

Se ingresa al directorio scripts, y se creará el archivo Python con en el que se va a trabajar, para ello se utiliza el comando "touch" seguido del nombre del archivo incluida la extensión ".py". Después, se utiliza un comando para dar permisos al archivo para que se pueda editar desde un IDLE, a través del comando "chmod +x".

```
pi@navio:~/catkin_ws/src/aidra/scripts $ touch sensors.py
pi@navio:~/catkin_ws/src/aidra/scripts $ chmod +x sensors.py
```

De esta manera, se puede crear todos los archivos que se requieran para el proyecto, simplemente se cambia el nombre y se sigue los mismos pasos.

2.6 División de pantallas en el terminal multiplex

Cuando se trabaja con ROS, es común ejecutar múltiples scripts simultáneamente, por lo que es recomendable utilizar más de un terminal para esta tarea. En Raspberry Pi, existe la opción para llamada "multiplex", lo cual permite dividir la pantalla según sea necesario. Esto es útil cuando se trabaja desde la consola de comandos de la Raspberry. Para crear una pantalla multiplex, se debe ejecutar el siguiente comando:

```
tmux new -s session-name
```

El comando "tmux" especifica el tipo de ejecución que se realizará, en este caso, la creación de un multiplexor de pantallas. El comando "new -s" indica que se creará una nueva sesión con el nombre que se desee para trabajar. Si se tiene una sesión creada y se desea unirla con otra nueva o manejar múltiples pantallas de trabajo, se puede hacer mediante el siguiente comando:

```
tmux a -t session-name
```

Se resalta que el “sesión-name” hace referencia a la sección creada con anterioridad a la cual se va a unir. Para desconectarse con una sección se usa el comando:

```
tmux detach
```

Por último, para la eliminación de una sección simplemente se utiliza el comando que se observa a continuación:

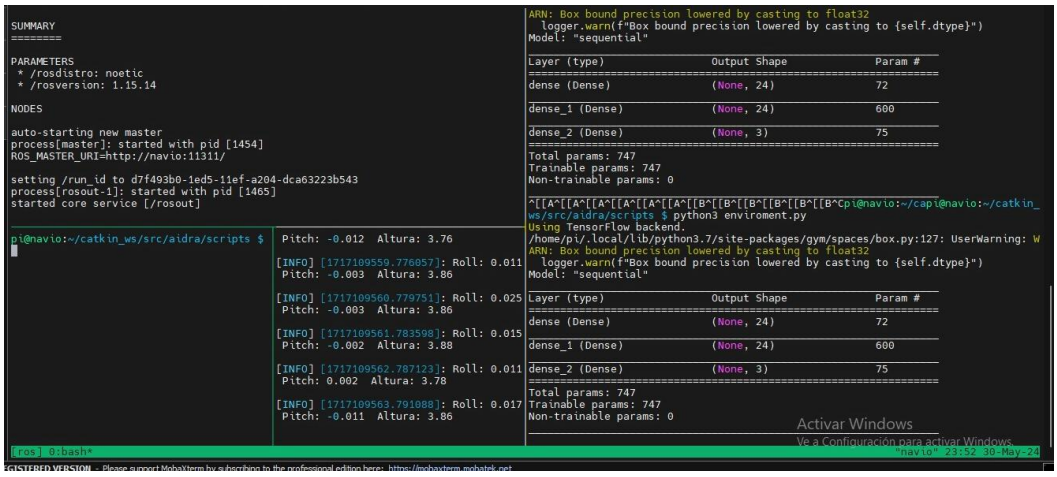
```
tmux kill-session -t session-name
```

Con esto se estará creando las secciones donde se va a dividir las pantallas que se deseen trabajar, pueden ser tanto de forma horizontal como vertical, Asimismo, se puede desplazar por cada terminal. A continuación, se mostrará un resumen de los comandos rápidos para las acciones antes mencionadas.

- Para mostrar teclas rápidas *“*`ctrl+b +?`”
- Para poner nombre a la sesión “*`ctrl+b +$`”*
- Para dividir horizontal “`ctrl+b +%`”
- Para dividir vertical “`ctrl+b +”“`
- Para cambiar de una ventana a otra “`ctrl+b + o`”
- Para eliminar una ventana “`ctrl+b + x`”

Figura 5

Tmux división de pantalla



En la figura que se observa anteriormente, se muestra cómo queda la pantalla dividida donde en cada pantalla puede ejecutarse un script o activar roscore.

2.7 Programación de conexión ROS con los nodos

Se han creado tres archivos. El primero se llama sensors.py y contiene la programación para los sensores. El segundo se llama servo.py y contiene la programación para el movimiento de los servomotores. Y el tercer archivo se llama environment.py y contiene el entorno y el agente para implementar la inteligencia artificial.

2.7.1 *Nodo sensores*

El nodo "sensores" enviará información hacia el nodo "environment" sobre los ángulos roll, pitch y la altura. Para inicializar el nodo, se utilizará la siguiente línea de código:

```
rospy.init_node('sensor_node',anonymous = True)
```

Asimismo, se asignará una variable llamada "pub" que será de tipo Publisher. Para ello, se requieren algunos argumentos como el nombre del tópico, el tipo de dato que se está enviando y la frecuencia de envío de los datos.

```
pub = rospy.Publisher('sensors_topic', Float32MultiArray,
queue_size = 10)
```

Como se puede apreciar, el nombre del tópico es "sensors_topic", lo que implica que el otro nodo que va a recibir los datos debe tener el mismo nombre. Además, el tipo de dato que se envía es un array, debido a que se envía tres datos, lo cual es más eficiente hacerlo juntos en un array para evitar la creación de otros tópicos o la modificación de mensajes.

Es importante destacar que se deben importar las dependencias necesarias tanto para inicializar el nodo como para enviar los mensajes. Para inicializar el nodo, se importa simplemente “rospy”, mientras que para definir el tipo de mensaje que se va a enviar se debe importar el tipo de mensaje específico necesario.

```
import rospy
from std_msgs.msg import Float32MultiArray
```

Después se asigna una variable array_msg del objeto Float32MultiArray, de igual forma, un array con los datos de los sensores.

```
sensors_data =[roll_ang,pitch_ang,altura]
array_msg = Float32MultiArray ()
```

Finalmente, al array_msg se le asigna los datos que va a contener con el fin de utilizar la función publish, para enviar los datos.

```
array_msg.data = sensors_data
```

```
pub.publish(array_msg)
```

Si se desea que se muestre los valores que se están enviando, puede hacer uso de la función loginfo como se muestra a continuación.

```
rospy.loginfo(roll_str + pitch_str + ' '+ alt_str)
```

Esto permite imprimir los datos en forma de cadena de texto String en la terminal de la consola. Este tipo de publicación también proporciona el tiempo en el que se está enviando la información.

2.7.2 *Nodo enviroment*

El nodo *enviroment* va a recibir información del nodo *sensor,* por lo que se debe importar el mismo tipo de mensaje.

```
import rospy
from std_msgs.msg import Int32MultiArray
from std_msgs.msg import Float32MultiArray
```

En este caso se ha importado otro tipo de mensaje “Int32MultiArray”, esto es debido a que el nodo *enviroment* será un *Publisher* y un *Subscriber*, entonces el tipo de dato que va a enviar será un array de enteros.

Para este nodo, se inicializa de la misma manera que se ha inicializado anteriormente, y va a tener como nombre *enviroment_node.*

```
rospy.init_node('enviroment_node',anonymous = True)
```

Como este nodo va a funcionar como *Subscriber* de “sensor_node” debe tener el mismo tópico. Y el mismo tipo de archivo, por lo que la línea de código será la siguiente.

```
rospy.Subscriber('sensor_topic', Float32MultiArray, self.callback)
```

Cabe resaltar que el objeto *Subscriber* pide como argumento el nombre del tópico, el tipo de dato que se envía y la función donde va a llegar el dato. En este caso está función tiene el formato “self.” Debido a que es una clase donde se ha inicializado el código, en el Capítulo 4 se mostrará a detalle la estructura y la lógica de los nodos.

Para el envío de los datos que ha procesado del nodo *enviroment* hacia el nodo servo, se debe hacer de la misma forma que utiliza el nodo *sensors*.

```
self.pub = rospy.Publisher('controlador', Int32MultiArray,
queue_size = 10)
```

Se asigna una variable, en este caso “self.pub” hacia el objeto Publisher, y tiene el formato self. porque está dentro de una clase. Para el envío de la información se realiza la misma estructura del nodo sensor; el nodo *enviroment* va a enviar un array hacia el nodo servo.

```
altura_data = nuevo_estado
array_msg.data = altura_data
self.pub.publish(array_msg)
```

2.7.3 *Nodo servo*

Por último, se tiene el nodo servo, en este caso será de tipo *Subscriber* y va a recibir los datos del nodo *enviroment*, como se inició en los nodos anteriores, se importa las dependencias que se requiere, que en este caso será del tipo “Int32MultiArray”.

```
from std_msgs.msg import Int32MultiArray
```

Y luego se asigna el nodo y el tópico que se requiera.

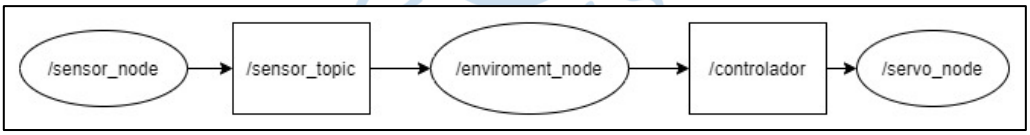
```
rospy.init_node('servo_node', anonymous=True)
sub = rospy.Subscriber('controlador', Int32MultiArray, callback)
```

2.7.4 *Graph de los nodos creados*

Como se está trabajando desde una Raspberry y se opera desde la consola, no se cuenta con una interfaz gráfica para visualizar el grafo de nodos y tópicos utilizando el comando "rosgraph". Sin embargo, se ha optado por crear una representación del esquema de los nodos y sus tópicos, similar a lo que ofrece el comando "rosgraph", como se observa en la Figura 6.

Figura 6

Diagrama ROS GRAPH nodos



En la figura, se observa de forma resumida el método de comunicación que se ha explicado anteriormente, con los nodos y el tópico que los conecta entre sí.

Capítulo 3
Ensamblaje e instalación de componentes

3.1 Diseño y ensamblaje del robot

Las partes del robot han sido obtenidas mediante impresión 3D con resina. El diseño completo del prototipo está disponible en los anexos de esta investigación. Parte del diseño es de código abierto *Open Source*, lo que permite descargarlo, realizar modificaciones o utilizar el mismo modelo.

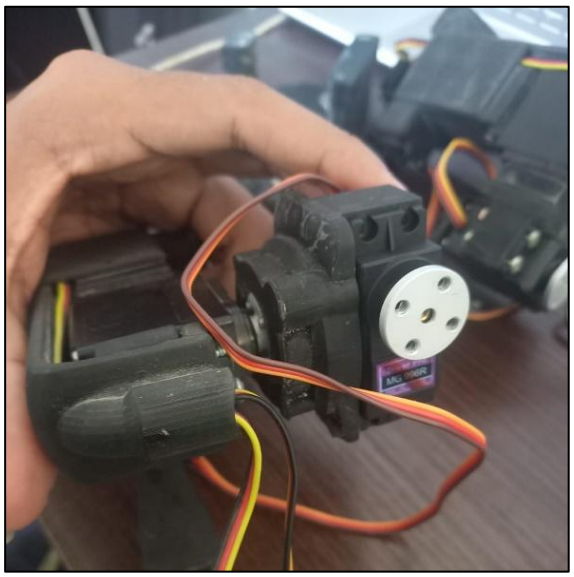
Figura 7

Partes del prototipo impresas

Este prototipo cuenta con cuatro extremidades, estas extremidades están conformadas por muñecas que van conectados a presión y en algunas partes pernos hacia los brazos medios, y hacia los hombros. Entre cada una de las partes va a estar un servomotor.

Figura 8

Extremidad expuesta hombro derecho

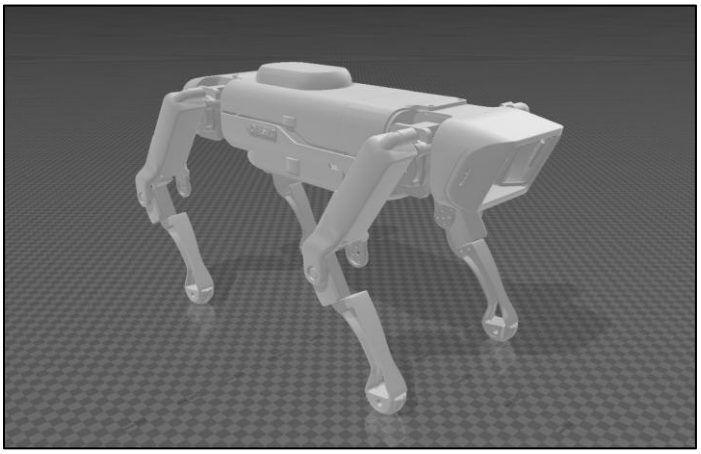


Cuenta con un chasis y una estructura donde tendrá el espacio suficiente para poder insertar los procesadores como la placa Navio2 y que tenga espacio para que pueda tener los cables de los servomotores ordenados adecuadamente.

En la Figura 9 se muestra un renderizado del prototipo con las piezas ensambladas. Para fines prácticos, se han impreso únicamente las piezas necesarias para este proyecto, como las extremidades y el chasis del robot.

Figura 9

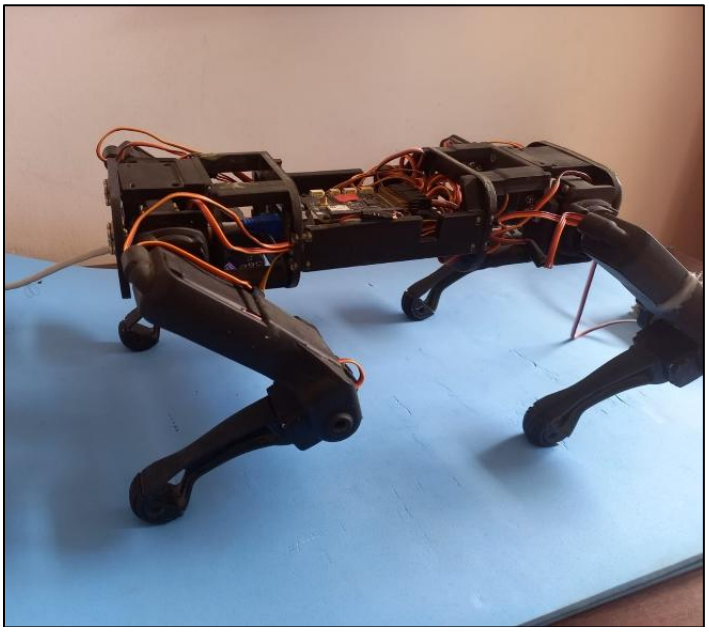
Renderizado de robot cuadrúpedo



En la Figura 10, se muestra el aspecto final del prototipo una vez que todas las piezas han sido ensambladas y configuradas según el diseño.

Figura 10

Prototipo final



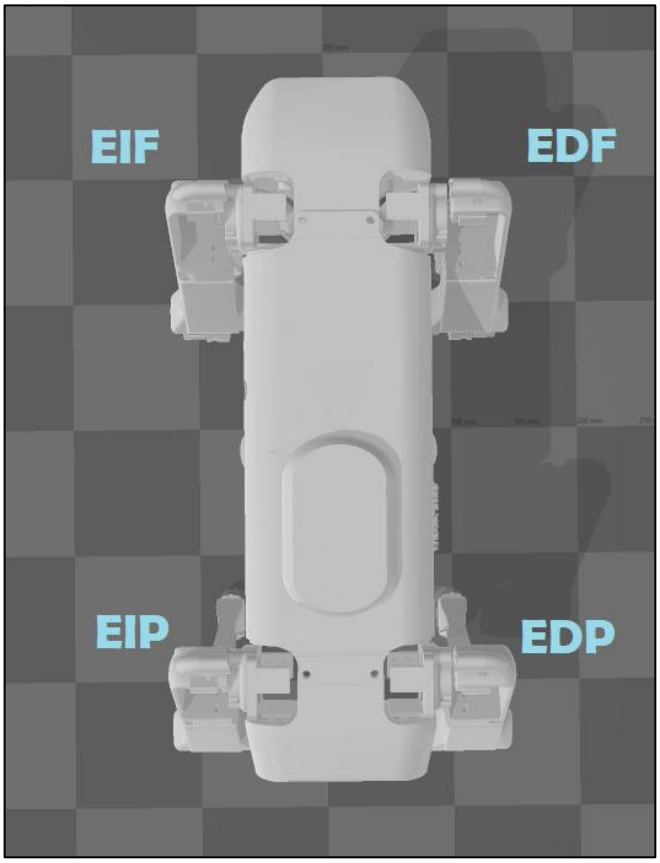
3.1.1 *Asignación de las extremidades*

Para mantener un orden de los componentes del robot, se ha decidido establecer una nomenclatura para las extremidades basada en su posición, lo cual facilitará tanto la construcción como la programación.

Las extremidades serán denominadas por las siglas EDF, EIF, EDP y EIP. La sigla comienza con "E" que hace referencia a Extremidad. Las letras "D" e "I" indican si la extremidad está ubicada a la Derecha o Izquierda del cuerpo del cuadrúpedo, respectivamente. Las letras "P" y "F" determinan si la extremidad se encuentra en la parte Posterior o Frontal del robot. En la siguiente figura se muestra la vista en planta del cuadrúpedo, donde se indica la nomenclatura mencionada anteriormente.

Figura 11

Vista de planta del robot cuadrúpedo



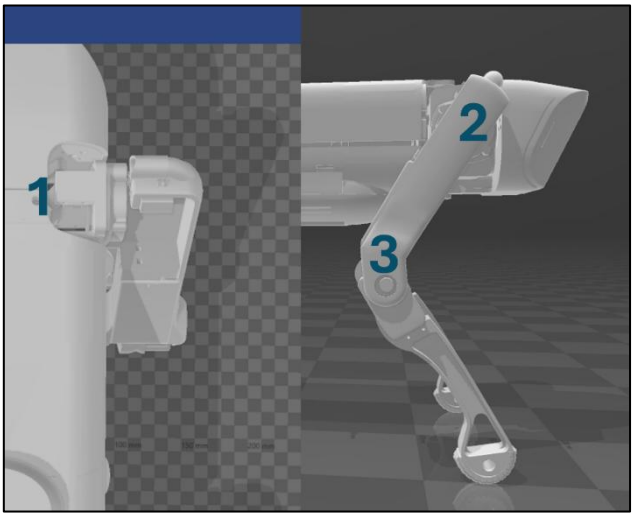
3.1.2 *Asignación de articulaciones*

Cada extremidad contará con tres servomotores, los cuales serán las articulaciones de cada extremidad. Se asignará un número para identificarlos, siguiendo la nomenclatura mencionada anteriormente.

Para una mayor coherencia, los números se asignarán de forma descendente, desde uno hasta tres, correspondiendo de arriba hacia abajo en la extremidad. En la Figura 8 se muestra más detalladamente esta nomenclatura.

Figura 12

Numeración de servomotores



Con esta nomenclatura, se va a tener un mejor la ubicación de los servomotores lo que aporta a su ensamblaje y para la asignación de las variables en la programación que se va a ver más a adelante. A modo de resumen, se tiene la siguiente tabla de los servomotores que se van a requerir y las articulaciones.

Tabla 1

Asignación de articulaciones

Ítem	Sigla	Descripción
1	EDF1	Extremidad Derecha Frontal articulación 1
2	EDF2	Extremidad Derecha Frontal articulación 2
3	EDF3	Extremidad Derecha Frontal articulación 3
4	EIF1	Extremidad Izquierda Frontal articulación 1
5	EIF2	Extremidad Izquierda Frontal articulación 2
6	EIF3	Extremidad Izquierda Frontal articulación 3
7	EDP1	Extremidad Derecha Posterior articulación 1
8	EDP2	Extremidad Derecha Posterior articulación 2
9	EDP3	Extremidad Derecha Posterior articulación 3
10	EIP1	Extremidad Izquierda Posterior articulación 1
11	EIP2	Extremidad Izquierda Posterior articulación 2
12	EIP3	Extremidad Izquierda Posterior articulación 3

3.2 Placa de control Navio2

Navio2 es una placa de control equipada por sensores y puertos para controladores con la finalidad de poderse adaptar a una Raspberry Pi, siendo de este modo una forma portátil de tener controladores y un microprocesador que puede utilizarse para drones o fines robóticos. Dentro de sus componentes principales se encuentran los siguientes:

- *Sensores IMU:* contiene dos sensores que brindan información de acelerómetro, giroscopio y magnetómetro con el fin de tener una orientación y un sentido de movimiento. Los sensores son MPU9250 y LSM9DS1.

- *Receptor GNSS:* es capaz de realizar un seguimiento de los satélites GLONASS, Beidou, Galileo y SBAS por medio de GPS, por medio de una antena externa con conector MCX.

- *Fuente de alimentación triple redundante:* Esto con el fin de evitar sobre tensiones, y también contiene un puerto de alimentación para detección de tensiones y corrientes.

- *Co-procesadores RC I/O:* recibe entradas PPM/SBUS y permite salidas PWM en 14 canales para motores y servomotores.

- *Puertos de extensión:* expuesto a I2C, ADC, y UART, para sensores y radios.

- *Barómetro de alta resolución:* capaz de detectar una altitud con resolución de 10cm.

Figura 13

Placa Navio2 en una Raspberry Pi

Navio2, es preconfigurado con Raspbian, el cual contiene instalado ArduPilot y ROS listo para ser ejecutados en un par de simples comandos. Asimismo, contiene DroneKit y GStreamer (Navio2 – Autopilot HAT for Raspberry Pi, s. f.).

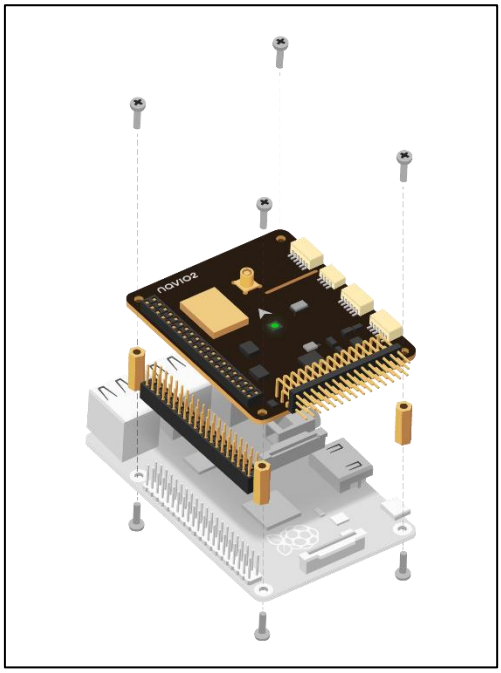
3.2.1 *Instalación de Navio2*

La Navio2, como se mencionó anteriormente, es compatible con ordenadores monoplacas como Raspberry Pi, por lo que se va a utilizar el modelo Raspberry Pi 4 B, el cual cuenta con una memoria RAM de 4GB, permitiendo procesar información y soportar procesamiento de datos y cálculos de computación.

La Navio2 cuenta con los componentes necesarios para realizar el montaje de esta sobre la Raspberry. Posee una extensión al puerto GPIO de 40 pines, tornillos y espaciadores de longitud necesaria entre la Navio2 y la Raspberry Pi 4, como se observa en la siguiente figura.

Figura 14

Ensamblaje de Navio2 a la Raspberry



Con esto se adapta adecuadamente a placa Raspberry Pi, y está lista para poder trabajar con los componentes que sean necesarios.

3.2.2 *Configuración de Navio2*

Navio2 requiere de una preconfiguración en la Raspberry para comenzar a ejecutarse; por lo que se necesita una SD e instalar una imagen Raspbian que se encuentran en la página(*Raspberry Pi Configuration | Navio2*, s. f.).

La Navio2, al estar conectada con una Raspberry se puede programar conectando una micro HMI y luego a un monitor. Asimismo, se puede conectar un teclado y un mouse y programar desde ahí.

Sin embargo, se puede agregar un archivo que tiene como nombre wpa_supplicant.conf con el fin de poder ingresar una red Wifi, así como también se debe configurar para que se tenga acceso vía SSH y se puede programar desde una computadora por medio de otro software como puede ser Putty o MobaXterm.

Dentro de este archivo wpa_supplicant.conf, va a ingresar un código donde tenga la contraseña y el nombre de la red Wifi a que se acceda.

```
network={
  ssid="yourssid"
  psk="yourpasskey"
  key_mgmt=WPA-PSK
}
```

Con esto, cada vez que se encienda la Raspberry y la red se encuentre activa, se podrá acceder de manera automática. Cabe resaltar que los softwares que se mencionaron anteriormente requieren de un IP, por lo que se puede utilizar un escáner de IP o asignar un IP fijo ala Raspberry.

3.3 Servomotores

Se debe de contar con doce servomotores, en este caso se va ha optado por los servomotores Tower pro-servo MG946R, este servomotor cuenta con la característica que puede generar torques de 10.5Kg/cm a 4.8V hasta 14Kg/cm a 6.0V.

Figura 15

Servomotor pro MG946R

Asimismo, este cuenta con reducción metálica y notable robustez eficaz para proyectos como brazos robóticos, bípedos o artrópodos en general. Las características principales de este servomotor son las siguientes:

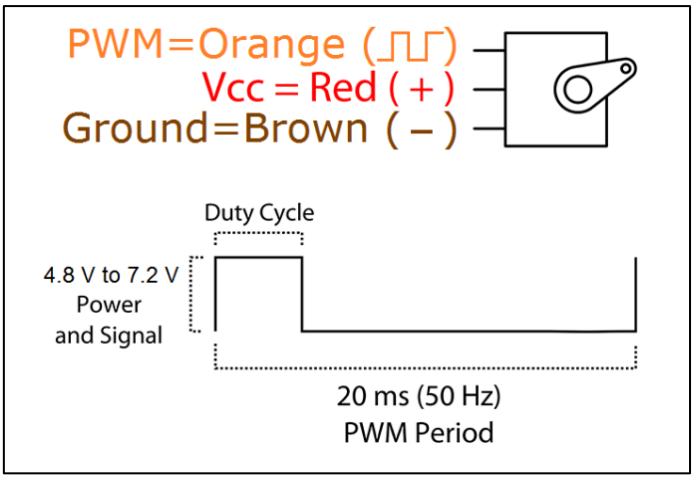
- Voltaje de alimentación: 6.0V – 7.2V DC

- Stall Torque: 10.5Kg/cm (4.8V); 14Kg/cm (6.0V)
- Velocidad de Funcionamiento: 0.20seg/60º (4.8V sin carga); 0.17seg/60º (6V)
- Ángulo de Rotación: 0-180°
- Periodo de pulso: 20ms
- Ancho del pulso: entre 500us y 2400us
- Dead Band Width: 20useg
- Plug: JR, FUTABA general
- Cable de conexión de 300mm
- Engranajes de metal
- Temperatura de Trabajo: 0ºC hasta +55ºC
- Dimensiones: 40.6*19.8*42.9 mm
- Peso: 55 gramos (*Servo MG946R 13kg - Tower Pro Original*, s. f.)

Los servomotores trabajan con PWM (Pulse Width Modulation) o Modulación por ancho de pulso, es un tipo de señal de voltaje utilizada para enviar información, o para que se pueda modificar la cantidad de energía que se envía a una carga. Los servomotores, cuenta con 20ms (50Hz) como periodo de PWM, Por otra parte, el *DutyCycle*, va a determinar el ángulo que va a mover, esta señal se va a enviar por medio del cable de color naranja.

Figura 16

PWM para servo motores



Nota: tomado de (*Servomotor MG996R*, s. f.).

Estos servomotores van a estar conectados a la Navio2, cada uno de ellos va a estar en un pin en la placa, algunos de ellos, en específicos los de la articulación tres, se tendrán que extender para que puedan llegar hasta el procesador. Su ubicación es de manera estratégica, cada servomotor en su pin de la placa se ha seleccionado de tal forma que pueda ser más cómodo y fácil de reconocer, y que la placa esté posiciona adecuadamente a la estructura del robot.

Con la nomenclatura que se ha mencionado en Asignación de articulaciones, se va a trabajar los servomotores y en la siguiente tabla se va a mostrar los pines a los que van a estar conectados; asimismo, para su programación cada pin corresponde a un valor anterior, es por eso en la tabla se ha agregado una columna con el valor del código que le corresponde.

Tabla 2

Configuración de servomotores.

NAME:	PIN	CODE
EDF 1	13	12
EDF 2	12	11
EDF 3	11	10
EIF 1	1	0
EFI 2	2	1
EFI 3	3	2
EPD 1	8	7
EPD 2	9	8
EPD 3	10	9
EPI 1	6	5
EPI 2	5	4
EPI 3	4	3

Estos servomotores, por la forma en cómo trabajan, son más que suficientes para el prototipo y las pruebas que se requiera.

3.4 Sensor ultrasónico

El sensor ultrasónico que se va a utilizar es el HC-SR04 debido a que es el más adecuado para medir distancia de un rango de 2 a 450 cm. y a bajo precio, bajo consumo energético y cuenta con buena precisión.

Este sensor se emplea en la mayoría de los proyectos de robótica. El HC-SR04, cuenta con 4 pines, dos de ellos son para los transductores, otro de sus pines es su alimentación y el otro es un GND. Los transductores son emisor y un receptor piezoeléctrico.

El transductor emisor emite 8 pulsos a 40kHz luego de recibir la orden del pin *TRIGGER*, la onda rebota en el objeto y regresa en el transductor receptor, luego lo decepciona el pin *ECHO*.

Figura 17
Sensor ultrasónico HC-SR04

Este dispositivo debe estar conectado con un controlador para poder procesar la información, por lo que se hará por medio de un Arduino UNO. El HC-SR04, cuenta con las siguientes especificaciones:

- Voltaje de Operación: 5V DC
- Corriente de reposo: < 2mA
- Corriente de trabajo: 15mA
- Rango de medición: 2cm a 450cm
- Precisión: +- 3mm
- Ángulo de apertura: 15°
- Frecuencia de ultrasonido: 40KHz
- Duración mínima del pulso de disparo TRIG (nivel TTL): 10 µS
- Duración del pulso ECO de salida (nivel TTL): 100-25000 µS
- Dimensiones: 45*20*15 mm
- Tiempo mínimo de espera entre una medida y el inicio de otra 20ms recomendable 50ms(Sensor Ultrasonido HC-SR04, s. f.).

Estas características son adecuadas para poder trabajar en el prototipo; la ubicación del sensor puede estar en la parte frontal o en la parte posterior, pero sobre todo debe estar posicionado en la parte de abajo del chasis. Con el fin de que se pueda implementar para determinar la altura a la que se encuentra el robot desde el suelo hasta el chasis.

Se ha empleado un Arduino UNO, para la lectura del sensor de ultrasonido y se envía por vía serial. Para ello se emplea las ecuaciones de cinemática de la aceleración.

$$v = \frac{distancia}{tiempo}$$ Ecuación 13

Tomando como referencia a la velocidad del sonido que es 340m/s, se debe convertir a cm/us.

$$340\frac{m}{s} \times \frac{1s}{1\,000\,000\mu s} \times \frac{100cm}{1m} = \frac{2d}{t}$$

La distancia recorrida tiene que ser el doble de la distancia a la que se encuentra el objeto. Por lo tanto, simplificando la expresión y despejando el valor de la distancia, se obtiene lo siguiente.

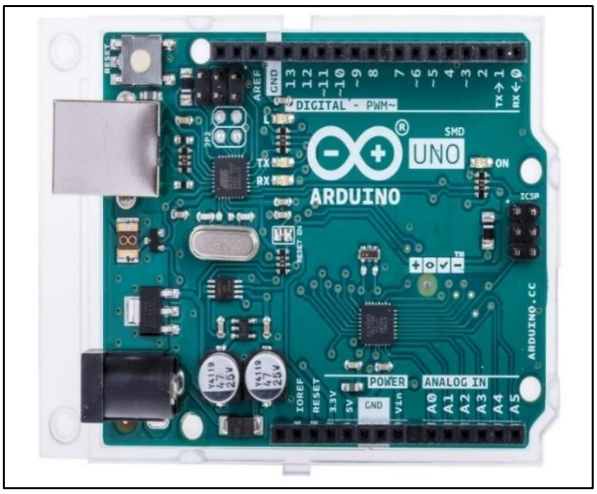
$$d[cm] = \frac{t[\mu s]}{59\ \left[\frac{cm}{\mu s}\right]}$$ Ecuación 14

Con la expresión Ecuación 14 es la ecuación que se deberá tener en cuenta cuando se inserte en microcontrolador Arduino.

3.5 Arduino UNO

Arduino UNO es una placa de desarrollo que utiliza un microcontrolador Atmega328P(Atmel/Microchip), Cuenta con 14 entradas y salidas digitales las cuales puede ser utilizadas para como PWM, seis entradas analógicas, resonador cerámico de 16MHz, cuenta con conexión USB, conector de alimentación, conectar ICSP y un botón para reiniciar.

Figura 18
Arduino UNO



Es una de las placas con mayor uso para proyectos básicos en áreas como robótica o sistema de IoT. Arduino cuenta con un lenguaje propio basado en el lenguaje de alto nivel, es decir que se basa en el lenguaje de programación C++.

Este tipo de lenguaje permite trabajar con cadena de caracteres como *string, bit*, números direcciones entre otras. De manera resumida se mostrarán las especificaciones técnicas que cuenta el Arduino UNO:

- Microcontrolador: ATmega328P (8-bit)
- Chip USB: ATmega16U2
- Conector USB: Tipo B
- Voltaje de operación: 5V DC
- Voltaje de alimentación: 6V - 20V DC(7-12V recomendado)
- Pines digitales I/O: 14 (6 salidas PWM)
- Entradas analógicas: 6 (ADC 10-bit)
- Corriente entrada/salida por pin: 40mA máx.
- Memoria FLASH: 32KB (2KB usados por el Bootloader)
- Memoria SRAM: 2KB
- Memoria EEPROM: 1KB
- Frecuencia de reloj: 16MHz
- Leds indicadores: Power, L(Pin 13), TX y RX
- Diseño compatible con Arduino® Uno R3
- Procedencia: China
- Incluye: Cable USB 30cm
- Dimensiones: 73*53*13 mm
- Peso: 30 gramos

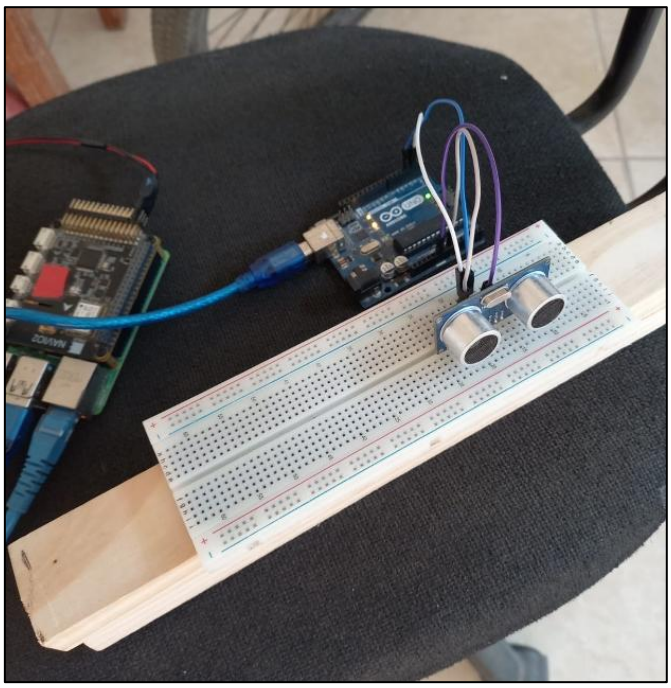
En este microcontrador se va a implementar el sensor ultrasónico HC-SR04, por lo que el pin de alimentación va a estar conectados al pin VC del Arduino, mientras que el GND del sensor a uno de los pines de GND que tiene el Arduino.

Asimismo, los pines de *TRIGGER* y *ECHO* del sensor, van en uno de los 16 pines digitales que cuenta; para esta investigación el pin 3 será el *TRIGGER*, mientras que el pin4 será el *ECHO*. Luego en el Capítulo 4 se verá más detallado la programación que se ha empleado.

El Arduino UNO va a estar conectado a la Raspberry a través de uno de los puertos USB, con el fin de enviar los datos de forma serial hacia la Raspberry, para su posterior uso en el modelo de aprendizaje. En la siguiente figura se puede observar una prueba de la instalación de la placa Navio2 con el Arduino UNO.

Figura 19

Arduino UNO y sensor



En la figura anterior, se ha implementado un prototipo de conexión para verificar su funcionamiento y la recepción de los datos en la Raspberry. Cuando todo se encuentre bien definido se procese a desarrollar el entorno y los modelos de aprendizaje.

Capítulo 4
Desarrollo del entorno y el modelo de entrenamiento

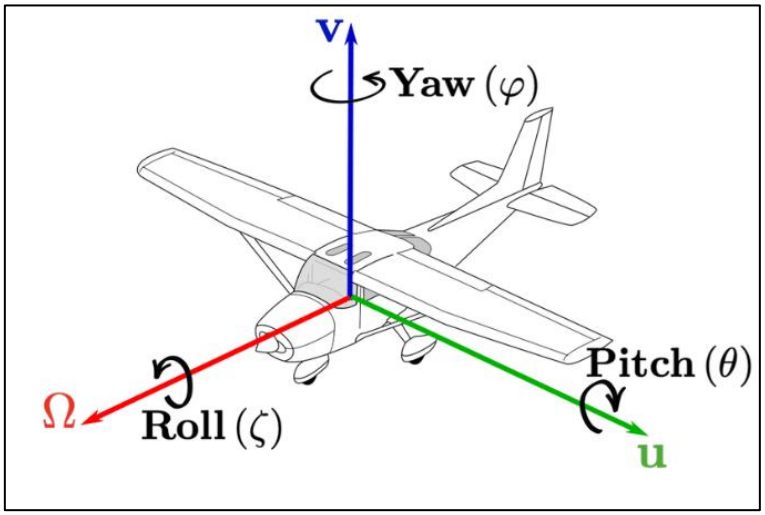
En este capítulo se desarrolla los conceptos y el algoritmo para el modelo de aprendizaje profundo y los nodos con ROS. Asimismo, la programación implementada y las pruebas que se han llevado a cabo para los diferentes modelos que se van a experimentar.

4.1 Ángulos de navegación

Los ángulos de navegación determinan la orientación de un vehículo en un espacio tridimensional, se caracteriza por los ángulos Roll, Pitch y Yaw. El ángulo Roll hace referencia al movimiento horizontal, es decir que, si se tiene un plano tridimensional XYZ, el ángulo Roll indica cuanto ha girado sobre el eje y, el ángulo Pitch sobre cuánto ha girado sobre el eje X y el ángulo Yaw sobre el eje Z, como se indica en la siguiente imagen.

Figura 20

Ángulos de navegación, Roll, Pitch y Yaw



Nota: tomado de (*Figure 10*, s. f.)

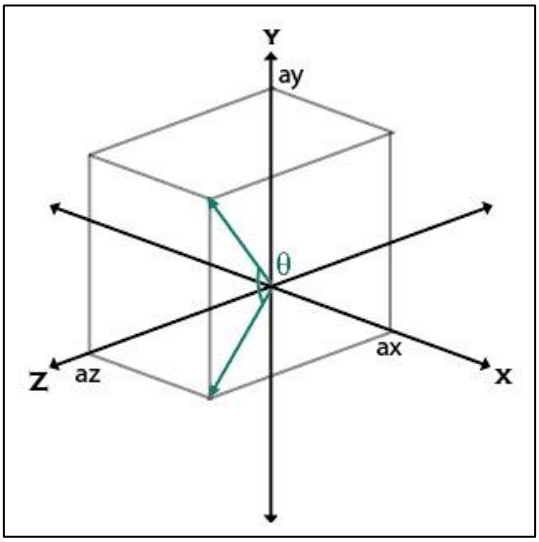
Los ángulos de navegación se determinan por medio de un sensor IMU. Se basa de una expresión matemática simple, cuando se realiza un movimiento, el sensor va a mostrar valores del acelerómetro en sus tres ejes, este se puede posicionar en un punto en el espacio y por geometría analítica, se determina la proyección con respecto al plano ZY o ZX, y con esa proyección determinar el ángulo con respecto al eje X o Y.

4.1.1 *Ángulo Pitch θ*

La Figura 21 se tiene una representación gráfica del ángulo Pitch con los valores del acelerómetro. Estos datos se interpretan como coordenadas de un vector; cuyo ángulo de inclinación respecto al plano XZ da como resultado el Pitch representado por símbolo θ.

Figura 21

Representación de ángulo de Pitch



Para ello, se aplica algebra vectorial, se determina el módulo entre plano XZ con los valores que da el acelerómetro, para luego aplicar el arco tangente con el dato del acelerómetro en la coordenada “y”, por lo que el ángulo Pitch a través de la siguiente expresión matemática.

$$\theta_0 = \tan^{-1}\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right) \qquad \text{Ecuación 15}$$

Sin embargo, esta expresión requiere de un filtro para tratar de disminuir el ruido de la señal, es por eso por lo que se utiliza los valores del giroscopio, donde permite corregir el valor del ángulo. Al ángulo que se ha medido anteriormente se le añade el valor del giroscopio con respecto a la coordenada x y como este tiene como unidades de velocidad angular (°/s) por lo que se requiere de una variación de tiempo para obtener el mismo valor de unidades. Dando como resultado la siguiente expresión matemática.

$$\theta = \theta_0 + \omega_x \Delta t \qquad \text{Ecuación 16}$$

Donde:

- θ, representa el ángulo pitch,
- θ_0, representa el ángulo pitch previamente calculado,
- ω_x, representa el valor del giroscopio en el eje X,
- Δt, representa la variación del tiempo.

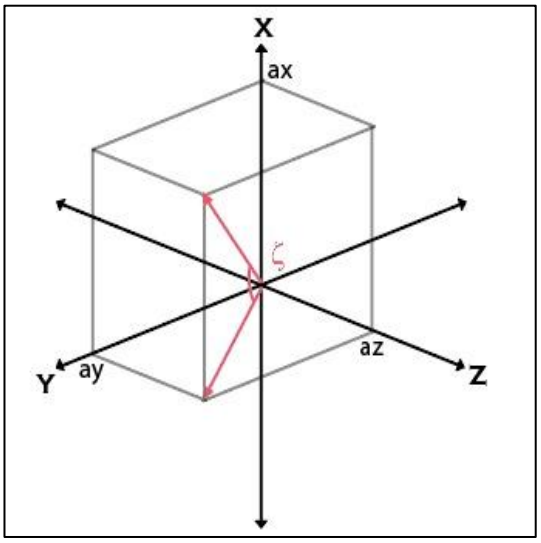
4.1.2 *Ángulo Roll ζ*

Teniendo en cuenta lo aplicado anteriormente, el ángulo Roll se realiza el mismo razonamiento, en este caso el ángulo del vector se determinará por la inclinación de este hacia

el plano YZ. La siguiente imagen muestra como se ve la representación gráfica del vector cuyo ángulo viene determinado por el siguiente símbolo ξ.

Figura 22

Representación del ángulo de Roll



Teniendo en cuenta se aplica la Ecuación 15 y Ecuación 16 modificada, se adapta a las coordenadas que se tiene para este caso.

$$\zeta_0 = \tan^{-1}\left(\frac{a_x}{\sqrt{a_y^2 + a_z^2}}\right) \qquad \text{Ecuación 17}$$

$$\zeta = \zeta_0 + \omega_y \Delta t \qquad \text{Ecuación 18}$$

Donde:

- ζ, representa el ángulo roll,
- ζ_0, representa el ángulo roll previamente calculado,
- ω_y, representa el valor del giroscopio en el eje Y,
- Δt, representa la variación del tiempo.

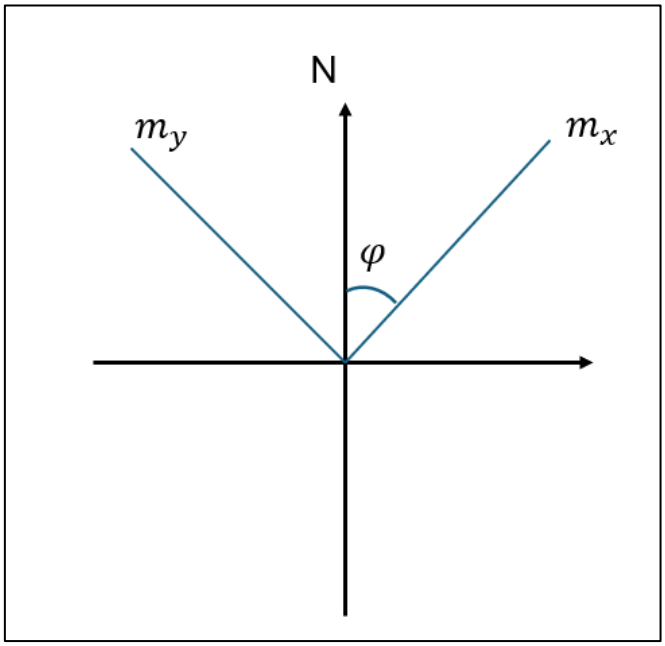
4.1.3 *Ángulo Yaw φ*

El ángulo Yaw va a indicar el ángulo de inclinación con respecto campo magnético de la tierra y toma de referencia el polo Norte; es por lo que se emplea un magnetómetro para obtener dicho valor; para ello se debe tener en cuenta las coordenadas X y Y que ofrece el sensor.

En la siguiente figura se va a mostrar un caso hipotético de ubicación del sensor en donde se representa los ejes que da el sensor con sus respectivas amplitudes y el ángulo que se desea calcular el cual viene representado por φ.

Figura 23

Representación del ángulo de Yaw



Por lo tanto, el ángulo de inclinación se determina a través del arco tangente de las coordenadas X y Y que otorga el sensor, lo que queda en la siguiente expresión matemática.

$$\varphi = \tan^{-1}\left(\frac{m_y}{m_x}\right) \qquad \text{Ecuación 19}$$

Estos tres ángulos van a ser parte del modelo de aprendizaje, servirá como orientación para el prototipo, y que aportará para la recompensa que va a estar obteniendo el prototipo.

4.2 Programación de los sensores

En esta investigación, como se ha mencionado anteriormente, se va a contar con el uso de tres sensores, los cuales son el MPU9250 que va a otorgar los valores del acelerómetro y el giroscopio; el LSM9DS1 que permitirá dar valores del magnetómetro, y un sensor ultrasónico HR el cual está conectado con un Arduino Uno y se enviaran los datos por medio de serial. La finalidad de los sensores es para tener los ángulos Roll, Pitch y Yaw, que se utilizan para tener una orientación del prototipo.

El primero se encargará del ángulo Roll y Pitch, mientras que el segundo sensor se encargará del ángulo Yaw. Por último, el sensor ultrasónico, se encargará de terminar la altura que hay entre el la parte inferior de la estructura del prototipo y el suelo, lo que permite determinar cómo deben estar configuradas las extremidades de este.

4.2.1 *Sensores MPU9250 y LSM9DS1*

En el caso de la programación de los sensores MPU9250 y LSM9DS1, cuenta con librerías las cuales llaman a los datos que perciben estos sensores. Estos datos que recibe son valores de aceleración en los ejes "X", "Y" y "Z", de igual forma, con los valores del giroscopio y el magnetómetro.

Para ello se debe llamar a la librería mpu9250 y lsm9ds1, la cual nos va a dar los valores del giroscopio, el acelerómetro, la primera librería y el magnetómetro la segunda. Para insertar en la programación se debe de llamar a las clases MPU9250 y a la LSM9DS1, por lo cual se va a implementar con la variable imu e imu2 para cada uno de ellos respectivamente. Cuando se ha inicializado estos objetos se llama al método getMotion9 tanto para imu como para imu2. Este método nos da como resultado tres *arrays* de longitud (3,), los cuales vamos a asignar tres variables; m9a, m9g y m9m, para el caso de imu; y l9a, l9g y l9m, para el caso del imu2.

Sin embargo, el array m9m no se va a utilizar, debido a que el sensor MPU9250 no muestra valores de magnetómetro. Mientras que en el caso de los *arrays* l9a y l9g, no se va a utilizar, debido a que el sensor LSM9DS1 no muestra los valores del acelerómetro ni el giroscopio.

Para cada valor de eje, se va a asignar un elemento del array para ax será m9a[0], ay será m9a[1] y para el az será m9a[2], del mismo modo para los valores del giroscopio m9g, dando los valores gx, gy y gz. En el caso del magnetómetro, se utilizará el l9m, con sus respectivos elementos que nos dan el mx, my y mz.

Para el cálculo del ángulo Roll y Pitch, se ha optado por la creación de funciones para optimizar líneas de código. Se crea una función llamada ángulos, la cual pida tres argumentos los cuáles serán las aceleraciones en los ejes y retorne el valor del ángulo en sexagesimales.

En base a la ecuación Ecuación 15 para el cálculo del roll y el pitch visto anteriormente se plantea las siguientes líneas de Código.

```
r = math.sqrt(b**2 + c**2);
d = math.atan2(a,r)*180.0/math.pi
```

Las líneas de códigos anteriores van a ir dentro de una función llamada "ángulo" donde se retorna el valor de la variable d.

Para precisión en la lectura de los datos se agrega un filtro, para ello tomamos como base la Ecuación 16 la cual se expresa en la siguiente línea de código.

```
0.98*(r_ang_prev + gy*dt) + 0.02*roll.
```

De igual forma se tiene que hacer para el ángulo pitch. En el caso del "dt" que se observa se implementa con la librería time, para que nos dé el momento cuando inicia, y se utiliza dos veces para tener una variación del tiempo como se muestra en la siguiente línea de código.

```
tiempo_prev = time.time()
dt = (time.time() - tiempo_prev)
```

Como esto se encuentra dentro de un bucle While, este va a estar en constante actualización de los valores. Finalmente, se va a imprimir los datos; se puede hacer esto por medio de función print, o se puede utilizar el log.info de ROS.

4.2.2 *Visualización del comportamiento de los sensores MPU9250 y LSM9DS1*

El programa *MobaXterm* se utiliza para conectarse desde un ordenador hacia la Navio2, por medio de una red Wifi; este programa da la opción de poder grabar los datos del terminal, con el fin extraer los datos del sensor IMU, y poder analizar su tiempo de reacción.

Para obtener los datos, se hacen movimientos a 90 y -90 grados respecto a la placa, tanto para la derecha como para la izquierda para determinar el ángulo Roll; y para arriba y abajo, para el ángulo Pitch. Para el caso del ángulo Roll, se hacen las pruebas partiendo del ángulo cero, luego pasará a 90 grado, regresa a su posición inicial, luego pasa al ángulo de -90 grados, y finalmente, se repite el mismo movimiento, de 0 a -90. Entre cambio de ángulo de 90 a cero, se va a dar un tiempo.

En el caso del ángulo Pitch, las pruebas se hacen partiendo desde el ángulo 0 y luego se cambia a 90 grados y retomar al ángulo inicial, luego se cambia desde -90 a 0, y finalmente, se de 90 a 0. Para este ángulo Pitch, ni bien se llegue al ángulo determinado, se cambia a otro ángulo, para evaluar su tiempo de reacción.

Terminada las pruebas se exportan los valores del ángulo Roll y Pitch en formato xls. Implementando Python se grafican los datos obtenidos. Las librerías que se emplean son Pandas, Matplotlib y Numpy. En Google Colab se sube los archivos a la nube y trabajarlos desde ahí. Se inserta la siguiente línea de código para la extracción de archivos de la nube:

```
from google.colab import drivedrive.mount('/content/drive')
```

Se importa las librerías que se mencionan anteriormente y se asigna una variable que va a contener el dataframe de los datos del sensor.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataframe=pd.read_excel("/content/drive/MyDrive/TESIS/ROLL_GRAPH_5.
xlsx")
dataframe.head(7)
```

La última línea de código `dataframe.head(7)` muestra en una tabla a los valores del ángulo roll y el tiempo en segundos. Solo se van a visualizar los 7 primeros datos, con la función head, se puede configurar la cantidad de datos que se desee observar dentro del *dataframe*. Solo como modo de prueba se ha puesto la cantidad de datos que se ha mencionado anteriormente, dando como resultado la Tabla 3.

Tabla 3
Tabla de ángulo Roll
dataframe.head (7)

	Tiempo	Ángulo roll
0	1	0.009
1	2	0.024
2	3	0.04
3	4	0.049
4	5	0.056
5	6	0.067
6	7	0.081

Se debe analizar si es que dentro del *dataframe*, existe algún elemento que sea un objeto y que no sea un *float* o *int*, para ello se puede implementar el siguiente comando:

```
dataframe.info()
```

El cual da como resultado información de la tabla que se ha ingresado, como se observa a continuación.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4805 entries, 0 to 4804
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   tiempo  4805 non-null   int64
 1   angulo  4805 non-null   object
dtypes: int64(1), object(1)
```

Para este caso ha dado como resultado que la columna de los ángulos es del tipo *Object*, por lo que se va a tener problemas al momento de graficar; por lo tanto, se tiene cambiar a tipo *float*, esto se hace a través del siguiente comando:

```
dataframe["angulo"] = pd.to_numeric(dataframe["angulo"],
errors="coerce")
```

Con el comando anterior, se debería corregir los datos, por lo que para verificar nuevamente se ingresará el comando *info*, para evaluar cómo se encuentra el *dataframe* que se está trabajando, el cual da como resultado lo siguiente.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4805 entries, 0 to 4804
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   tiempo  4805 non-null   int64
 1   angulo  4804 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 75.2 KB
```

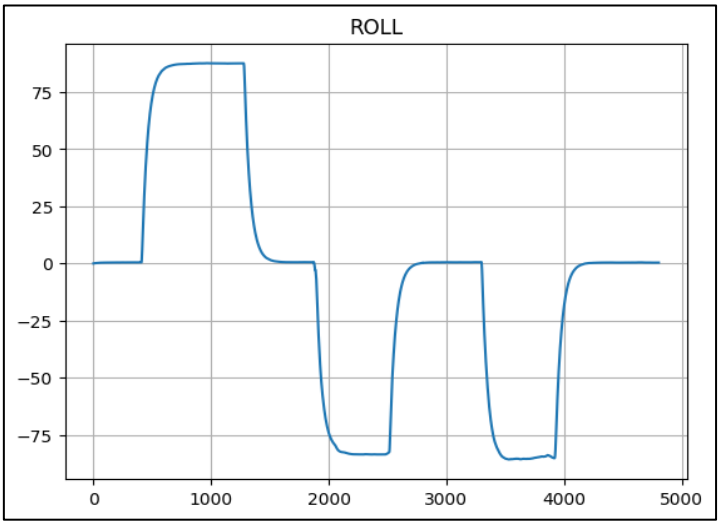
Como se puede observar ahora la columna de los ángulos roll son del tipo float6, por lo tanto, ya es posible graficar estos valores, para ello, se emplea a través de la siguiente línea de código:

```
x = dataframe['tiempo']
y = dataframe['angulo']
fig, ax = plt.subplots()
ax.plot(x,y)
plt.title("ROLL")
plt.grid()
plt.show()
```

Con esta línea de código, se asigna a la columna “tiempo” la variable “x”, mientras que la columna “ángulo” como la variable “y” dando como resultado el siguiente grafico que se observa a continuación.

Figura 24

Sensor IMU con el ángulo de Roll



Como se observa en el gráfico anterior y con la ayuda del filtro, se obtiene resultados limpios con los que se puede trabajar, no hay presencia de ruido y el tiempo de estabilidad es relativamente corto, adecuados para poder implementar una función de transferencia. De igual forma, se hace un análisis para el ángulo pitch, se importa el archivo xls por pandas.

```
dataframe=pd.read_excel("/content/drive/MyDrive/TESIS/PITCH_GRAPH.xlsx")
```

Se evalúa, si es que existe un dato vacío o que tenga otro tipo de formato por medio de las funciones que antes se han implementado; posteriormente, se visualizará en una tabla, implementado el comando head (7) de la librería Pandas, dando como resultado la siguiente tabla.

Tabla 4

Tabla de ángulo Pitch

dataframe.head (7)

	Tiempo	Ángulo Pitch
0	**1**	**0.019**
1	**2**	**0.034**
2	**3**	**0.061**
3	**4**	**0.09**
4	**5**	**0.114**
5	**6**	**0.136**
6	**7**	**0.162**

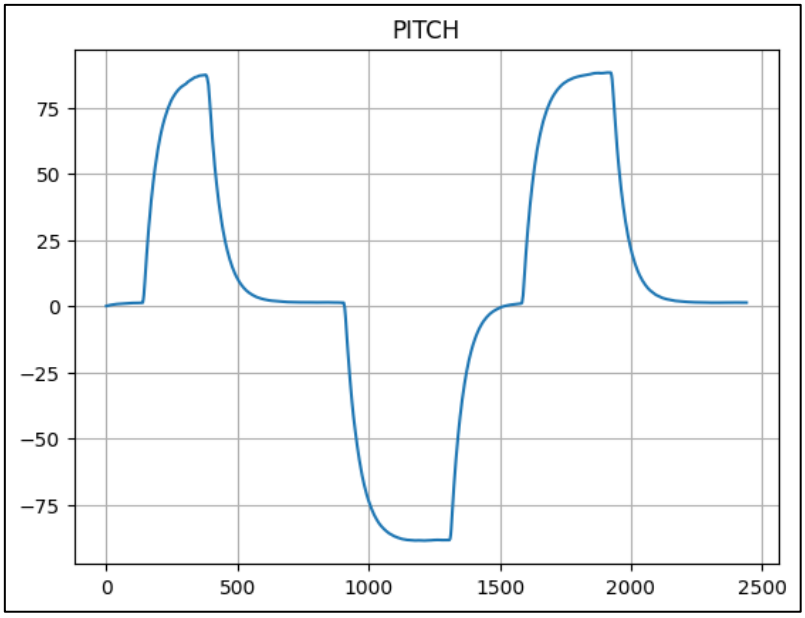
Cuando se ha recolectado los datos para este ángulo, se han asignado las variables “y2” y “x2”, para el ángulo Pitch y Tiempo respectivamente, con el fin de poder diferenciarlos de los datos del ángulo Roll con los que se ha trabajado anteriormente; luego se grafica verifica si es que los datos se encuentran sean del tipo adecuado para graficar.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2440 entries, 0 to 2439
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   tiempo  2440 non-null   int64
 1   pitch   2440 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 38.2 KB
```

Como se ha observado anteriormente, no tiene ningún dato tipo object, por lo que se implementa las mismas funciones con las que se ha graficado el ángulo anterior, obteniendo como resultado el siguiente gráfico.

Figura 25

Gráfico del sensor IMU con el ángulo de Pitch



Similar a lo que se ha obtenido en el gráfico del ángulo Roll, el filtro a porta a que los datos de los ángulos sean más precisos y no existe ruido o interferencia siendo adecuado para desarrollar el sistema de control por *Deep Reforcement Learning*.

4.3 Cinemática inversa

Uno de los principios que se aborda en la robótica en general, es la cinemática inversa. Se basa en conceptos fundamentales de geometría, como el teorema de cosenos, trigonometría, entre otros; para determinar los ángulos que deben estar las articulaciones, que luego podrán aplicarse a los servomotores.

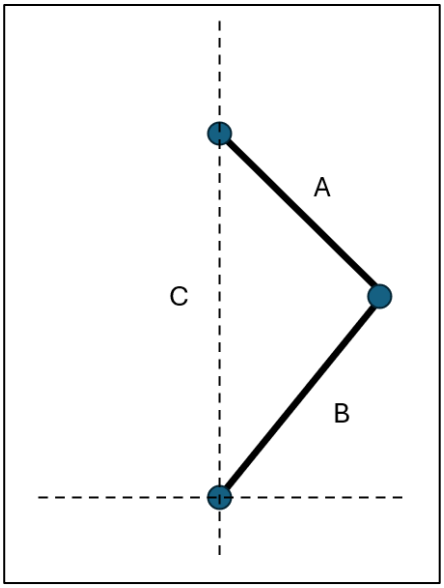
En esta investigación se tiene en cuenta tres situaciones donde se va a implementar la cinemática inversa, una de ellas en las extremidades para que el robot pueda ponerse de pie, la otra es cuando la altura en las extremidades sea diferente y la última es como se va a trasladar el robot. Para los tres casos, se plasmará las ecuaciones necesarias que se han tomado en cuenta para dar con los ángulos.

4.3.1 *Cinemática inversa en las extremidades*

Para determinar los valores de un ángulo entre articulación y articulación, se debe ubicar un punto de referencia, y asignar coordenadas. En el caso de las extremidades del prototipo, se va a tomar toda la estructura junta, por lo tanto, el diagrama va a tener la siguiente forma, como se observa en la siguiente figura.

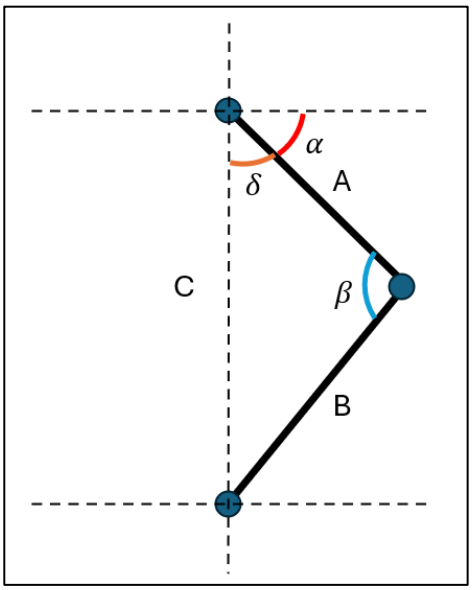
Figura 26

Diagrama de extremidades



Las variables A y B son las longitudes de las extremidades; mientras que el valor C va a ser referencia de la altura a la que va a estar el prototipo del suelo al chasis. Como se observa, esto tiene forma de un triángulo, por lo que se puede asignar los ángulos, como se observa en la siguiente figura.

Figura 27
Diagrama de extremidades con ángulos asignados



Por lo tanto, con la figura anterior, se aplicará el teorema de cosenos, teniendo la siguiente expresión matemática, para poder hallar el ángulo δ, y β.

$$C^2 = A^2 + B^2 - 2AB\cos\beta$$ Ecuación 20

Por lo tanto, se puede despejar el valor de ángulo β, la cual da como resultado la siguiente expresión.

$$\beta = \cos^{-1}\left(\frac{A^2 + B^2 - C^2}{2AB}\right)$$ Ecuación 21

De igual forma, se puede hacer para determinar el valor del ángulo δ.

$$\delta = \cos^{-1}\left(\frac{A^2 + C^2 - B^2}{2AC}\right)$$ Ecuación 22

Con el dato del ángulo δ, se pueda calcular el ángulo α, debido a que es el complemento del ángulo anterior, por lo tanto, quedaría la siguiente expresión.

$$\alpha = 90 - \delta$$ Ecuación 23

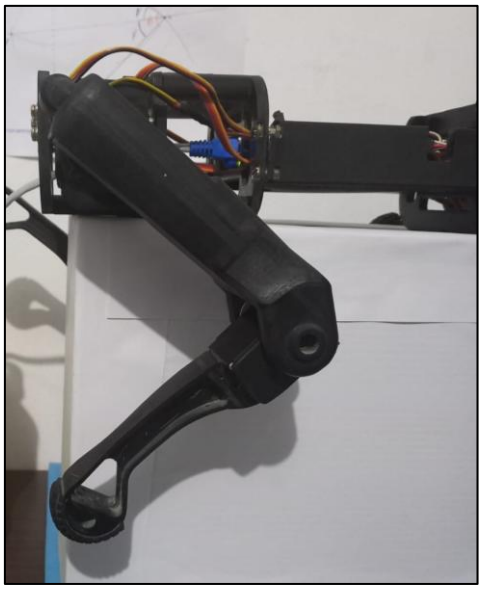
Estas ecuaciones, aportarán para la programación de los servomotores de las extremidades, que servirán para el movimiento que estos puedan tener. El fin de realizar esta cinemática es poder tener un control sincronizado de los movimientos de las extremidades.

En la siguiente figura, se observa en un momento adecuado una de las extremidades posicionada de acuerdo con cómo se ha venido desarrollando; se debe tener en cuenta que el

movimiento de las extremidades debe estar acorde a ese posicionamiento, caso contrario no se puede desplazar de manera adecuada.

Figura 28

Representación en físico



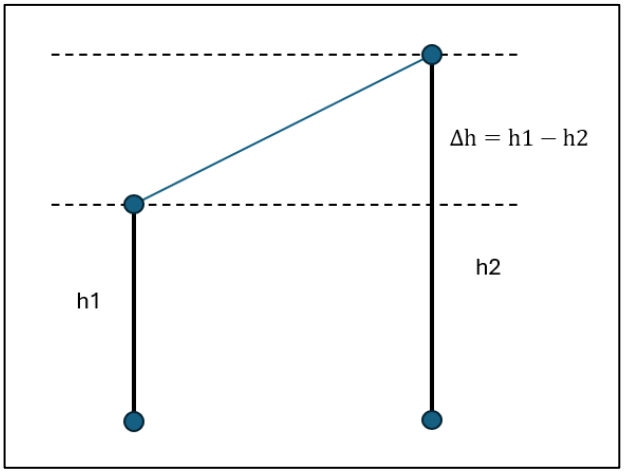
4.3.2 *Cinemática inversa posteriores y frontales*

Para la parte frontal y posterior de las extremidades se debe tener en cuenta otra cinemática, esta cinemática es exclusivamente para los servomotores de tipo 1, es decir los servomotores EIF1, EDF1, EIP1 y EDP1.

Debido a que, si una de las extremidades tiene una altura diferente, el servomotor tiene que estar acorde a esos movimientos. Para ello, se debe tener en cuenta una situación hipotética, que se observa en la siguiente figura:

Figura 29

Diagrama de extremidades para roll-pitch

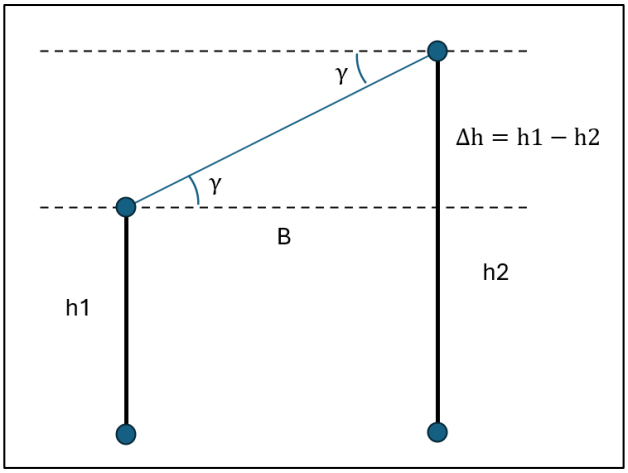


La extremidad de la izquierda tiene una altura diferente a la extremidad de la derecha, lo cual va a tener una diferencia de altura, lo que proporciona un ángulo respecto a la horizontal.

Este ángulo, por principio de geometría, va a ser mismo ángulo en la parte de la otra extremidad, esto se logra trazando una paralela a la base, como se observa en la siguiente figura.

Figura 30

Diagrama de extremidades para roll-pitch con asignación de ángulos



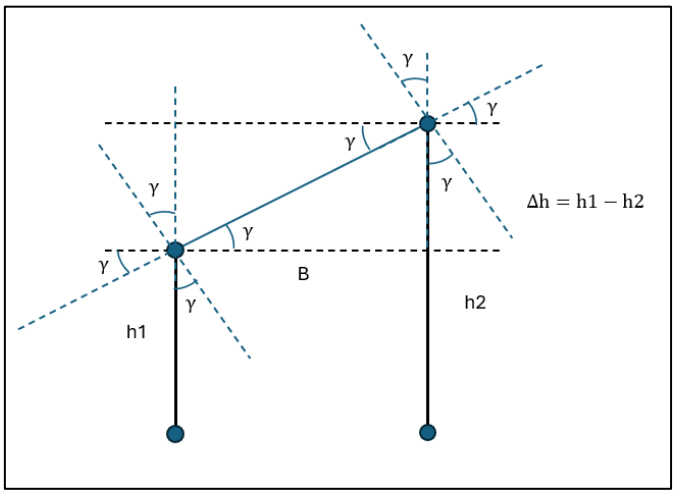
A este ángulo lo denominaremos como γ, y representa cuanto ha cambiado la altura de una extremidad respecto a la otra. Por lo tanto, se puede calcular dicho ángulo aplicando el arco tangente de la diferencia de las alturas de las extremidades entre la base, es decir la distancia de una extremidad y la otra, dando la siguiente expresión matemática.

$$\gamma = \tan^{-1}\left(\frac{h2 - h1}{B}\right) \qquad \text{Ecuación 24}$$

Esta ecuación también es aplicable cuando $h2$ sea menor que $h1$, lo cual aporta para que el ángulo sea positivo o negativo. Luego se emplea una prolongación de las rectas desde la recta de los puntos donde termina la altura de cada extremidad, así como también la perpendicular de esas rectas por lo que por geometría el ángulo va a estar ubicado en los 4 cuadrantes como se puede ver en la siguiente figura.

Figura 31

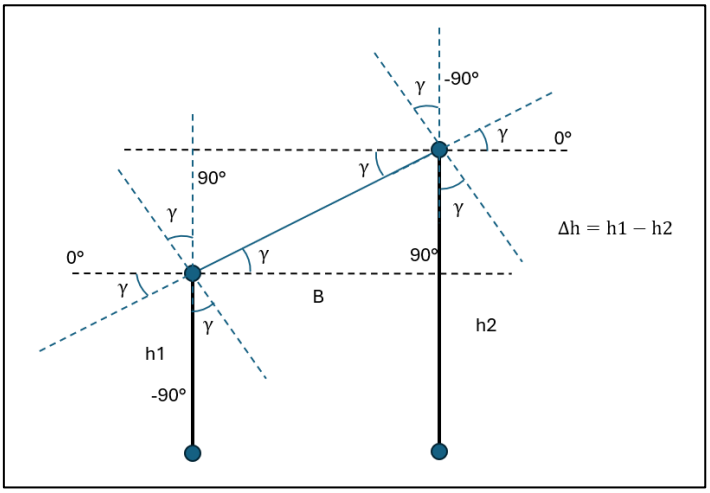
Diagrama de extremidades para roll-pitch con asignación de ángulos extendidos



Este arreglo, nos permite ver cómo sería el comportamiento de la articulación, y los ángulos que intervienen. Teniendo en cuenta esto, se puede asignar la orientación de los cuadrantes como se observa en la siguiente figura.

Figura 32

Diagrama de extremidades para roll-pitch con orientación de ángulos



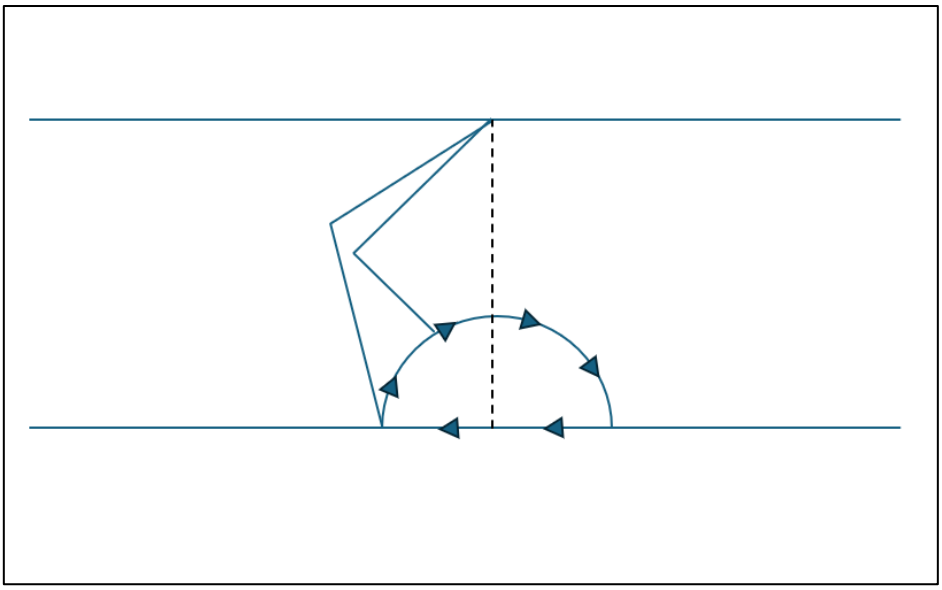
Lo que se ha observado en la figura anterior, aporta a que los servomotores de esa articulación se puedan configurar, por la orientación que este está haciendo. Todo esto se ve en la programación de los servomotores.

Figura 33
Representación en físico

4.3.3 *Cinemática inversa de traslación*

Para que el prototipo pueda realizar una traslación sobre la superficie, la parte inferior debe realizar un movimiento en forma de una semicircunferencia, como se observa en la siguiente imagen.

Figura 34
Cinemática de traslación

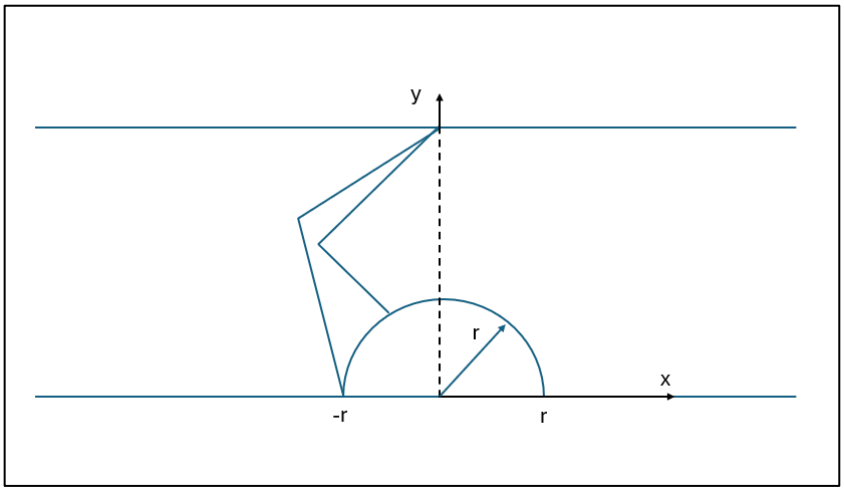


El movimiento debe seguir las flechas, esto permite que pueda realizar una traslación hacia adelante. La primera parte es para que pueda levantar su punto de apoyo y la segunda parte es para que pueda arrastrar y se pueda mover.

Se toma un punto de referencia, en este caso, el centro de la semicircunferencia; a partir de ello se traza un eje “y” y “x” los cuales aportan para definir el movimiento que va a realizar. La semicircunferencia tiene como radio r, por lo que puede ser asignado cualquier valor. En la siguiente figura se observa la geometría expresada.

Figura 35

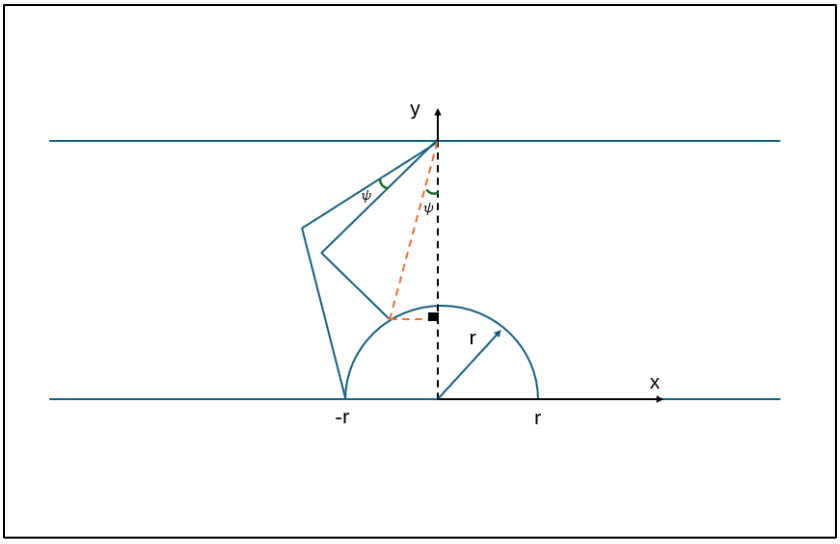
Cinemática de traslación con punto de referencia



Con ello se puede obtener el ángulo ψ que es la inclinación con respecto al eje “y”, siendo este congruente al ángulo cuando la extremidad este sobre la superficie, como se observa en la siguiente figura.

Figura 36

Asignación de ángulos en las extremidades al trasladarse.



Teniendo en cuenta esta cinética, se determinará cuanto ha ido variando la altura a lo largo del recorrido de la semicircunferencia, por lo que se tomará en cuenta la siguiente

ecuación de la circunferencia, que representa la circunferencia cuyo centro se encuentra ubicado en punto de referencia.

$$x^2 + y^2 = r^2$$ Ecuación 25

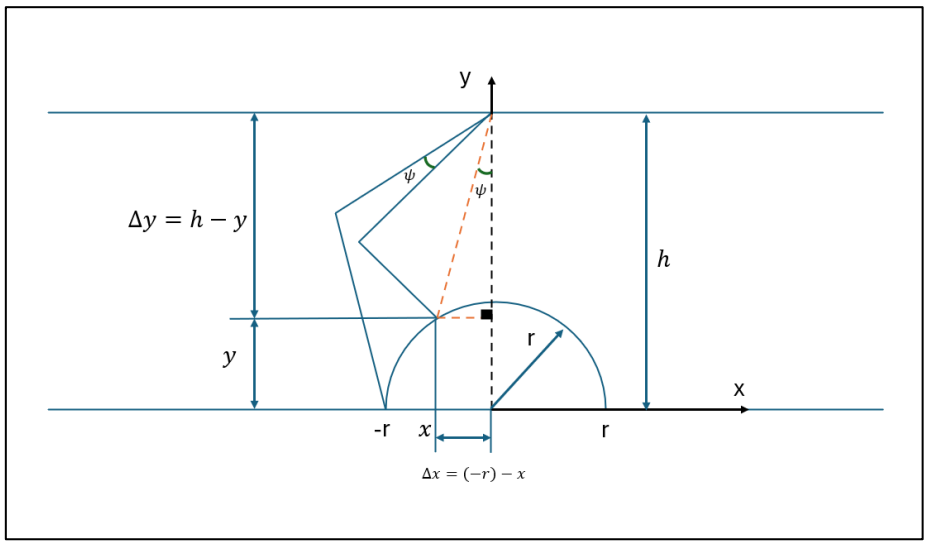
Como se había mencionado, se debe determinar la altura, por lo que de la Ecuación *25*, se va a tener en cuenta la variable "y"; y solo los valores positivos, dando la siguiente ecuación.

$$y = \sqrt{r^2 - x^2}$$ Ecuación 26

Con esta expresión, dicha altura que hay entre la superficie y la parte inferior de la extremidad del robot. En la siguiente figura, se puede apreciar algunas cotas que se deben tener en cuenta para poder calcular el valor de ψ.

Figura 37

Asignación de ángulos en las extremidades al trasladarse acotaciones



Por lo tanto, la expresión que se debe implementar es la tangente del ángulo ψ, cuyos valores de cateto opuesto es Δx y su cateto adyacente es Δy, los que se han indicado en la figura anterior.

$$\tan\psi = \frac{\Delta x}{\Delta y}$$ Ecuación 27

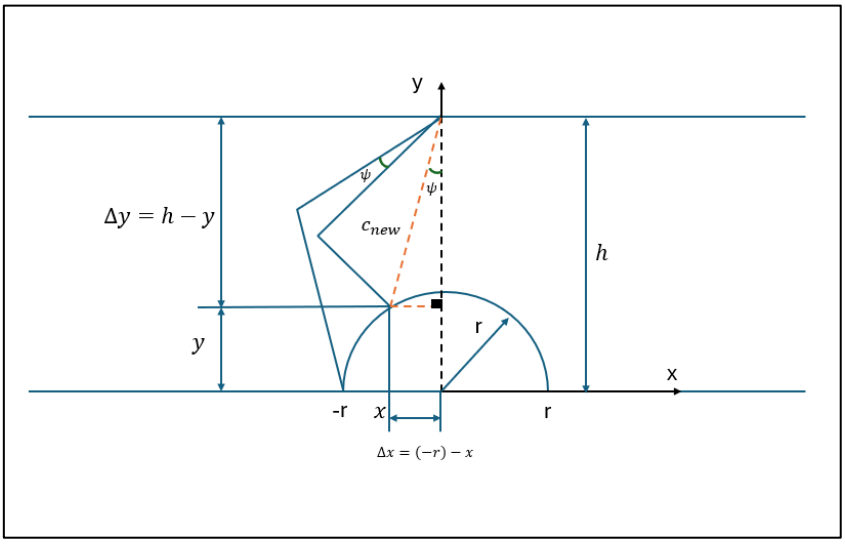
Como la ecuación se determinará la siguiente expresión matemática, para determinar el ángulo ψ.

$$\psi = \tan^{-1}\frac{\Delta x}{\Delta y}$$ Ecuación 28

Cuando se determinado el ángulo ψ, se hallará la nueva distancia que existe entre la parte inferior de la extremidad y la parte superior, el cual se le ha denominado como C_{new}. Para mayor detalle, en la siguiente figura se observa la ubicación de la distancia C_{new}.

Figura 38

Cnew ubicación



Por lo tanto, se calcula aplicando el coseno de dicho ángulo cuyo cateto adyacente es Δy y su hipotenusa es C_{new}, dejando la siguiente expresión.

$$\cos\psi = \frac{\Delta y}{C_{new}} \qquad \text{Ecuación 29}$$

Con eso se despeja el valor de C_{new}, dejando la siguiente expresión:

$$C_{new} = \frac{\Delta y}{\cos\psi} \qquad \text{Ecuación 30}$$

De igual forma, el ángulo ψ influye en el ángulo α, como se observa en la siguiente expresión matemática.

$$\alpha = 90 - \delta - \psi \qquad \text{Ecuación 31}$$

4.4 Programación de los servomotores

Navio2, cuenta en su repertorio con ejemplos utilices, tales como controladores de servomotores, giroscopio, acelerómetro, LED, etc. Estos ejemplos se encuentran en lenguaje de programación Python y C++, lo cual Navio2 se vuelve versátil en su uso. Para proyecto de investigación se utilizará el lenguaje de programación Python.

4.4.1 Programación de servomotores modelos estáticos

La estructura base de la programación de los servomotores se encuentra en importar las librerías navio.pwm y navio.util, con esto se llamará a la clase `check_apm()`, después se

asignan las variables que corresponden a la ubicación del servomotor en la Navio2, así como también el ancho de pulso el cual corresponde con el ángulo de movimiento del servomotor.

Luego con la función *With*, se asociará a una variable de la clase PWM la cual servirá para llamar al método `set_duty_cycle()`, cuyo argumento es el ancho de pulso que se desee.

Para esta investigación, la estructura se tiene que modificar, puesto que ahora se requiere de 12 servomotores para el prototipo, lo que implica que se tiene que añadir 12 variables que correspondan a la entra de los servomotores, así mismo, se debe evaluar el ángulo máximo y mínimo que puede soportar en la estructura. En la siguiente tabla se puede observar todas las configuraciones que tiene los servomotores en el prototipo.

Tabla 5

Análisis de los servomotores y sus mínimos y máximos

NAME:	PIN	CODE	ANG_MIN (°)	CODE ANG_min (ms)	ANG_MAX (°)	CODE ANG_max (ms)
EFD 1	13	12	90.00	2.500	-90.0	0.500
EFD 2	12	11	90.00	2.000	-45.0	0.500
EFD 3	11	10	41.21	2.500	180.0	1.000
EFI 1	1	0	0.00	0.500	-90.0	1.500
EFI 2	2	1	90.00	0.500	-45.0	2.000
EFI 3	3	2	41.21	1.000	180.0	2.500
EPD 1	8	7	45.00	0.500	-45.0	2.500
EPD 2	9	8	90.00	2.000	-45.0	0.500
EPD 3	10	9	41.21	2.500	180.0	1.000
EPI 1	6	5	90.00	2.500	-90.0	0.500
EPI 2	5	4	90.00	1.250	-45.0	2.500
EPI 3	4	3	41.21	1.000	180.0	2.500

En la tabla anterior, se observa unos ángulos mínimos y máximos con su respectivo valor en *dutycyle*. Esos datos se han obtenido con respeto a la ubicación que tiene el servomotor en el prototipo.

Con estos datos, y lo que se analizó anteriormente con la Cinemática inversa, se debe realizar una interpolación entre los ángulos que da la cinemática y los valores en ms del *dutycycle*. El método de interpolación es una interpolación lineal, la cual aplica la siguiente expresión matemática.

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \qquad \text{Ecuación 32}$$

Siendo "x" los valores de entrada de los ángulos en sexagesimales; mientras que la "y", seria los valores en ms para el *dutycycle*. Para hacerlo de manera automática, se crea una función la cual, se reciba como argumento el ángulo y retorne el valor en ms, como se va a mostrar a continuación.

```
def inter_EIF1(angulo):
    y1 = 3.25
    x1 = -90.00
    y2 = 1.25
    x2 = 90.00
    y_int = ((y2 -y1)/(x2-x1))*(angulo-x1)+y1
    return y_int
```

De esta manera se debe hacer para todos los servomotores, por lo que se va a tener 12 funciones donde cada una de ellas va a contar con los parámetros mínimos y máximos como se indica en el código anterior basados en la Tabla 5.

Los ángulos se obtienen por medio de una función, en el caso de calcular el ángulo α la función recibe de entrada la altura, dentro de esta función ya estarán las medidas de las extremidades, por lo que se podrá verá en el siguiente código se basa en la Ecuación 21.

```
def alfa_ang(h):
    alf_rad = math.pi / 2 - math.acos((11.0**2 - 13.3**2 + h**2) /
(2 * 11.0 * h))
    alf = alf_rad * 180 / math.pi
    return alf
```

En el caso de la función del ángulo beta, se hace de la misma manera, recibirá como argumento la altura y retornará el ángulo beta.

```
def bet_ang(h):
    bet_rad = math.acos((11.0**2 + 13.3**2 - h**2) / (2 * 11.1 *
13.3))
    bet = bet_rad * 180 / math.pi - 36.9
    return bet
```

En el caso del ángulo theta, recibirá como argumento dos alturas, dentro de esta función se dará el valor de la base y al final se retornará el ángulo.

```
def thet_ang(h1,h2):
    thet_rad = math.atan((h1-h2)/10)
    thet = thet_rad * 180 / math.pi
    return thet
```

Con estos códigos ya se obtendría los valores los ángulos que, junto con la interpolación anterior mencionada, se enviará al objeto set_duty_cycle().

4.4.2 Programación de servomotores modelos dinámicos

Para la simplificación en el nodo servo, se determinarán funciones para sección, es decir para la asignación de los puertos de los servomotores, así como también en la frecuencia con la que trabajan. Primero se llama a las variables de las articulaciones como se observa en la siguiente línea de código.

```
SERVO_FREQ = 50

A1, A2, A3 = 0, 1, 2
B1, B2, B3 = 12, 11, 10
C1, C2, C3 = 7, 8, 9
D1, D2, D3 = 5, 4, 3
```

Por lo que, para poder llamar a cada uno de los servomotores, se hará en un bucle *for* que sea capaz de llamar a cada una de las variables y que les asigne la inicialización, la frecuencia y la disponibilidad, como se observa en las siguientes líneas de código.

```
PWM_OUTPUTS = [A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3]
pwms = []

for output in PWM_OUTPUTS:
    pwm = navio.pwm.PWM(output)
    pwm.initialize()
    pwm.set_period(SERVO_FREQ)
    pwm.enable()
    pwms.append(pwm)
```

Cada uno de los servomotores tiene una interpolación, como se ha mencionado anteriormente, por lo que se puede crear una función general donde reciba como argumentos el servomotor, el ángulo y la función de interpolación. Donde dentro de ella se encuentre el

set_duty_cycle, que recibirá los valores de los ángulos, pase por la función de interpolación y ese lo pase por el método antes mencionado para su ejecución. Como se observa en las siguientes líneas de código.

```
def set_servo_angle(pwm, angle, interpolation_function):
    duty_cycle = interpolation_function(angle)
    pwm.set_duty_cycle(duty_cycle)
```

Cuando se tenga esta función diseñada, se va a tener otra función la cual reciba los ángulos y que estos se asignen a cada uno de los servomotores. Es decir, una sola función donde contarán con cada uno de los servomotores y los ángulos, en este caso tendrá como argumento cuatro ángulos, los cuales son para las articulaciones (2D,3D) y (2I, 3I). Estos van a estar conforme a la función que se ha diseñado anteriormente. Esto se ve evidenciado en las siguientes líneas de código.

```
def caminar_paso(angle1I, angle2I,angle1D, angle2D):


    set_servo_angle(pwms[0], 0.0, inter_EIF1)
    set_servo_angle(pwms[1], angle1I, inter_EIF2)
    set_servo_angle(pwms[2], angle2I, inter_EIF3)


    set_servo_angle(pwms[3], -10.0, inter_EDF1)
    set_servo_angle(pwms[4], angle1D, inter_EDF2)
    set_servo_angle(pwms[5], angle2D, inter_EDF3)


    set_servo_angle(pwms[6], 0.0, inter_EDP1)
    set_servo_angle(pwms[7], angle1I, inter_EDP2)
    set_servo_angle(pwms[8], angle2I, inter_EDP3)


    set_servo_angle(pwms[9], 0.0, inter_EIP1)
    set_servo_angle(pwms[10], angle1D, inter_EIP2)
    set_servo_angle(pwms[11], angle2D, inter_EIP3)
```

Por último, se va a diseñar otra función cuyo argumento de entrada sea la altura que va a recibir; dentro de esta función se va a contener las funciones que se han diseñado anteriormente, así como también la forma en cómo va cambiando el valor de la altura cuando realizar el movimiento semicircular.

Para ello, se va a utilizar dos bucles for, el cual el primero va a ser parte de la región semicircular y como se ha observado en la Figura 35 desde un valor de -r hasta r. Mientras que el segundo bucle *for* desde r a -r.

El primer bucle *for* va a estar en un rango de -40 a 40, con un paso de 1; en este caso 40 el valor del radio del círculo multiplicado por 10, y como va variando de uno en uno, cuando pase por la iteración se tendrá que dividir entre 10 dando valores de "x" cada 0.1. Asimismo, cuando va variando el valor de x, este va a entrar dentro de una función llamada c_{new}, la cual se basa en la Ecuación 30 y con ello dar valores de alfa y beta que ingresarán a la función caminar_paso. En el segundo bucle *for* tendrá las mismas características que el primero, a diferencia que este va a ir retrocediendo, es decir tendrá un rango de 40 a -40, con un paso de -1. Como se observa en las siguientes líneas de código.

```
def caminar_semicirculo(h):
    for i in range(-40,40,1):
        x = i/10
        x2 = -i/10
        c, psi = c_new(h,x)
        alfa = alfa_ang(c,psi)
        beta = bet_ang(c)
        c2, psi2 = c_new2(h,x2)
        alfa2 = alfa_ang(c2,psi2)
        beta2 = bet_ang(c2)
        caminar_paso(alfa,beta,alfa2,beta2)
        time.sleep(0.01)

    time.sleep(0.5)

    for i in range(40,-40,-1):
        x = -i/10
        x2 = i/10
        c, psi = c_new(h,x)
        alfa = alfa_ang(c,psi)
        beta = bet_ang(c)
        c2, psi2 = c_new2(h,x2)
        alfa2 = alfa_ang(c2,psi2)
        beta2 = bet_ang(c2)
        caminar_paso(alfa2,beta2,alfa,beta)
        time.sleep(0.01)

    time.sleep(0.5)
```

4.5 Sensor ultrasónico HC-SR04

Para determinar los valores de la altura, desde la parte inferior hasta el suelo. Como se había mencionado se ha utilizado un Arduino Uno, por lo que, para poder ver los valores de la altura, se debe asignar las variables que se van a servir como *TRIGGER* y *ECHO*, en este caso:

```
const int Trigger = 2;
```

Lo cual implica que esté en el pin 2, será el de salida, mientras que la siguiente línea de código es para Echo.

```
const int Echo = 3;
```

Lo que implica que será in input. Estas consignas, se ven reflejadas en la parte principal del código del Arduino Uno, asimismo como inicializar la velocidad de transmisión de datos para la comunicación serial. Esto va a estar dentro de función *void setup ()*.

En la función *void loop ()*, va a contener; se asigna dos variables "t" y "d" del tipo float, con la finalidad de poder utilizarlas para almacenar el tiempo y la distancia medida. Con la función:

```
digitalWrite(Trigger, HIGH);
```

Activa el envío de las pulsaciones del sensor ultrasónico, después de un *delay* de 10 microsegundos, se detendrá el envío de pulsaciones con la función:

```
digitalWrite(Trigger, LOW);
```

Para determinar el tiempo que demora en llegar al puso que se ha enviado desde Trigger hacia el Echo después de rebotar por la superficie, se utiliza la función:

```
pulseIn(Echo,HIGH),
```

Donde el primer argumento, hace referencia al pin que está conectado el sensor, el segundo argumento indica que se espera que llegue un pulso alto. Posteriormente, se divide el valor de t entre 59 para obtener el valor de la distancia, basándonos en la Ecuación *14* enviando ese dato por medio de la comunicación serial a través de la siguiente línea de código:

```
Serial.printn(d);
```

Finalmente se da un *delay* de 100, para que se vuelva a ejecutar el código. En la placa Navio2, utilizando lenguaje de programación Python, se importará la librería serial, la cual llamará al objeto:

```
serial.Serial('/dev/ttyACM0',9600)
```

La cual indica que en el puerto ACM0 se va a enviar información por comunicación Serial y con esto se asigna una variable *PuertoSerie*. Después se llamará al método *readline()* a la variable denominada anteriormente.

Por último, el comando *decode('utf-8')*, permitirá decodificar la información que llega del puerto serial. Teniendo finalmente, la lectura en el terminal de Python. En objetivo de este, es trabajar todo en un mismo lenguaje de programación.

4.6 Creación del entorno y el agente

Para el desarrollo del entorno se va a utilizar las librerías *Gym* de parte de open AI, esta librería nos permite diseñar parte de los elementos que se necesitan para la clase que se va a crear. El entorno comenzará de manera básica, el cual se va a ir modificando a medida que se vaya requiriendo o teniendo mayor complejidad. Para ello, se importa las librerías que antes se han mencionado, por lo que se hará utilizando las siguientes líneas de código.

```
import gym
from gym import spaces
from gym.spaces import Discrete,Box
```

Se comenzará teniendo un modelo para el control de altura, otro para el control del ángulo *roll* y otro para el ángulo pitch. Para este control se debe crear una clase la cual se llamará *Enviroment*. Esta clase va a contar con cuatro métodos, los cuales van a ayudar para el desarrollo del agente. Dentro de estos métodos van a estar presentes los conceptos de recompensa, observación y acciones; así como también el estado.

4.6.1 *Método __init__()*

El primer método se va a denominar *Init*, el cual tiene como finalidad inicializar algunas variables que va a ser utilizadas dentro de los otros métodos. Este método va a requerir de un argumento llamado *receiver_node*, el cual viene de la clase que recibe los datos llamada *ReceiverNode*, esta clase recibe datos del nodo *sensor_node*, que se ha mencionado anteriormente. Cada método debe de contar con un argumento llamado "*self*", el cual permitirá llamar a una variable u otro método dentro de la clase.

```
def __init__(self,receiver_node):
self.receiver_node = receiver_node
```

Se va a tener en cuenta algunos parámetros para la creación de un entorno como el espacio de observaciones y el espacio de acciones; para ello se va a utilizar la función Box, que se importó anteriormente. Esta función va a contar un array, el cual se pedirá el límite inferior y el límite superior. Mientras que para el espacio de acciones se empleará la discrete. A continuación, unas líneas de código a modo de ejemplo, donde *n_action* representa el número de acciones discretas que va a requerir.

```
        self.action_space = Discrete(n_action)
        self.observation_space=Box(low=np.array([0.0]),high=np.array([100.0
        ]))
```

De igual forma, el estado con el que se va a inicializar. Para que el agente no esté ejecutando por un largo tiempo hasta que llegue a la meta, se va a asignar una variable el cual sería el máximo de tiempo que se lleva a cabo en el entorno, el valor será de diez segundos.

```
        self.timer_length = 10
```

Por último, también se va a inicializar la variable pub que será para publicar los valores que se va a estar obteniendo en el entorno.

```
        self.pub=rospy.Publisher('controlador',Int32MultiArray,queue_size=1
0)
```

4.6.2 *Método step()*

El segundo método tiene como nombre “step”, el cual pide como argumento a una acción. Dentro de este método, se va a calcular los *Rewards*; es decir, las recompensas que va a tener el agente cuando llegue a una meta y la perdida que va a tener si es que este no llega a cumplirlas.

Asimismo, las condicionales de cuándo se determine el entorno llamado *done*. Este método va a retornar las variables *state, reward, done* e *info*. El Step, va a variar dependiendo del modelo, aquí es donde se va a realizar las modificaciones necesarias.

4.6.3 *Método reset()*

Este método va a reiniciar los valores de las variables que se han estado operando en el método step, como son la variable *state* y la variable que se ha asignado para el tiempo que se va a tomar hasta que llegue a su meta. Este método retorna la variable *state.*

4.6.4 *Método discretizar()*

Este método se va a implementar exclusivamente para el modelo Qtable, recibe como argumento un valor, el cual será el *state*, y se implementará para discretizar. Este método, tomará los valores mínimos y máximos de las observaciones.

```
        def discretizar(self,valor):
            aux=((valor-
        self.observation_space.low)/(self.observation_space.high-
        self.observation_space.low)) *20
            return tuple(aux.astype(np.int32))
```

Como se observa en las líneas de código anteriores, el valor se restará con el valor mínimo de observaciones y dividirá con la resta de los dos valores máximos y mínimos; todo eso se va a multiplicar por 20. Para que de valores de 0 a 20 según sea el valor.

Este método retornará una tupla cuyo elemento será entero de 32 bits, a través de un array por medio de la función Numpy, como se muestra en el código.

4.7 Modificación del entorno según el modelo

Con los métodos principales que se han mencionado anteriormente, simplemente se debe modificar el algoritmo que cuenta cada método, los principales cambios que se va a tomar en cuenta estarán presentes en los métodos: __init__(), puesto que por cada modelo existirán variables e inicializaciones diferentes; step(), puesto que tendrá otro algoritmo por cómo se va a recolectar la información y obtener la recompensa por cada acción que toma. Y finalmente el método *reset()*, que tendrá las mismas modificaciones que el método __init__(). A continuación, se mostrará dichos cambios para cada modelo de entrenamiento que se va a tener en cuenta en esta investigación.

4.7.1 *Entorno para modelo de altura*

Para esto modelo, el entorno va a estar modificado por los tres métodos mencionados anteriormente. Hay que tener en cuenta que las acciones que se van a tomar son 3, estas son de valores discretos “0,1,2”. Primero se modificará el metodo __init__, como se muestra en las siguientes líneas de código.

```
self.action_space = Discrete(3)
self.observation_space = Box(low=np.array([0.0]), high = np.array([100.0]))
self.state = 15 + random.randint(-1,1)
self.timer_length = 10
```

Como se ha mencionado, va a tener tres acciones de valores discretos; la acción 0, significa que el agente va a bajar la altura, la acción 1, significa que el agente se va a mantener a esa altura, mientras la acción 2, va a hacer que el agente suba la altura.

En el método step, se va a tener las siguientes indicaciones para la modificación de los *state*, y el *timer_length*.

```
self.state += action -1
self.timer_length -= 1
```

Por cada acción se la va a restar uno, por lo que cuando la acción sea 0 al restar el 1, va a hacer que se reste un valor del *state*. Cuando el valor sea 1 al restar 1 quedará en 0, por lo tanto, el *state* no cambia. Por último, cuando la acción sea 2, al restar el 1, quedará en 1, por lo que se va a aumentar el valor del *state*.

De igual forma, se debe inicializar las variables de los datos que va a recibir del nodo sensores, como este dato es un *array*, para cada uno de los parámetros que vamos a seleccionarse por cada elemento de este *array*. Como se observa en las siguientes líneas de código.

```
altura = data[2]
pitch = data[1]
roll = data[0]
```

Para la recompensa se va a poner dentro de un condicional, y se va a asignar si es que el dato de altura está dentro de un valor que se selecciona. Por lo que, si se cumple, va a tener el valor de 1, y en el caso no se cumpla va a tener el valor de -1.

```
if (altura >= 14.0 and altura <= 16.5):
    reward = 1
else:
    reward = -1
```

Mientras que para que el entorno termine de ejecutarse, se debe de agregar la condición antes señalada y el tiempo que va a tomar llegue a 0, como se muestra en las siguientes líneas de código.

```
if ((altura >= 14.0 and altura <= 16.5) or (roll >= 0.0  and roll <= 10.0)) or (self.timer_length<= 0):
    done =True
    print("time:    {}    -    estado:    {}    -    altura_sensor: {}".format(self.timer_length,self.state,altura) )
else:
    done = False
    print("time:    {}    -    estado:    {}    -    altura_sensor: {}".format(self.timer_length,self.state,altura) )
```

Asimismo, se agrega un tiempo de reposo, este tiempo es para que el dato de altura que se obtiene en el entorno se pueda enviar de manera adecuada hacia el nodo *servo_node*.

4.7.2 *Entorno para modelo de roll-altura*

Para esto modelo, el entorno va a estar modificado para los tres métodos mencionados anteriormente. Hay que tener en cuenta que para este modelo las acciones que se van a tomar son 9, puesto que ahora se van a contar con dos alturas que van a ir variando, y como estas deben ser independientes entre sí, cada acción que ingrese no debe afectar a la otra, por lo que realizar una permutación de los valores con las cantidades extremidades, es la adecuada para tener cada decisión por separado, como se va a ver más a adelante.

Para ello se ha plasmado una expresión matemática que indica la cantidad de decisiones que se va a tener por cada extremidad o parámetro que se desea evaluar, bajo este contexto de modelo.

$$decisiones = 3^{\#extremidades} \qquad \text{Ecuación 33}$$

Por lo que por cada extremidad va a tener un valor de decisión, en este caso se va a considerar que las extremidades de la izquierda van a tener un valor inicial de altura, y la derecha otro valor, por lo que se va a considerar como dos extremidades, y si siguiendo con la Ecuación 33 nos da como resultado que el agente tendrá 9 decisiones.

A modo de resumen, se va a presentar una tabla de decisiones como se va a mostrar a continuación: Donde muestran las acciones que va a tener cada extremidad y como esta se expresa en un array para cada una de ellas, así como también la decisión general que va a tomar. Es decir, si es la decisión 0, eso quiere decir que será la acción 0 y 0 para las extremidades izquierda y derecha respectivamente; si es la decisión 1, será la acción 0 y 1 para izquierda y derecha y así sucesivamente.

Tabla 6

Decisiones para modelo con entorno roll

item	Decisión	Acción 1	Acción 2	Array
1	0	0	0	[0,0]
2	1	0	1	[0,1]
3	2	0	2	[0,2]
4	3	1	0	[1,0]
5	4	1	1	[1,1]
6	5	1	2	[1,2]
7	6	2	0	[2,0]
8	7	2	1	[2,1]
9	8	2	2	[2,2]

Los valores discretos “0 a 8”. Primero se modificará el método __init__, como se muestra en las siguientes líneas de código. En este caso se le va a agregar a una dimensión a

observation_space, en este caso se va a tomar el ángulo mínimo y máximo que podría "observar", el cual sería de -180.0 a 180.0.

En el caso de las accionas se creará con la función itertools, lo que se va a poner la estructura de las acciones que va a tener en este caso 0, 1, 2; y también la cantidad de repeticiones, la cual, para este modelo, será 2. Luego se hará un array donde va a realizar la permutación y se tendrá la lista creada.

```
      self.combinaciones = list(itertools.product([0, 1, 2], repeat=2))
      self.action_enviroment = [list(combinacion) for combinacion in self.combinaciones]
```

Para las accionas y las observaciones se hace como en el caso del modelo anterior.

```
      self.action_space = Discrete(9)
      self.observation_space = Box(low=np.array([-180.0,0.0]), high = np.array([180.0, 100.0]))
```

Como se ha mencionado, se va a tener dos alturas las cuales que se va a dar de manera aleatoria, y para tener en la variable *state*, se promediará las dos alturas, como se observa en la siguiente línea de código.

```
      self.altura1 = 15 + random.randint(-1,1)
      self.altura2 = 15 + random.randint(-1,1)
      self.altura =(self.altura1 + self.altura2)/2
```

Mientras que el caso del ángulo roll, se podrá como se observa en la siguiente línea de código, que se basa en la ecuación de la cinemática inversa.

```
      self.angulo=math.atan((self.altura1 - self.altura2)/10)*180.0/math.pi
```

Cuando se tiene el ángulo roll, y la altura, se creará lun array de nombre *state* que tendrá como elementos la altura y el ángulo. De igual forma, se asigna la variable timer_length para el tiempo que va a tomar en ejecutar el entorno.

```
      self.state = np.array([self.angulo,self.altura])
      self.timer_length = 10
```

Como se ha mencionado, va a contar con nueve acciones de valores discretos que serán las permutaciones que se han mencionado y de manera resumida son las siguientes; la acción 0, significa que el agente va a bajar la altura, la acción 1, significa que el agente se va a mantener a esa altura, mientras la acción 2, va a hacer que el agente suba la altura.

Por lo tanto, el método step, se va a contar con las siguientes indicaciones para la variable *state*, las alturas van a estar variando de forma independiente y de acuerdo con el espacio creado sobre las decisiones permutadas, por lo que se va a tener lo siguiente:

```
      self.altura1 += self.action_enviroment[action][0] -1
      self.altura2 += self.action_enviroment[action][1] -1
```

Como se observa anteriormente cuando se da un valor de acción, se la va a restar uno por lo que cuando la acción sea 0 al restar el 1, va a hacer que se reste un valor del *state*. Cuando el valor sea 1 al restar 1 quedará en 0, por lo tanto, el *state* no cambia. Por último, cuando la acción sea 2, al restar el 1, quedará en 1, por lo que se va a aumentar el valor del *state*. Y para que sea independiente para cada se agrega el otro corche que dato del array se escoge para cada altura, donde 0 será para la altura1 y 1 para la altura2.

Con esto simplemente, se va a promediar las alturas, y a calcular el ángulo *Roll*, como en se hizo en la parte de inicialización.

```
      self.altura = (self.altura1 + self.altura2)/2
      self.roll=math.atan((      self.altura1       -       self.altura2
)/10)*180.0/math.pi
```

Cuando se tiene estos datos, se crea el array, de los datos de *state*, como se observa la siguiente línea de código.

```
      self.state = np.array([self.roll, self.altura])
```

Asimismo, se debe inicializar las variables de los datos que va a recibir del nodo sensores, como este dato es un array, para cada uno de los parámetros que vamos a seleccionarse por cada elemento de este array. Como se observa en las siguientes líneas de Código.

```
      altura = data[2]
      pitch = data[1]
      roll = data[0]
```

La recompensa se asigna por medio de un condicional; si el dato del sensor de altura está dentro de los intervalos, se va a tener el valor de 1, en caso de que el ángulo esté dentro de los intervalos se le va a asignar 2, y en el caso no se cumpla va a tener el valor de -1. La recompensa por el ángulo se ha considerado como más importante debido este definirá mejor la orientación del prototipo en cuestión.

```
      if (altura >= 14.0 and altura <= 16.5):
          reward = 1
      if self.state[0] >= 0.0 and self.state[0] <= 10.0:
          reward = 2
      else:
          reward = -1
```

Mientras que para que el entorno termine de ejecutarse, se debe de agregar la condición antes señalada y el tiempo que va a tomar llegue a 0, como se muestra en las siguientes líneas de código.

```
      if ((altura >= 14.0  and altura <= 16.5) or (roll >= 0.0  and roll
<= 10.0)) or (self.timer_length<= 0):

          done =True

          print("time:    {}    -    estado:    {}    -    altura_sensor:
{}".format(self.timer_length, self.state,altura) )

      else:

          done = False

          print("time:    {}    -    estado:    {}    -    altura_sensor:
{}".format(self.timer_length,self.state,altura) )
```

Asimismo, se agrega un tiempo de reposo, este tiempo es para que el dato de altura que se obtiene en el entorno se pueda enviar de manera adecuada hacia el nodo servo_node.

Este mismo entorno se va a utilizar si es que se quiere analizar el ángulo pitch, lo único que debe cambiar es la base y en el caso del nodo servo_node, a que servomotores va a llegar ese dato.

Por último, en el método reset() va a inicializar las variables similares el método __init__(), se tomarán las variables altura1 y altura2, así como también la altura y el ángulo, para tenerlos en el array *state*.

4.7.3 *Entorno para modelo de pitch-roll-altura*

Para este modelo, se va a considerar cada extremidad de forma independiente, por lo que como dato de entrada va a ser distinto, lo que implica que se va a tener cuatro valores de altura. Y que en este caso se va a tener 81 decesiones siguiendo la Ecuación 33. Por lo tanto, se debe diseñar el array para las acciones. Se realiza el mismo método que se ha empleado anteriormente, pero en este caso en el parámetro repeat, se va a considerar 4.

```
      self.combinaciones = list(itertools.product([0, 1, 2], repeat=4))

      self.action_enviroment  =  [list(combinacion)  for  combinacion  in
self.combinaciones]
```

Y de igual forma que se ha asignado las decisiones, utiliza de manera discreta. Mientras que el espacio de observaciones aumentará el array a uno de tres, con los limites inferiores y superiores.

```
      self.action_space = Discrete(81)

      self.observation_space  =  Box(low=np.array([-180.0,  -180.0,  0.0]),
high = np.array([180.0, 180.0, 100.0]))
```

Se asigna cuatro variables las cuales van a ser las alturas de cada extremidad, como se observa en la siguiente línea de código.

```
self.extremidad1 = 15 + random.randint(-1,1)
self.extremidad2 = 15 + random.randint(-1,1)
self.extremidad3 = 15 + random.randint(-1,1)
self.extremidad4 = 15 + random.randint(-1,1)
```

Por lo que se determinará el promedio de las alturas y con ello se obtendría la altura que iría en el array.

```
self.altura = (self.extremidad1 + self.extremidad2 + self.extremidad3 + self.extremidad4)/4
```

Mientras que para el ángulo Roll y el ángulo Pitch, se debe calcular de igual forma como se ha estado haciendo anteriormente, pero en este caso se tendrá un ángulo Roll para la parte frontal y posterior; después de eso se tendrá que obtener el promedio de este ángulo. De igual forma, se hace con el ángulo Pitch. Por lo que se tiene la siguiente línea de código.

```
self.roll1 = math.atan((self.extremidad1 - self.extremidad2)/10)*180.0/math.pi
self.roll2 = math.atan((self.extremidad3 - self.extremidad4)/10)*180.0/math.pi
self.roll = (self.roll1 + self.roll2)/2
self.pitch1 = math.atan((self.extremidad1 - self.extremidad3)/10)*180.0/math.pi
self.pitch2 = math.atan((self.extremidad2 - self.extremidad4)/10)*180.0/math.pi
self.pitch = (self.pitch1 + self.pitch2)/2
```

Con estos parámetros, se puede obtener el array *state*, para que se parte de nuestro entorno, como se muestra a continuación.

```
self.state = np.array([self.roll, self.pitch, self.altura])
```

Mientras que, en el método step, como se ha trabajado anteriormente, será una sumatoria iterativa de los datos de altura. Por lo que quedaría de la siguiente manera.

```
self.extremidad1 += self.action_enviroment[action][0] -1
self.extremidad2 += self.action_enviroment[action][1] -1
self.extremidad3 += self.action_enviroment[action][2] -1
self.extremidad4 += self.action_enviroment[action][3] -1
```

Como las alturas van cambiando acorde con los datos de entrada de la acción, solo se llama a la variable y se efectúa los cálculos para tener los datos en el array *state*. Se tiene el dato de la variable altura, así mismo se calcula los ángulos roll1 y roll2, para las extremidades frontales y posteriores.

```
      self.altura = (self.extremidad1 + self.extremidad2 + self.extremidad3
+ self.extremidad4)/4
      self.roll1   =   math.atan((   self.extremidad1   -   self.extremidad2
)/10)*180.0/math.pi
      self.roll2   =   math.atan((   self.extremidad3   -   self.extremidad4
)/10)*180.0/math.pi
      self.roll = (self.roll1 + self.roll2)/2
      self.pitch1   =   math.atan((   self.extremidad1   -   self.extremidad3
)/10)*180.0/math.pi
      self.pitch2   =   math.atan((   self.extremidad2   -   self.extremidad4
)/10)*180.0/math.pi
      self.pitch = (self.pitch1 + self.pitch2)/2
```

En el caso de la recompensa, similar a lo que calculó anteriormente, se debe hacer emplear condicionales y se va a tomar más peso en los valores de los ángulos Roll y Pitch.

```
          if self.state[0] >= 0.0 and self.state[0] <= 10.0:
            reward = 2
          if self.state[1] >= 0.0 and self.state[1] <= 10.0:
            reward = 2
          if self.state[2] >= 37.0 and self.state[2] <= 39.0:
            reward = 1
          else:
            reward = -1
```

Y para dar por terminado el entorno, se va a considerar cuando el valor cumpla las condiciones anteriores o que el tiempo llegue a 0, como se observa en la siguiente línea de código.

```
      if   ((self.state[0]   >=   0.0   and   self.state[0]   <=   10.0)   and
(self.state[1] >= 0.0 and self.state[1] <= 10.0)   and (self.state[2] >=
37.0 and self.state[2] <= 38.0)) or (self.shower_length <=0):
          done = True
          print("time:{}                      -                      estado:
{}".format(self.shower_length,self.state) )
      else:
```

```
            done = False
            print("time:{}                -                estado:                {}
".format(self.shower_length,self.state) )
```

Este método va a retornar *self.state, reward, done, info,* para su uso en los modelos de entrenamiento. Por último, en el método reset, se inicializa los parámetros que se han utilizado en la función __init_().

4.7.4 *Entorno para modelo de traslación*

En este entorno, se tendrá se tendrá en cuenta como varía la altura a medida que el prototipo se va trasladando; por lo que similar a los entornos creados anteriormente, esta altura va a tener tres acciones aumentar, disminuir o mantener el valor.

Este entorno, se va a diferenciar de los demás por la forma en como recibe los estados. Anteriormente se obtenían a partir de cálculos matemática; sin embargo, para este caso se va a tener cuenta que los estados van a venir de los sensores, haciendo que este entorno para el prototipo sea más capaz de reconocer y las decisiones que se tome se las más adecuadas para su comportamiento en su entorno.

Al inicio, se modifica el método __init__() el cual se van a asignar las acciones como se ha venido trabajando, al igual como el espacio de acciones y espacio de estados; se toma en cuenta tres estados, el límite inferior será el array [-180.0, -180, 0.0] mientras que el límite superior será [180.0, 180.0, 100.0], los valores del estado se inician con los valores del nodo sensor, y se asigna con las variables self.roll, self.pitch y self.altura, y por último se asigna la variable *self.state* el cual es un array cuyo elementos serán los que antes se ha mencionado. En este caso se tomará al dato de altura como un valor entero, por lo que al inicio se tomará como entero este valor.

```
      self.combinaciones = list(itertools.product([0, 1, 2], repeat=1))
      self.action_enviroment = [list(combinacion) for combinacion in
self.combinaciones]
      self.receiver_node = receiver_node
      self.action_space = Discrete(3)
      self.observation_space = Box(low=np.array([-180.0,-180.0,0.0]), high
= np.array([180.0,180.0,100.0]) )
      while self.receiver_node.get_received_data() is None:
          time.sleep(1)
      data = self.receiver_node.get_received_data()
      self.roll = data[0]
      self.pitch = data[1]
      self.altura = int(data[2])
      self.state = np.array([self.roll,self.pitch,self.altura])
```

Por su parte, en el método step() se toma la variable altura y con cada acción que tome el entorno, su altura va cambiando como se ha especificado. Como se detalla en la siguiente línea de código.

```
self.altura += self.action_enviroment[action][0] -1
```

Mientras que la recompensa se tendrá de trabajando de igual forma, tendrá mayor peso cuando los valores del ángulo Roll y Pitch, los óptimos, mientras que la altura sea la adecuada. Para dar por terminado el episodio del entorno, no se ha considerado un tiempo como máximo por iteración como los entornos anteriores. Cuando se llegue a la altura determinada y se cumple con los ángulos se dará por terminado. Este método retornará el estado, la recompensa y si el entorno ha terminado.

Por último, en el método reset() se tendrá los valores que se han inicializado en el método __init__(), y retornará el valor del estado en ese momento. De igual forma, tanto en el método step(), como en el reset(), se va a incluir la función publish(), para enviar este valor hacia el nodo servo.

4.8 Creación del modelo Qtable

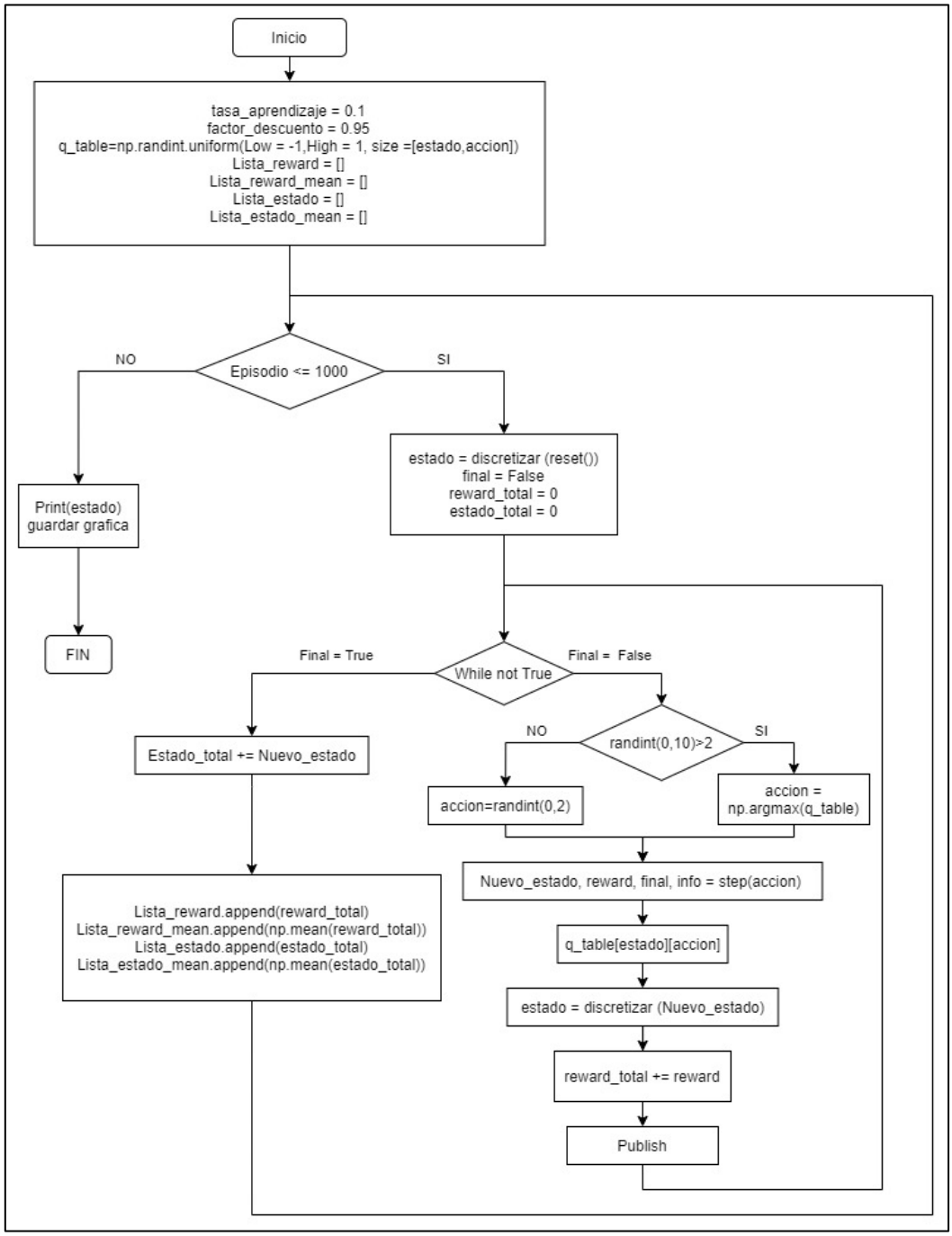
Como se ha mencionado anteriormente, se ha implementado con las ecuaciones de Bellman, por lo que se requiere de una Qtable, para ello se ha creado por un array con las dimensiones del estado y las acciones. Este array va a tener valores que están entre -1 y 1 de forma aleatoria por toda cada elemento del array. Esto se hace por medio de la siguiente línea de código.

```
q_table = np.random.uniform(low = -1, high = 1, size = [Estado,Acciones])
```

La ecuación de Bellman que se ha mencionado en la sección ***El entorno del aprendizaje por refuerzo*** requiere de una tasa de aprendizaje y un factor de descuento; tasa de aprendizaje debe de contar con valores cercanos al 0, mientras que el factor de descuento debe estar cercana a 1.

Figura 39

Diagrama de flujo de algoritmo de Qtable



4.8.1 *Modelo Qtable para altura*

En el caso de la altura se debe tener en cuenta las dimensiones del estado y las dimensiones de las acciones. El entorno va a dar un array de una dimensión, mientras que las acciones serán de valor 3. Sin embargo, estos valores van a ser continuos y para poder trabajar con Qtable deben ser discretos; por lo que, se requiere realizar una discretización, para ello se utilizar el método que se ha diseñado anteriormente.

La Qtable, ya con el valor discretizado, se tendrá de la dimensión de los valores que se han determinado en la función, en este caso es 20, entonces se va a tener un array de [20, 3], donde el primer elemento será la dimensión del estado y el segundo será el de las acciones. Su diseño en Python será el siguiente.

```
q_table = np.random.uniform(low = -1, high = 1, size = [20,3])
```

Con esto, ya se tiene la Qtable lista para implementar con los algoritmos, se escoge el factor de descuento y la tasa de aprendizaje que para fines prácticos será 0.95 y 0.1, respectivamente.

Se va a entrenar con 1000 episodios, por lo que se va a utilizar un bucle *for* donde se va a ir iterando en cada evento del entorno. Al principio de este bucle se va a tener que inicializar el valor del estado, esto se hará utilizando el método reset() y este dato pasa por el método discretizar().

De igual forma se considera que la variable "final" sea False, con el fin de que se emplee un bucle while, el cual pueda iterar el entorno y no cambie hasta que dicha variable retorne un True.

Luego se tiene que seleccionar la acción, se generará de manera aleatoria; para ello se utiliza un condicional, el cual, si es que acción es mayor que 2, se selecciona el valor máximo que pueda dar la Qtable, en caso contrario que escoja entre 0 a 2.

Dicha acción escogida se va a emplear en el método step(), el cual va a dar un nuevo estado, una recompensa y si ha terminado el entorno o no. Con estas variables se pueden implementar en la ecuación de la Qtable, lo cual nos deja la siguiente línea de código.

```
q_table[estado][accion] = q_table[estado][accion] + tasa_aprendizaje
*          (recompensa          +          factor_descuento          *
np.max(q_table[discretizar(nuevo_estado)]) - q_table[estado][accion]
```

Con ello, cada valor que toma la qtable, será representado por estado y acción. Luego se actualiza el estado con el estado_nuevo que da el método step(). Para poder graficar, simplemente se hace una sumatoria a una variable de reward_total, y se va acumulando.

Para que esto se pueda ejecutar de manera adecuada, los datos de la altura se van a enviar por medio de un Publish de ROS, para que las extremidades puedan moverse de acuerdo con la mejor opción que da la Qtable.

4.8.2 *Modelo Qtable para roll y altura*

En el caso de este modelo, el espacio de observaciones se ha modificado como se mencionó anteriormente, eso quiere decir que ahora el espacio de observaciones va a tener un array de (2,) como mínimo y máximo. De igual forma se puede discretizar con el método creado, y con ello tendríamos 20 estados discretizados para cada observación, es decir 20 estados para la variable del ángulo y 20 estados la altura.

Por lo tanto, la variable Qtable se le debe agregar otra dimensión por lo que va a tener la forma de [20,20,9], como se puede observar en el array el ultimo va a ser el de las acciones que para este modelo será ochenta y uno. Por lo que, la Qtable se diseñará de la siguiente manera.

```
q_table = np.random.uniform(low = -1, high = 1, size = [20,20,9])
```

Se empleará los mismos valores para los parámetros de tasa de aprendizaje y el factor de descuento. De igual forma se ejecutará con 1000 episodios para que pueda aprender de manera adecuada. Y como se ha estado elaborado en los otros dos modelos, se iniciará la variable "estado" discretizando lo que se obtiene del método reset(). Y la variable "final" con el valor False.

Se ingresa al bucle while, donde se seleccionará la acción adecuada, es decir si es que la función de la acción que se ha seleccionado de manera aleatoria es mayor a los valores de las acciones, por lo tanto, se tomará el valor más alto de la Qtable, mientras que si no lo es escogerá valores entre 0 y 8.

Ese valor de acción ingresará al método step(), para que nos de valores del Nuevo_estado, reward, final y la info, en la ecuación de la Qtable.

Luego se actualiza el valor de la variable "estado" con el nuevo valor que se obtiene del step(), y se va aumentando el valor de la variable reward, con el dato que se tiene modificado de la variable de la altura, se emplea la función publish para enviar ese dato hacia el nodo servo, para su posterior ejecución. En este caso se enviará dos valores correspondientes a los a las alturas de las extremidades.

4.8.3 *Modelo Qtable para Roll, Pitch y altura*

En el caso de este modelo, el espacio de observaciones se ha modificado como se mencionó anteriormente, eso quiere decir que ahora el espacio de observaciones va a tener un array de (3,) como mínimo y máximo. De igual forma se puede discretizar con el método creado, y con ello tendríamos 20 estados discretizados para cada observación, es decir 20 estados para la variable del ángulo roll, 20 estados para la variable del ángulo pitch y 20 estados la altura.

Por lo tanto, la variable Qtable se le debe agregar otra dimensión por lo que va a tener la forma de [20,20,20,81], como se puede observar en el array el ultimo va a ser el de las acciones que para este modelo será nueve. Por lo que, la Qtable se diseñará de la siguiente manera.

```
q_table = np.random.uniform(low = -1, high = 1, size = [20,20,20,81])
```

Se empleará los mismos valores para los parámetros de tasa de aprendizaje y el factor de descuento. De igual forma se ejecutará con 1000 episodios para que pueda aprender de manera adecuada. Y como se ha elaborado anteriormente, y siguiendo el algoritmo, se iniciará la variable "estado" discretizando lo que se obtiene del método reset(). Y la variable "final" con el valor "False".

Se ingresa al bucle while, donde se seleccionará la acción adecuada, es decir si es que la función de la acción que se ha seleccionado de manera aleatoria es mayor a los valores de las acciones, por lo tanto, se tomará el valor más alto de la Qtable, mientras que si no lo es escogerá valores entre 0 y 80.

Ese valor de acción ingresará al método step(), para que nos de valores del Nuevo_estado, reward, final y la info, en la ecuación de la Qtable, que es igual a la que se ha mencionado anteriormente, no se modifica la estructura del código.

Luego se actualiza el valor de la variable "estado" con el nuevo valor que se obtiene del step(), y se va aumentando el valor de la variable reward, luego con el dato que tiene modificado de la variable de la altura, se emplea la función publish para enviar ese dato hacia el nodo servo, para su posterior ejecución.

4.8.4 *Modelo Qtable para traslación*

Para este modelo, se tendrá de igual forma la creación de la Qtable similar a la que se plasmado en el modelo anterior de forma estática, sin embargo, como se han considerado los demás parámetros del estado como el ángulo Roll y Pitch, por lo tanto, la variable de estado tendrá las mismas dimensiones que se han considerado en el modelo anterior.

El entorno va a dar un array de una dimensión, mientras que las acciones serán de valor 3. Sin embargo, estos valores van a ser continuos y para poder trabajar con Qtable deben ser discretos; por lo que, se requiere realizar una discretización, para ello se utilizar el método que se ha diseñado anteriormente "discretizar()".

```
q_table = np.random.uniform(low = -1, high = 1, size = [20,20,20,3])
```

Dentro del algoritmo de la Qtable, se seguirá implementando los valores como se indica en el diagrama de flujo, en este caso, como las acciones son tres, cuyo mínimo será 0 y el máximo será 2, cuando se va a considerar los valores de las acciones, estas se van a considerar como el modelo para la Qtable de altura.

Este entorno al no contar con un tiempo de máximo, las etapas van a ser relativamente rápidas o tomarán mayor tiempo, de acuerdo con el agente y como este se vaya comportando para lograr la recompensa esperada.

Con estos cuatro modelos, se podrá trabajar de forma iterativa el aprendizaje por refuerzo, sin embargo, se trabajará con un modelo de redes neuronales con el fin de poder obtener un modelo que luego se pueda implementar.

4.9 Creación del modelo con DQN

Para la creación del modelo de red neuronal, se debe tener en cuenta que, para cada tipo de modelo, la estructura neuronal es diferente, puesto que tiene entradas de diferentes dimensiones y las salidas de igual forma. Así como también la cantidad de neuronas debe ser diferentes para evitar un sobreajuste o sub-ajuste, es decir que el modelo no pueda aprender adecuadamente, o que el modelo este restringido para ese caso.

4.9.1 *Modelo Altura*

Para la creación del modelo se debe tener en cuenta los valores de entra y de salida que va a tener, en específico las dimensiones. En el caso del modelo creado para la altura, se debe tener en cuenta la dimensión del array que va a ingresar se una sola dimensión, por lo tanto, al momento que ingresa a la primera capa no habría ningún problema.

Se va a crear una función de nombre "*build_model*" donde va a contener la estructura de la red que se va a trabajar para este modelo; primero se asigna que la variable va a recibir como argumentos "*state*s" y "*actions*". Estos parámetros son de la clase entorno que se ha mencionado anteriormente. Luego se asigna la variable "model", la cual va a tener como objeto Sequential, lo que significa que el modelo va a ser de tipo secuencial y que las capas se van a agregar de esa manera.

La primera capa va a contar con 24 neuronas, estas neuronas van a tener como salida una función de activación, en este caso se va a utilizar una función *"relu",* esa capa va a contar con la entra "*states*" cuya dimensión es (1,). La siguiente capa contará nuevamente con 24 neuronas, cuya salida va a ser de igual forma una función de activación *"relu".*

Por último, se va a contar con una capa de salida la cual va a contar con las dimensiones de *"actions",* que para este modelo serán 3 y contará con una función de activación *"linear".* Para finalizar esta función va a contar retornar el modelo creado. En python, el código sería como se observa a continuación.

```
def build_model(states, actions):
    model = Sequential()
    model.add(Dense(24, activation='relu',input_shape=states))
    model.add(Dense(24, activation='relu'))
    model.add(Dense(actions, activation='linear'))
    return model
```

Para crear el modelo primero se debe asignar las variables *"states"* y *"actions*", como se mencionó anteriormente, estas provienen del entorno. La variable *"states"* debe de ser del tipo tuple, por lo que se debe de utilizar la extensión *".shape";* mientras que *"actions"* al ser discretos, solo requiere de la extensión ".n".

```
states = gym_env.observation_space.shape
actions = gym_env.action_space.n
```

Con esto, se puede crear el modelo, se asigna una variable y se solicita los argumentos que antes se han creado.

```
model = build_model(states, actions)
```

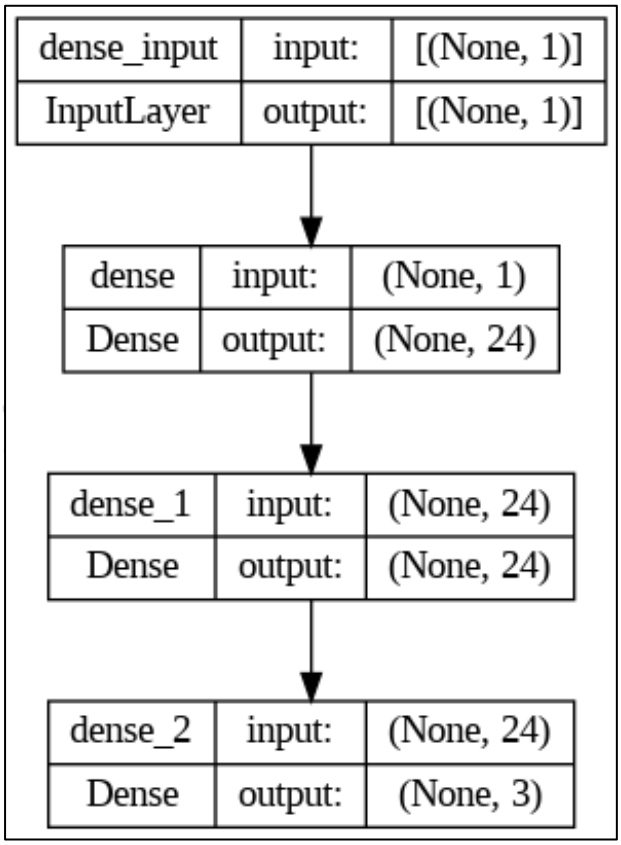
La dependencia tensorflow, cuenta con la opcion de poder visualizar el modelo, y poderlo guardar dicha imagen, para ello se emplea la función plot_model, cuyo argumento será el modelo creado y otros argumentos como, si se decide guardar el archivo entre otras opciones.

```
      plot_model(model      =      model,      show_shapes      =      True,
to_file='model_altura_plot.png')
```

Esto hará que se descargue la imagen en forma png, dando la información de las capas de entrada con la cantidad de neuronas de salida entre capa y capa como se muestra en la siguiente figura.

Figura 40

Grafica de la red neuronal del modelo altura



De igual forma, se puede tener un resumen con la información del modelo; para poder guardar este resumen, se creará una función que pueda guardar los datos en un archivo txt, para que al final se puede guardar este archivo en una imagen.

Para ello, se debe de recrear una función *with*, la cual va a hacer crear un archivo cuyo nombre sea “model_summary.txt”, donde en este archivo, se va a crear la función “model.summary”. Cuando se tenga creado, se procede a abrir este archivo, con la extensión “.read()”, y con librería Matplotlib, se escoge las dimensiones de la imagen y por último, como se va a guardar el archivo, como se observa en la siguiente línea de código.

```
      model.summary()
```

```
    with open ('model_summary.txt', 'w') as f:
        with redirect_stdout(f):
            model_altura.summary()

    with open ('model_summary.txt', 'r') as f:
        model_altura_summary_text = f.read()

    plt.figure(figsize=(10, 5))
    plt.text(0.1,    0.5,    model_altura_summary_text,    fontsize=10,
family='monospace')
    plt.axis('off')

    plt.savefig('model_altura_summary.png',        bbox_inches='tight',
pad_inches = 0.1)
```

En este resumen, que se puede observar en la siguiente figura, nos indica las capas y la cantidad de neuronas que tiene por cada capa, así como también los parámetros que va a tener por cada capa, así como también la cantidad de parámetros totales y los parámetros que va a estar en entrenamiento. Por lo que, según la figura, todos los valores que va a pasar por cada capa van a servir para el entramiento de la red.

Figura 41

Tabla resumen de la red neuronal del modelo altura

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 24)                48
_________________________________________________________________
dense_1 (Dense)              (None, 24)                600
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 75
=================================================================
Total params: 723
Trainable params: 723
Non-trainable params: 0
_________________________________________________________________
```

Para este modelo, se ha optado una red relativamente sencilla, esto se puede ir modificando para obtener mejores resultados. Se puede agregar más neuronas, modificar la función de activación por capa, agregar más capas, y así sucesivamente.

4.9.2 *Modelo Roll-Altura*

Para la creación de este modelo se debe tener en cuenta los valores de entra y de salida que va a tener. En el caso del entorno creado para el ángulo y altura, se debe tener en cuenta que la dimensión del estado es un array de (2,); por lo tanto, para que pueda ingresar a la primera capa se debe realizar una modificación en su dimensión.

Se va a crear una función de nombre "build_model2" donde va a contener la estructura de la red que se va a trabajar para este modelo; esta función, similar como la anterior, va a recibir de entrada "*states*" y de salida *"actions"*. Luego se asigna la variable "model", la cual va a tener como objeto *Sequential*, lo que significa que el modelo va a ser de tipo secuencial y que las capas se van a agregar una tras otra.

Primero se debe modificar la entrada añadiendo una capa que tenga de característica *"Flatten",* la cual va a tener dentro de su argumento, un tamaño de entrada, de (1,) sumado las dimensiones del *"states".* Una vez que se obtenga esta modificación se procede a diseñar la estructura de la red.

La primera capa va a contar con 24 neuronas, estas neuronas van a tener como salida una función de activación, en este caso se va a utilizar una función *"relu"*. La siguiente capa contará nuevamente con 24 neuronas, cuya salida va a ser de igual forma una función de activación *"relu".* Por último, se va a contar con una capa de salida la cual va a contar con las dimensiones de *"actions",* que para este modelo serán nueve y contará con una función de activación *"linear".*

Para finalizar esta función va a contar retornar el modelo creado. En Python, el código sería como se observa a continuación.

```
def build_model2(states, actions):
  model = Sequential()
  model.add(Flatten(input_shape=(1,) + states))
  model.add(Dense(24, activation='relu'))
  model.add(Dense(24, activation ='relu'))
  model.add(Dense(actions, activation='linear'))
  return model
```

Para la ejecución de este modelo, primero se debe asignar las variables *"states"* y *"actions",* de igual forma como se hizo en el modelo anterior. La variable *"states*" debe de ser del tipo tuple, por lo que se debe de utilizar la extensión *".shape";* mientras que *"actions"* al ser discretos, solo requiere de la extensión ".n" .

```
states = gym_env.observation_space.shape
actions = gym_env.action_space.n
```

Con esto, se puede crear el modelo, se asigna una variable y se solicita los argumentos que antes se han creado.

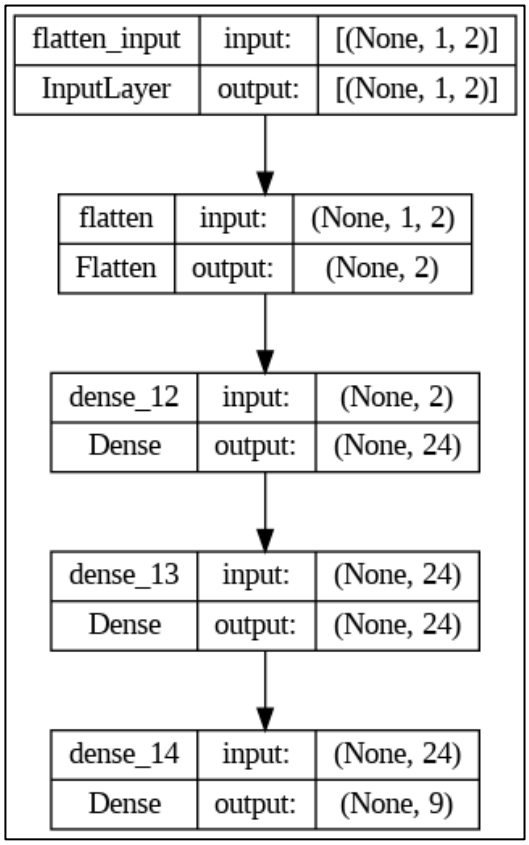
```
model2 = build_model2(states, actions)
```

En el diagrama de la estructura neuronal se puede observar que se ha modificado, la función build_model da como resultado una capa extra, la cual ajusta a las dimensiones que se requiere.

En detalle, al inicio la variable *states* tiene una dimensión de (1,1,2), mientras que el modelo como tal recibe (1,2), por lo que con la modificación se puede ajustar de manera adecuada a lo que requiere el modelo neuronal. Estos detalles se pueden visualizar mejor con la función model_plot, la función que se ha implementado en el modelo anterior, la cual da como resultado el diagrama de flujo donde se visualizan las capas, las neuronas y las dimensiones que posee ver Figura 42.

Figura 42

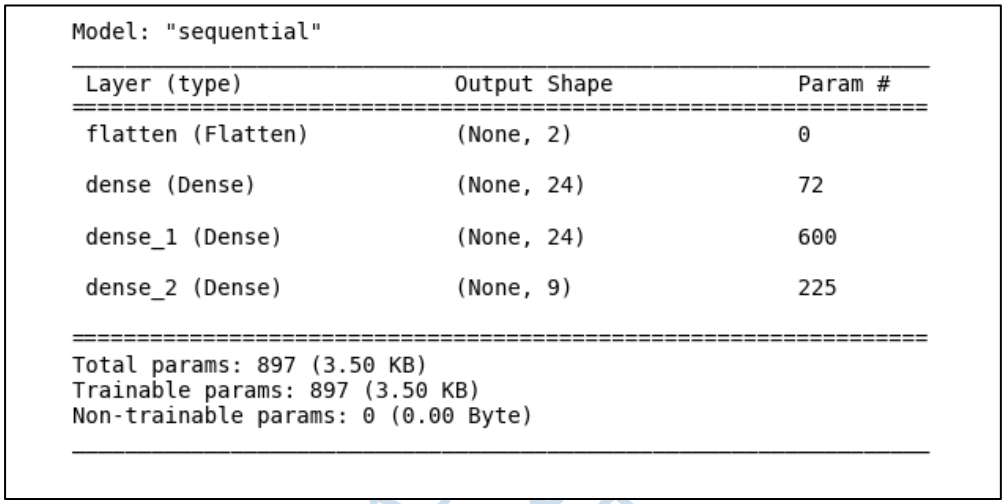
Grafica de la red neuronal del modelo roll-altura



De igual forma, se tendrá un resumen de la estructura neuronal con la que se ha estado trabajando, se emplea la misma función con la que se ha estado trabajando, para que nos alcance de los parámetros que se han estado abordando por cada capa.

Figura 43

Tabla resumen de la red neuronal del modelo roll-altura



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 2)                 0

 dense (Dense)               (None, 24)                72

 dense_1 (Dense)             (None, 24)                600

 dense_2 (Dense)             (None, 9)                 225

=================================================================
Total params: 897 (3.50 KB)
Trainable params: 897 (3.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Como se observa en la figura, en este caso, al cambiar la capa de entrada, realizando un aplanado de las dimensiones de entrada, la siguiente capa de veinticuatro neuronas ha cambiado de parámetros, en el anterior modelo, nos daba un modelo cuarenta y ocho parámetros, mientras que, para este modelo, se obtiene setenta y dos parámetros, de igual forma surgen cambios con respecto a las demás capas, dando un total de ochocientos noventa siete parámetros en comparación a los setecientos veintitrés del modelo anterior.

4.9.3 *Modelo Roll, Pitch y Altura*

Para la creación de este modelo se debe tener en cuenta los valores de entra y de salida que va a tener, en específico las dimensiones. Por lo que, para este modelo, las entradas son de dimensión (3,).

Primero se asigna que las variables *"states"* y *"actions"*, los cuales servirán como argumento para la función donde va a contener el modelo creado. Luego se asigna la variable "model", la cual va a tener como objeto *Sequential*, lo que significa que el modelo va a ser de tipo secuencial y que las capas se van a agregar de esa manera.

Similar con el modelo anterior, se debe modificar la capa inicial agregando la función Flatten, para aplanar las dimensiones del *state*, puesto que este va a ingresar con dimensiones (1,1,3) mientras que la red va a requerir de (1,3), cuando se haya modificado esta capa de inicio, se agregaran las demás.

La siguiente capa que le procede a la capa de inicio, va a contar con nueve neuronas, esto decisión se debe a que se va a seguir un patrón ascendente y descendente con la cantidad de neuronas que hay por cada, es decir que las primeras capas van a ir incrementando y luego se van a ir reduciendo de manera simétrica. Con cinco capas, siendo la tercera la que tenga el valor más alto de neuronas y el punto de descenso. Así mismo, las neuronas deben ser múltiples tanto de las entradas como de las salidas.

Con lo mencionado, la primera capa que sigue después de la de entrada, va a contar con nueve neuronas y una función de activación *"relu",* la siguiente capa será de ochenta y un neuronas teniendo de igual forma una función de activación *"relu",* como se puede observar se va ir subiendo exponencialmente, es decir que la siguiente capa contará con seis mil quinientos sesenta y uno, este será la capa que contenga la mayor cantidad de neuronas, después de eso se va a recibir por cada capa con la misma cantidad de neuronas que ha comenzado. Cabe resaltar que por cada capa seguirá siendo la misma función de activación *"relu".* Por último, se va a contar con una capa de salida la cual va a contar con las dimensiones de *"actions",* que para este modelo serán ochenta y uno, y contará con una función de activación "linear". Para finalizar esta función va a retornar el modelo creado. En Python, el código sería como se observa a continuación.

```
      def build_model2(states, actions):
        model = Sequential()
        model.add(Flatten(input_shape=(1,) + states))  model.add(Dense(9,
activation='relu'))
        model.add(Dense(81, activation ='relu'))
        model.add(Dense(6561, activation ='relu'))
        model.add(Dense(81, activation ='relu'))
        model.add(Dense(actions, activation='linear'))
        return model
```

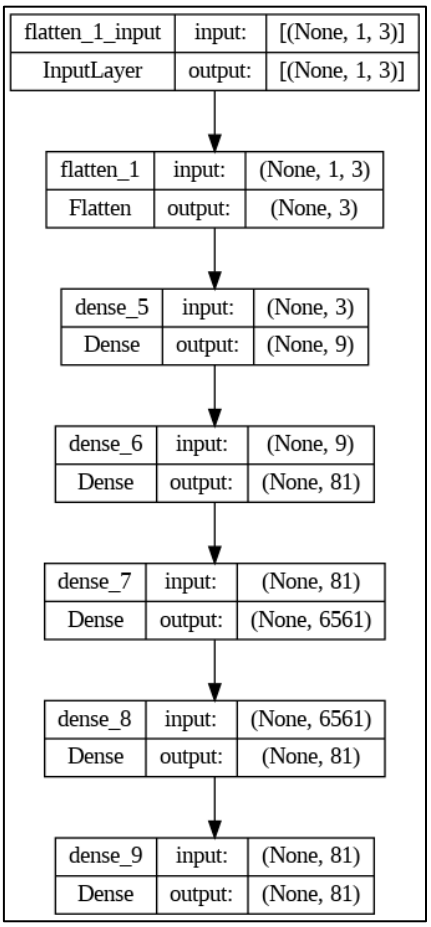
Para crear el modelo primero se debe asignar las variables *"states"* y *"actions",* como se mencionó anteriormente, estas provienen del entorno. Con esto, se puede crear el modelo, se asigna una variable y se solicita los argumentos que antes se han creado.

```
      model = build_model(states, actions)
```

La dependencia tensorflow, cuenta con la opción de poder visualizar el modelo, y poderlo guardar dicha imagen, para ello se emplea la función plot_model, cuyo argumento será el modelo creado y otros argumentos como, si se decide guardar el archivo entre otras opciones. Esto hará que se descargue la imagen en forma png, dando la información de las capas de entrada con la cantidad de neuronas de salida entre capa y capa como se muestra en la siguiente figura.

Figura 44

Tabla resumen de la red neuronal del modelo roll-pitch- altura



Mientras que la tabla resumen del modelo da como resultado la siguiente figura.

Figura 45

Tabla resumen de la red neuronal del modelo roll, pitch altura

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 3)                 0

 dense_5 (Dense)             (None, 9)                 36

 dense_6 (Dense)             (None, 81)                810

 dense_7 (Dense)             (None, 6561)              538002

 dense_8 (Dense)             (None, 81)                531522

 dense_9 (Dense)             (None, 81)                6642

=================================================================
Total params: 1077012 (4.11 MB)
Trainable params: 1077012 (4.11 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

4.9.4 *Modelo de traslación*

Para el modelo de traslación, la estructura neuronal es similar a como ha estado trabajando en el modelo de altura, con la diferencia que la entrada se tiene que poner una capa flatten para los estados de inicio, puesto que estos son un array y luego va a ingresar a la capa que cuenta con veinticuatro neuronas y con su función de activación de *"relu".* Luego pasará por su segunda capa que cuenta con las mismas características, y finalmente pasará por una capa de salida la cual cuenta con las funciones de salida. A detalle se tiene lo siguiente:

La primera capa va a contar con 24 neuronas, estas neuronas van a tener como salida una función de activación, en este caso se va a utilizar una función *"relu",* esa capa va a contar con la entra *"states"* cuya dimensión es (1,). La siguiente capa contará nuevamente con 24 neuronas, cuya salida va a ser de igual forma una función de activación *"relu".*

Por último, se va a contar con una capa de salida la cual va a contar con las dimensiones de *"actions",* que para este modelo serán 3 y contará con una función de activación "linear". Para finalizar esta función va a contar retornar el modelo creado.

Después, se graficará los valores de la estructura neuronal que se ha diseñado con la función que se ha implementado anteriormente plot_model, con el fin de tener una noción de cómo va a trabajar la red neuronal en el prototipo, dando como resultado lo que se observa en la **Figura 46**, de igual forma se graficará la tabla resumen con los parámetros de operación que tiene esta estructura neuronal, como se va a observar en la Figura 47.

Figura 46
Grafica de la red neuronal del modelo traslación

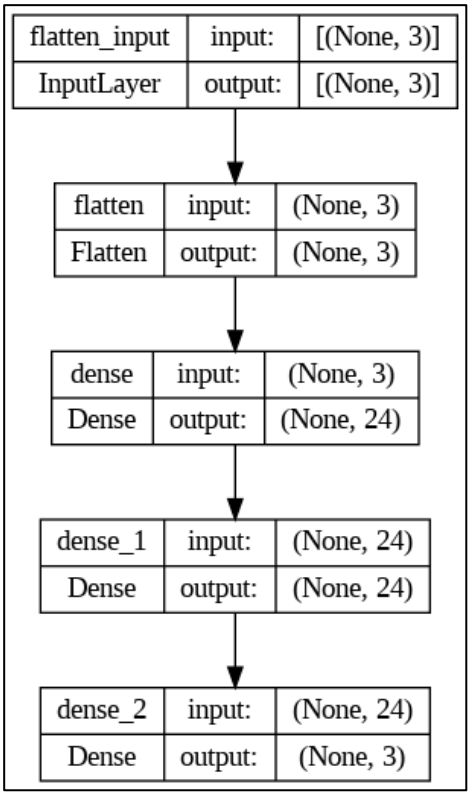
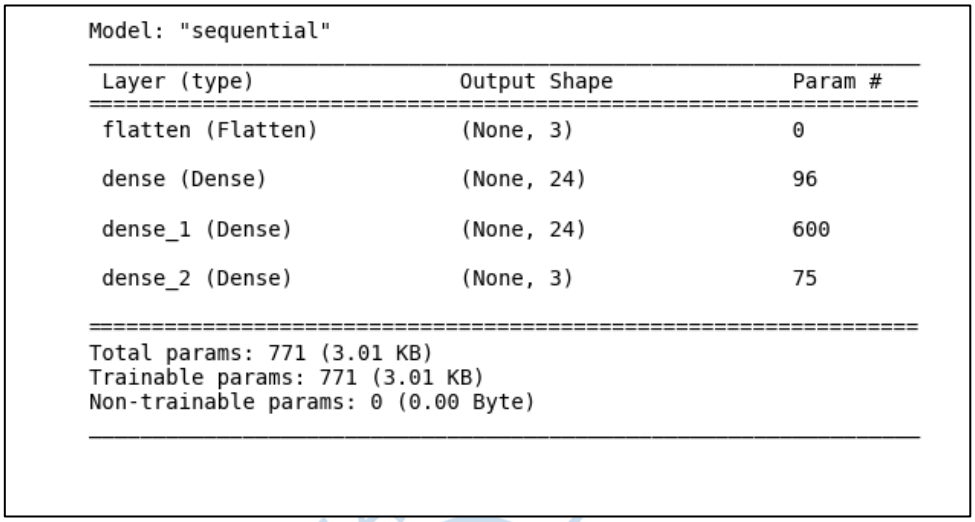


Figura 47

Tabla resumen de la red neuronal del modelo traslación



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 3)                 0

 dense (Dense)               (None, 24)                96

 dense_1 (Dense)             (None, 24)                600

 dense_2 (Dense)             (None, 3)                 75

=================================================================
Total params: 771 (3.01 KB)
Trainable params: 771 (3.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

4.10 Creación del modelo del agente

Los modelos que se han creado anteriormente son la estructura de la red neuronal que se ha estado trabajando; sin embargo, para este tipo de aprendizaje se debe de poder trabajar para en el entorno, se debe crear una función llamada build_agent, la cual reciba como argumento, el modelo que se han creado anteriormente, y la cantidad de acciones que presente el modelo.

En esta función se van a definir las políticas que va a trabajar el modelo, así como también el método de memoria con el que trabaja, en este caso escogerá un objeto de modo de memoria secuencial, donde se escoge los límites. Al final se crea un objeto llamado DQNAgent, el cual va a pedir como argumentos el modelo, el tipo de memoria, y la política con la que trabaja, así como también los valores de las acciones, y el paso; esta función va a retornar la variable a que se ha asignado el objeto DQNAgent. Esta función en Python sería de la siguiente manera.

```
      def build_agent(model,actions):
          policy = BoltzmannQPolicy()
          memory = SequentialMemory(limit = 10000, window_length=1)
          dqn = DQNAgent(model = model, memory = memory,policy = policy,
nb_actions = actions, nb_steps_warmup =10, target_model_update=1e-2)
          return dqn
```

Cuando se haya diseñado se procede a la creación de una variable llamada “dqn”, y se inserta el modelo y las acciones. Finalmente, se compila el modelo del agente creado, se escoge un método de compilación de optimización, en este caso será el método Adam, cuya tasa de aprendizaje será de 0.001, y la métrica con la que se va a trabajar para obtener la mayor eficiencia será mean squeare error, o promedio de errores al cuadrado.

```
dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mse'])
```

Cuando se tenga la compilación se procede al entrenamiento el modelo, para ello se escoge el objeto “fit”, cuyos argumentos serán el entorno denominado “env”, la cantidad de episodios que va a estar entrenando, si se desea visualizar o no, en este caso se considera como False, debido a que esta función es para un entorno simulado y si se requiere de visualización es para el renderizado, la forma en como hemos implementado esta compilación es de manera diferente. También se debe implementar el argumento “verbose” el cual tiene tres opciones, en este caso se seleccionará la opción 1 para poder visualizar a través de una barra de progreso.

```
dqn.fit(env, nb_steps = 100, visualize= False, verbose = 1)
```

Por último, cuando se haya entrenado el modelo, se realizará un testeo, para esto se empleará la función “test”, la cual va a pedir como argumento el entorno, la cantidad de episodios que se requiera testear, de igual forma si es que se requiera visualizar en este caso “False” y el “verbose” que este caso será 1 de igual forma.

```
dqn.test(env, nb_episodes = 1000, visualize = False, verbose=1)
```

Este modelo, se utiliza para todas las estructuras de redes que antes se han diseñado, la función test, nos permite ver cómo se ha comportado el modelo y si se tiene la recompensa que se espera.

4.11 Reconocimiento visual de entorno

En esta sección se va a implementar un valor extra al prototipo, el cual se basa en la percepción del entorno por medio de visión artificial. Para ello, se va a importar unas dependencias, las cuales serán Opencv y Numpy.

```
import cv2
import numpy as np
```

Con OpenCV, se tendrá la posibilidad de poder encender la cámara, a través de la función VideoCapture y de argumento un valor discreto como 0, 1, 2 etc. Siendo esto los valores de los dispositivos, como se observa a continuación:

```
cap = cv2.VideoCapture(0)
```

A modo de prueba, se hará un reconocimiento de una gama de colores azules, para ello, se asignará los valores de azul en bajo y azul en alto, poniendo en un array, los valores que corresponde en formato BRG.

```
azulBajo = np.array([100,100,20], np.uint8)
azulAlto = np.array([125,255,255], np.uint8)
```

Cuando se tiene asignado estas variables, entrará en un bucle while, donde siempre será True, y que leerá las variables frame y ret, siendo este ultimo la resolución de la imagen. Si el valor de ret retorna un True, es decir que lea correctamente, dentro de él se tendrá una variable el cual será un objeto que reciba los frames y pueda obtener los colores de este.

Asimismo, otra función donde recibe los colores que antes se ha determinado, y que estén dentro del rango de colores azules que se han asignado anteriormente. Finalmente, se mostrará en un contorno los colores que ha detectado, así como también el centro aproximado de ese contorno.

```
while True:
  ret, frame = cap.read()
  if ret==True:
    frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(frameHSV, azulBajo, azulAlto)
    cv2.imshow('maskAzul', mask)
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('s'):
      break
```

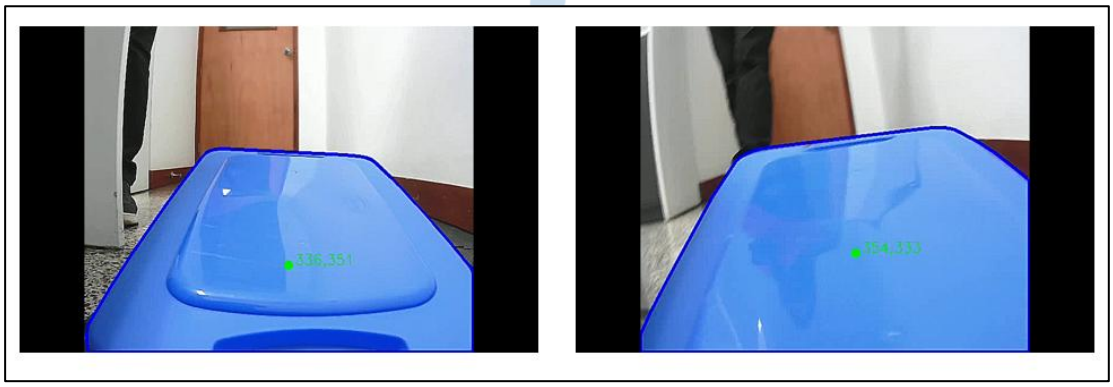
Luego se terminará la iteración cerrando la cámara y cerrando la pestaña donde se trabaja a través del siguiente código.

```
cap.release()
cv2.destroyAllWindows()
```

Con esto, se obtiene los siguientes resultados, como se puede observar en la Figura 48, el robot puede captar superficies de color azul y tratar de obtener su centro de este contorno.

Figura 48

Vista del entorno del prototipo



Para que se aprecie mejor, en la Figura 49 se aprecia donde se encuentra ubicada la cámara y como el robot se traslada sobre una superficie.

Figura 49

Vista externa del robot con su visión

Capítulo 5
Análisis y resultados del modelo

5.1 Modelo Qtable

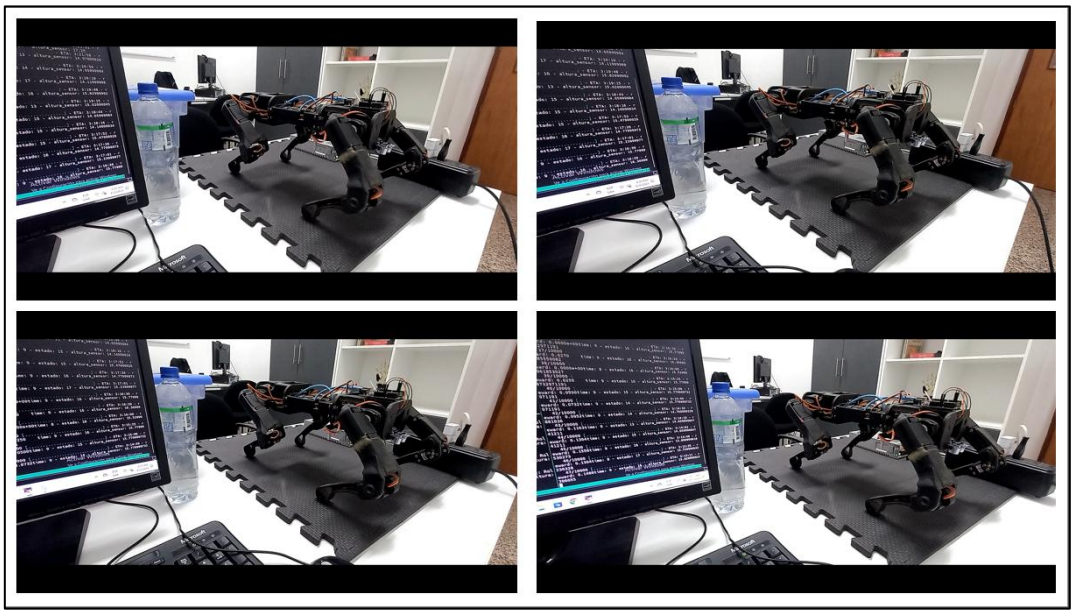
En esta sección por cada episodio se ha ido recolectando el valor de la recompensa y se ha ido promediando para ver cómo ha ido influyendo por cada episodio. Al final, se va a presentar los resultados que se han obtenido en la experimentación aplicando el modelo Qtable, para cada uno de los entornos.

5.1.1 *Modelo Qtable para altura*

En este primer modelo se ha realizado dos experimentos, el primero es insertando a los servomotores una posición, por lo que se espera que el robot se vaya adecuado a la posición que se ha especificado en el entorno, mientras que el segundo experimento es poniendo al robot desde el suelo, y que él vaya intentando llegar a la posición determinada.

Figura 50

Experimento de Qtable para atura 100 episodios desde arriba

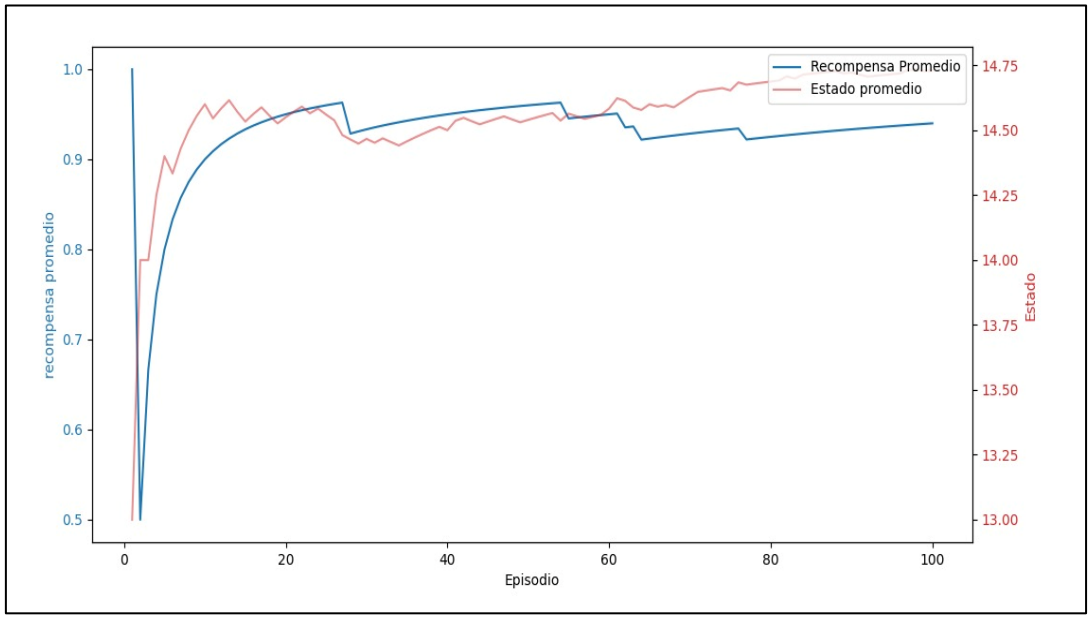


En la figura anterior, se puede observar el entramiento del prototipo, como comienza a cambiar su posición de altura a lo largo del tiempo, hasta que pueda converger en una altura determinada propuesta por el entorno diseñado.

Por lo tanto, para evidenciar como ha ido evolucionando este cambio se va a graficar la recompensa y el estado, que es caso será el valor de la altura. El siguiente grafico muestra como se ha ido comportando el prototipo a lo largo de cada episodio.

Figura 51

Resultado de Qtable para atura 100 episodios desde arriba

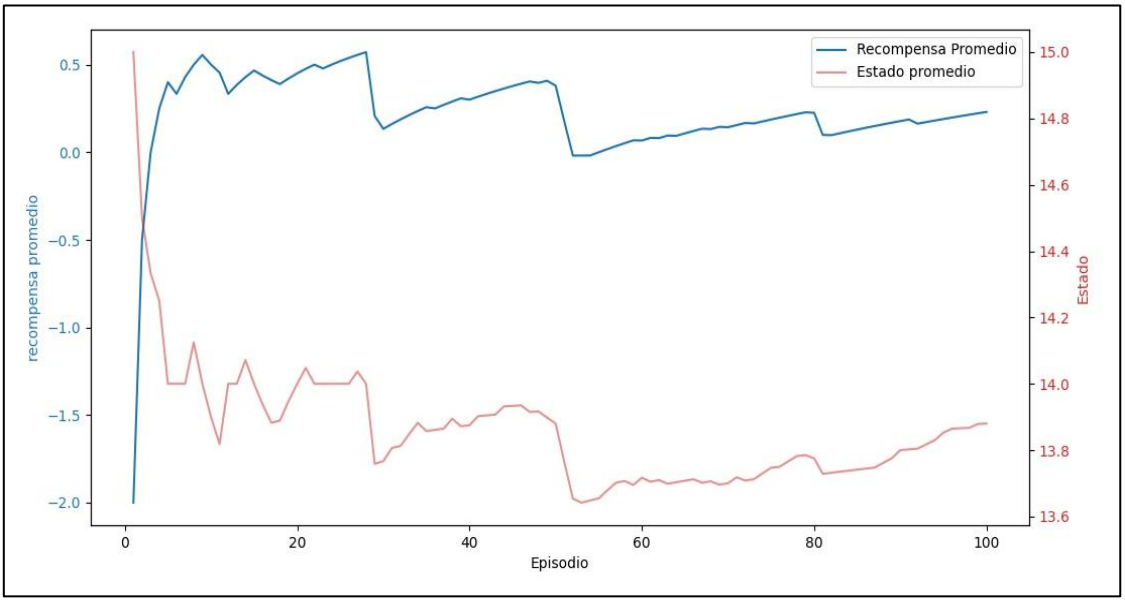


Como se observa en el gráfico, el prototipo va iniciando desde una altura de trece centímetros y va contantemente subiendo, tratando de llegar a su meta por cada episodio, esto se ve representado por la gráfica de color rojo. Cada decisión que va tomando el robot de subir, bajar o quedarse en esa posición, hace que vaya aumentado o disminuyendo la recompensa que viene representada por el color azul. Como se observa, va subiendo constantemente, tendiendo a un valor de aproximadamente 0.95 en su punto más alto.

Entre el episodio veinte y treinta, se ha visto una caída de recompensa debido a que la altura iba cayendo, sin embargo, para los siguientes episodios se iba reponiendo, por lo tanto, cuando existe perturbaciones como en este caso fallo en los motores la superficie cambia, el prototipo busca la forma de poder llegar a su objetivo, a la altura que se ha planteado. El siguiente grafico será del experimento donde el robot ha comenzado desde la parte de abajo, es decir desde el suelo y ha intentado subir hacia la altura determinada.

Figura 52

Resultado de Qtable para atura 100 episodios desde abajo



Como el robot ha comenzado desde una altura baja, los valores de recompensa son bajos, por lo tanto, los valores de recompensan llega como máximo a 0.5 de recompensa, mientras que el valor de la altura llega a 13.8. Hay momentos entre el episodio veinte y treinta que la recompensa decae, similar que, en el experimento anterior, mientras que la altura decae de igual forma. A pesar de ello, se observa que el modelo, sigue tratando de llegar a su altura determinada buscando mayor recompensa.

Este experimento, en comparación al primero, se observa que el prototipo trata de buscar la posición que se le ha planteado en su entorno buscando para que pueda llegar su objetivo. A pesar de que tiene poca recompensa sin embargo es la adecuada para estas condiciones. En el primero, ya ha ido teniendo unas recompensas positivas y lo que buscaba era seguir aumentando, mientras que, en el segundo, comenzaba con recompensas negativas y paulatinamente iba consiguiendo obtener más recompensas subiendo constantemente.

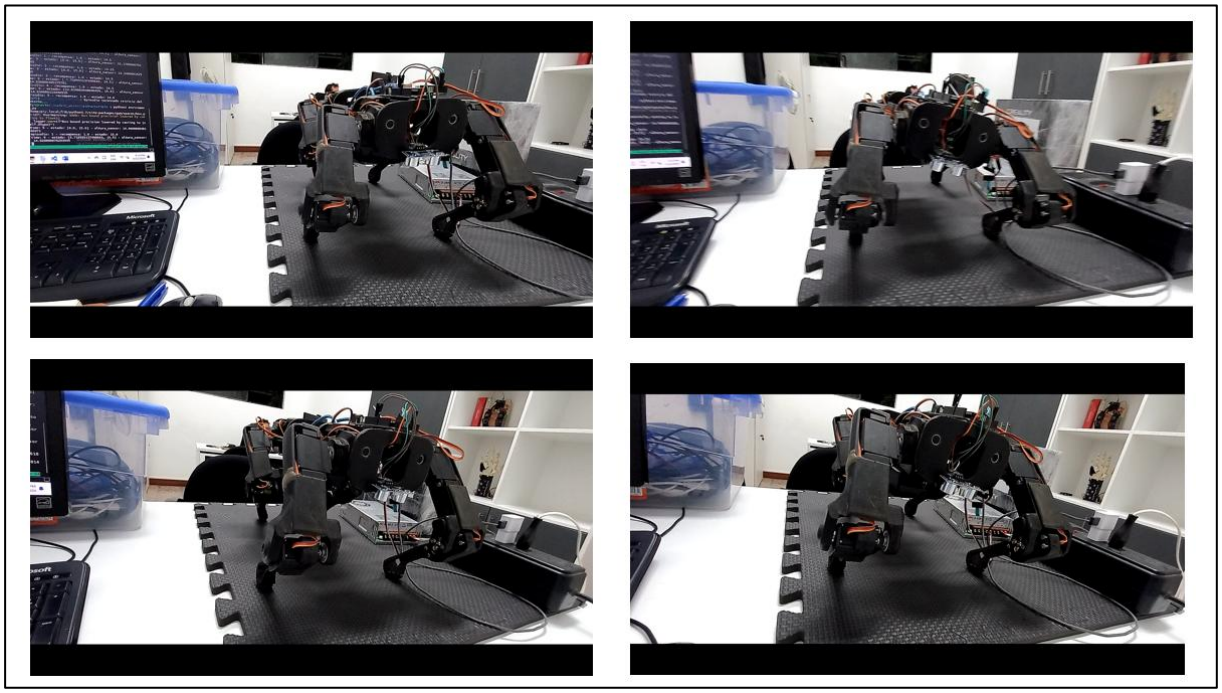
El cambio de recompensa entre los veinte primeros episodios del primer experimento es de aproximadamente 0.45 puntos, mientras que en el segundo ha sido de aproximadamente de 0.7 puntos. Lo que significa que las acciones que ha estado tomando entre ese intervalo han sido las adecuadas para que suba su valor. Por lo que el modelo Qtable, es un modelo adecuado relativamente adecuado.

5.1.2 *Modelo Qtable para Roll-altura*

En este segundo modelo se ha realizado de igual forma dos experimentos, el primero es insertando a los servomotores una posición, por lo que se espera que el robot se vaya adecuado a la posición que se ha especificado en el entorno, mientras que el segundo experimento es poniendo al robot desde el suelo, y que él vaya intentando llegar a la posición determinada.

Figura 53

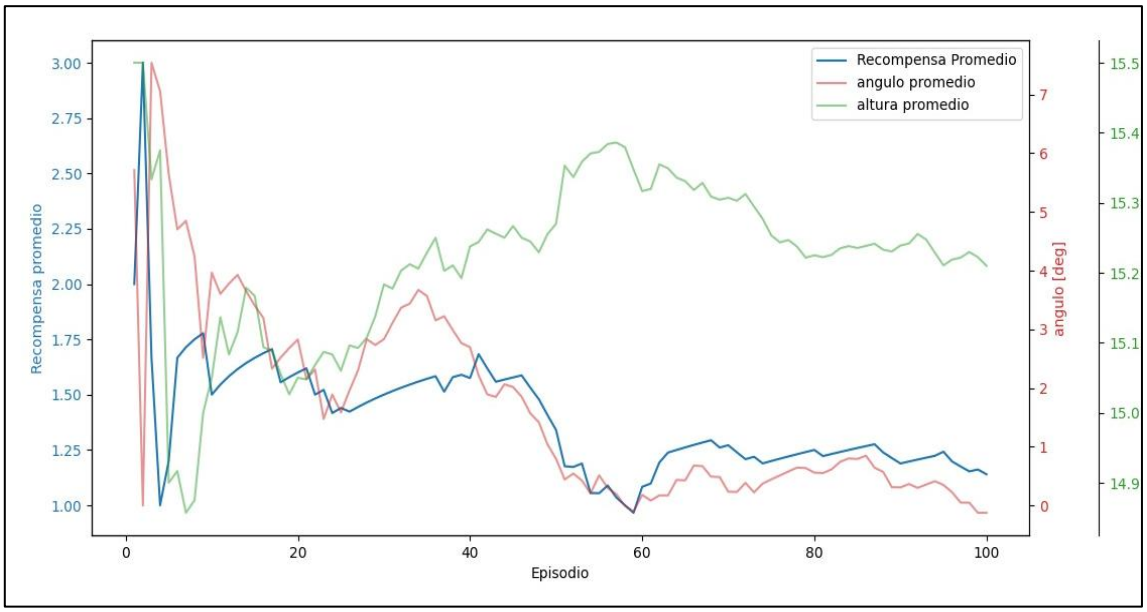
Experimento de Qtable para atura y roll 100 episodios



Por lo tanto, para evidenciar como ha ido evolucionando este cambio se va a graficar la recompensa y el estado, que es caso será el valor de la altura y el ángulo Roll en sexagesimales. El siguiente gráfico muestra como se ha ido comportando el prototipo a lo largo de cada episodio.

Figura 54

Resultado de Qtable para atura y roll 100 episodios desde arriba



Como se observa al iniciar desde una posición, el robot va a contar con una altura y un ángulo determinado, y como se observa la recompensa es alta, sin embargo, el ángulo no es el

adecuado, por lo que la acción que debe tomar es ir decayendo, como se evidencia en los siguientes episodios.

Se logra apreciar que, al ponerle mayor peso al ángulo, el comportamiento del agente tenderá a tomarle mayor importancia, es por eso por lo que, en los siguientes episodios, a pesar de que el prototipo se encuentra a una altura adecuada según el entorno, el ángulo no lo era; lo que causaba que existan cambios.

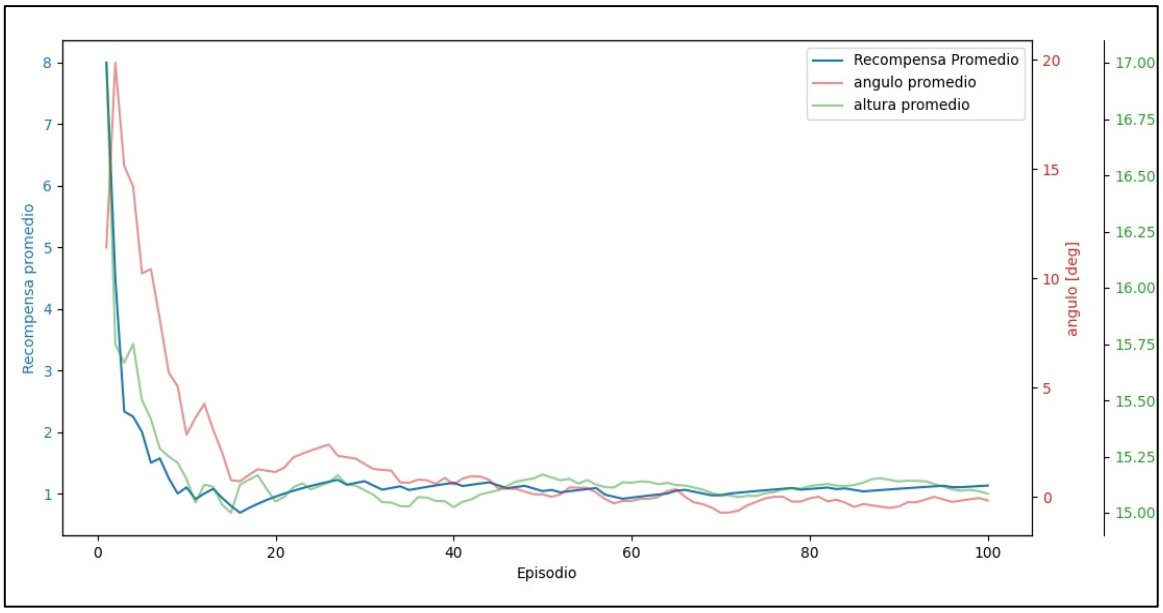
Entre el episodio veinte y cuarenta, el ángulo ha ido decayendo y los resultados de la recompensa han ido bajando de igual forma, debido a que aún no se llegaba al valor del ángulo propuesto en el entorno, sin embargo, la altura seguía incrementando.

A partir del episodio sesenta en adelante, el valor del ángulo ha ido siendo menor, cercanos al objetivo propuesto en el entorno. Por lo que el valor de la recompensa ha ido incrementado paulatinamente.

Para el siguiente experimento, se tendrá en cuenta que se iniciará desde la parte de abajo del prototipo, y constantemente el buscará adaptarse a la posición que se planteó como objetivo en el entorno.

Figura 55

Resultado de Qtable para atura y roll 100 episodios desde abajo



Como se ha observado el comportamiento del robot cuando ha iniciado desde abajo, no ha presentado muchas fluctuaciones en comparación con el experimento anterior. Al inicio ha presentado mayores valores en la recompensa y en los estados, y al ser estos altos, el modelo poco a poco se ha ido adecuando.

Entre los episodios diez y veinte, se ha visto que los valores del ángulo han ido bajando significativamente, al igual que la altura, mientras que los valores de la recompensa oscilan entre el valor uno.

Entre los valores episodios veinte y cien, el dato de la recompensa se ha ido manteniendo constante al igual que el valor de los ángulos y la altura, por lo que se deduce que el modelo ha aprendido de manera adecuada en este experimento.

En este experimento en comparación al primero de este modelo, se ha visto que no tiene más fluctuaciones, se ha comportado de manera adecuada, en comparación experimento anterior, donde al iniciar en una posición cambia sus valores de recompensa. Sin embargo, tanto como en el primero como en el segundo experimento ambos han convergido hacia el valor de recompensa de aproximadamente uno.

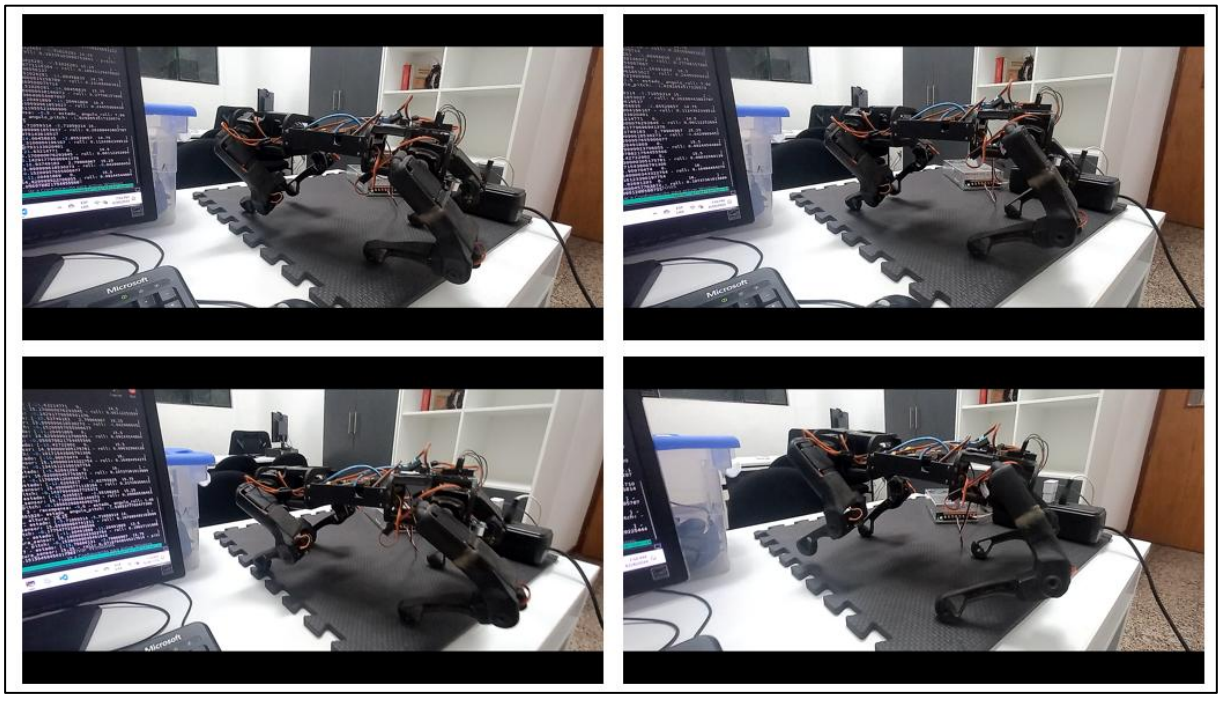
Por la forma en como se ha considera el peso de las recompensas, estos resultados del entorno Roll-Altura en comparación a la Altura, son más significativos pese a la complejidad del entorno diseñado. Al darle valor a los ángulos de inclinación, el prototipo tenderá a tener un mayor control en su estabilidad, es decir que tendrá un mayor control para estar en una pendiente y que los ángulos de posición se mantengan estables. Lo cual lo convierte en un prototipo óptimo para poder movilizar objeto de manera adecuada.

5.1.3 *Modelo Qtable para el ángulo Pitch, Roll y altura*

En este tercer modelo se ha realizado de igual forma dos experimentos, el primero es insertando a los servomotores una posición, por lo que se espera que el robot se vaya adecuado a la posición que se ha especificado en el entorno, mientras que el segundo experimento es poniendo al robot desde el suelo, y que él vaya intentando llegar a la posición determinada.

Figura 56

Experimento de Qtable para atura, pitch y roll 100 episodios desde arriba

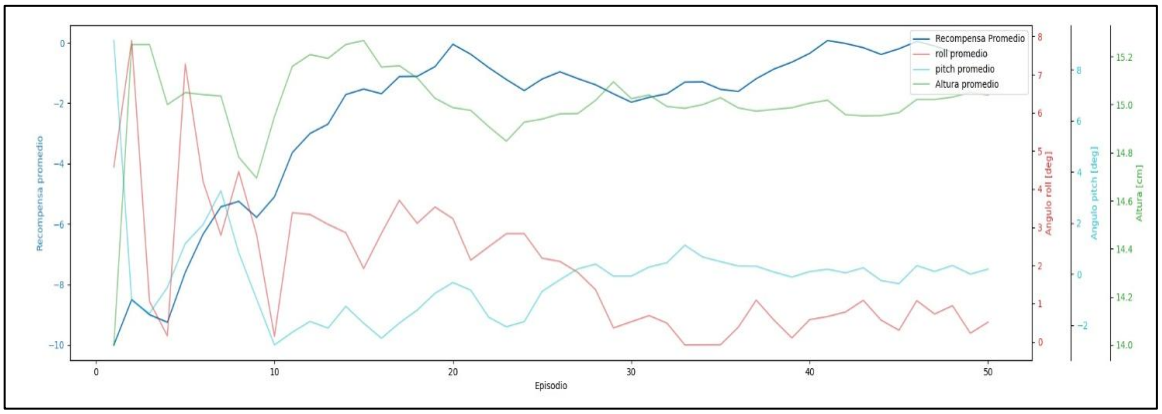


Por lo tanto, para evidenciar como ha ido evolucionando este cambio se va a graficar la recompensa y el estado, que es caso será el valor de la altura, el ángulo Roll y el ángulo Pitch

en sexagesimales. El siguiente gráfico muestra como se ha ido comportando el prototipo a lo largo de cada episodio.

Figura 57

Resultado de Qtable para atura, pitch y roll 50 episodios desde arriba



En este caso gráfico se observa cómo se ha ido comportando el robot a lo largo de cada episodio, los valores de ángulo roll, han ido decayendo hasta llegar a un valor entre 0 y 1, mientras que el pitch tiene valores más cerca al 0. De igual forma, se observa en la altura, cuando el valor de la altura tiende a llegar su valor deseado.

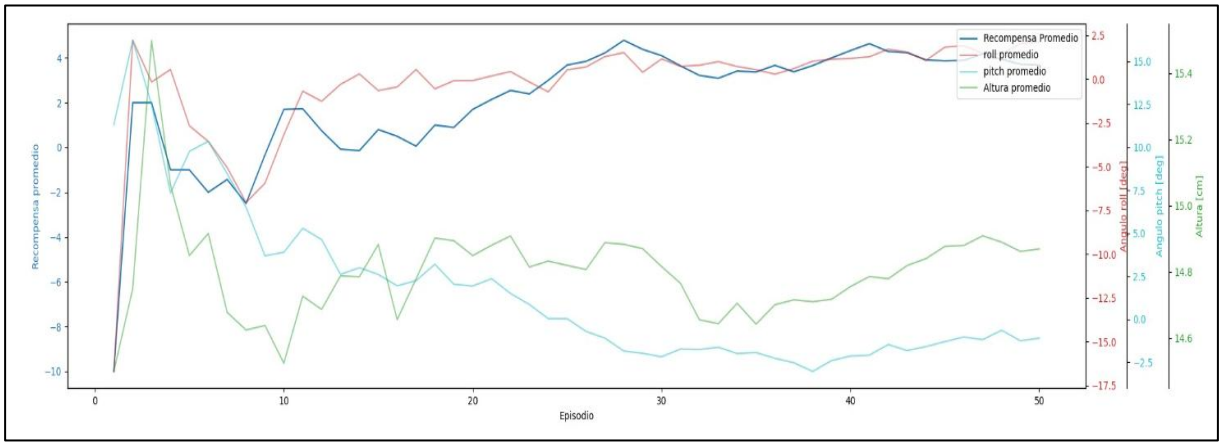
A partir del episodio 20 en adelante, los valores de la recompensa se han ido incrementando constantemente, al inicio de este experimento inicio con un puntaje de -10 y paulatinamente a medida que se iba dando la experimentación, se ha incrementado el valor hasta alrededor de 0. Esto no quiere decir que el modelo no haya sido el adecuado, sino que el modelo ha ido incrementando constante su valor a pesar de que al inicio presentaba valores bajos.

Como se ha podido observa, en este experimento solo se ha considerado 50 episodios a modo de prueba, debido a que cada extremidad se mueve de manera independiente hasta lograr su punto en específico.

Para el siguiente experimento, se tendrá en cuenta que se iniciará desde la parte de abajo del prototipo, y constantemente el buscará adaptarse a la posición que se planteó como objetivo en el entorno, teniendo en cuenta que las extremidades van a ser independientes en su movimiento unas de otras.

Figura 58

Resultado de Qtable para atura, pitch y roll 50 episodios desde abajo



Como se puede observar en el gráfico, los valores de la recompensa han sido mejores que los valores, llegando a puntaje de hasta 4 puntos. Mientras que los valores del ángulo Roll, iba variando conforme se extendía el experimento llegando a valores que oscilaban entre 2.5 y 0. Por otra parte, el valor del ángulo Pitch iba variando desde entre los valores de -0.5 y 0. Los valores de la altura, por su parte, se mantenían dentro del rango de valores que se han propuesto en el entorno.

A diferencia del primer experimento, al comenzar desde la parte de abajo en el prototipo ha implicado que este se encuentre a un aprendizaje más conforme obteniendo buenos resultados, en cuanto a variación de puntaje por episodio, este último experimento ha presentado mayor variación, de hasta 14 puntos de diferencia en comparación al primer experimento.

5.1.4 *Modelo Qtable para traslación*

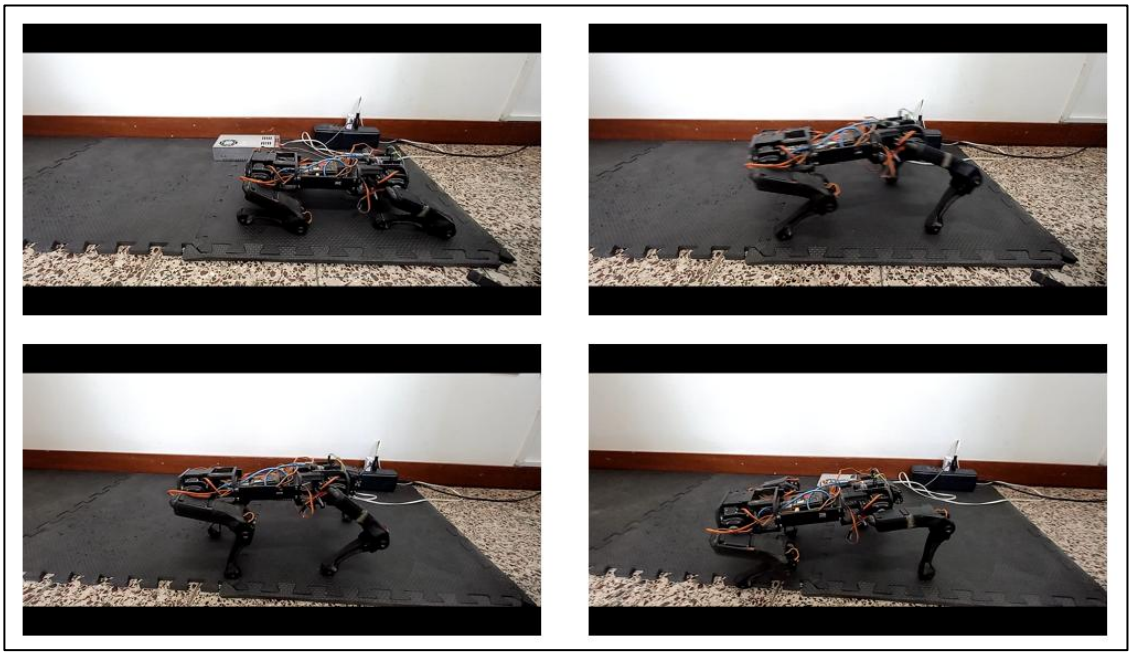
En este cuarto modelo se van a realizar dos experimentos, se ha iniciado desde la parte de abajo del robot por lo que se espera que él se vaya adecuado a la posición que se ha especificado en el entorno, mientras se va trasladando por la superficie. Y el segundo experimento es la traslación del prototipo sobre una superficie inclinada.

Por la forma en cómo se encuentra desarrollado este modelo, solo se va a contar con 50 episodios de entrenamiento. El prototipo se va a ir trasladando por una superficie como se observa en la Figura 59, variando la altura en la que se encuentra, hasta que logre obtener la altura desea por el modelo.

De igual forma se va a evaluar la forma en cómo va variando la recompensa y los estados propuesto por los sensores.

Figura 59

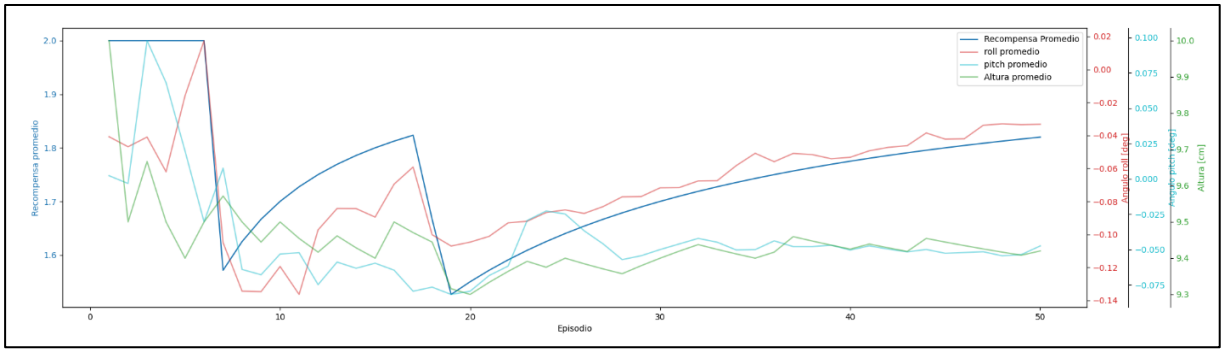
Experimento de Qtable 50 episodios traslación



En la ***Figura 60***, se observa la gráfica que ha venido acompañando al prototipo en este entrenamiento, donde se indica la recompensa, el ángulo roll, pitch y la altura a lo largo de los episodios trabajados.

Figura 60

Resultado de Qtable 50 episodios traslación

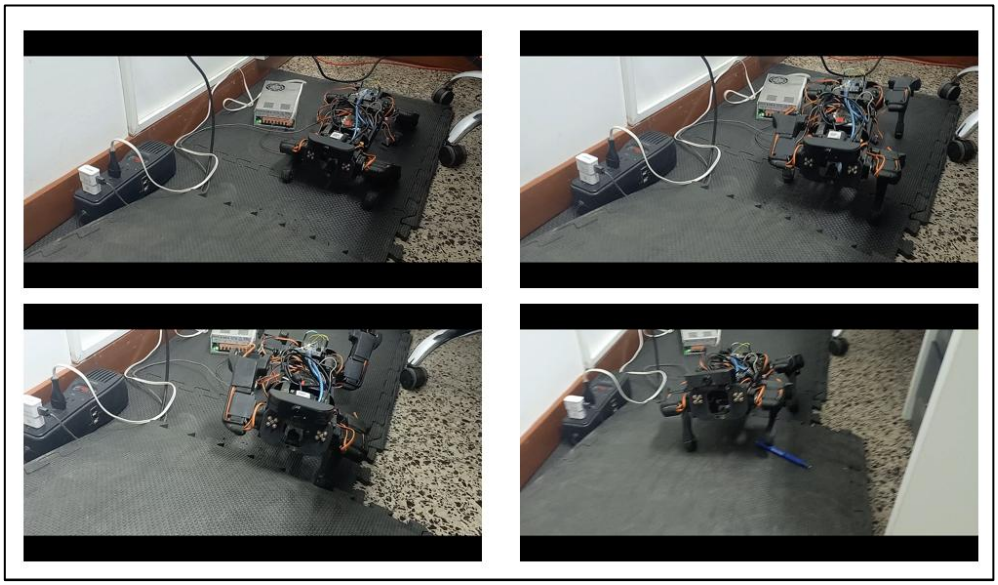


Como se podido observar, la gráfica de recompensa a comparación con los otros modelos no tiene muchas perturbaciones, puesto que el prototipo solo se traslada cambiando la altura y si la altura llega a su meta, sale otro episodio. Los valores de la recompensa ha sido los adecuados, ha ido incrementando, mientras que los valores de la altura, gráfica verde, permanecen en un rango adecuado, mientras que el ángulo pitch, grafica celeste, cambia, pero no tanto como el ángulo roll, grafica roja. Por lo que el prototipo debe mejorar en el entorno o en el movimiento.

En el siguiente experimento se pondrá una superficie inclinada como se observa en la Figura 61 para ver cómo se comporta el prototipo, teniendo en cuenta que el parámetro que influye en sus movimientos es la altura.

Figura 61

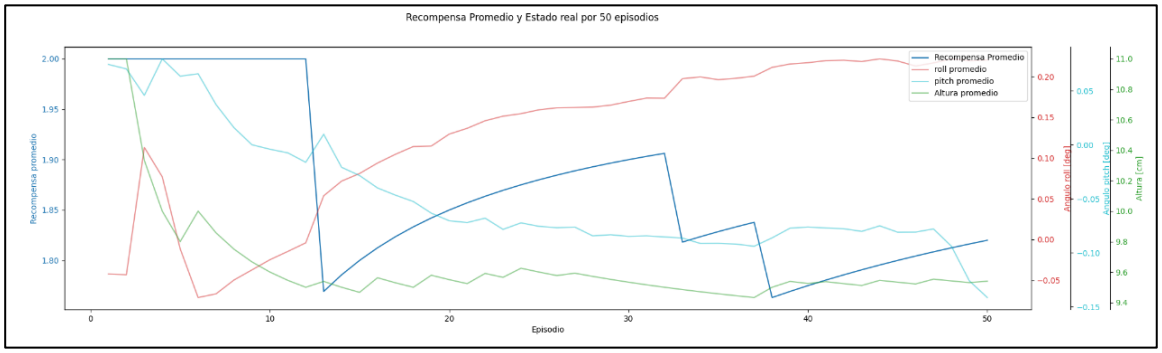
Experimento de resultado de Qtable 50 episodios en superficie inclinada



En la figura Figura 62 se observa la gráfica que ha venido acompañando al prototipo en este entrenamiento, donde se indica la recompensa, el ángulo roll, pitch y la altura a lo largo de los episodios trabajados.

Figura 62

Resultado de Qtable para atura 50 episodios, traslación superficie inclinada



En este experimento se ha podido observa que los resultados de la recompensa han ido variando conforme el prototipo iba subiendo por la superficie, este modelo se debe mejorar, puesto que los valores que se han plasmado se podrían tener mejor, y por cada episodio sea el intento de subir la superficie.

5.2 Modelo Deep Reforcement learning

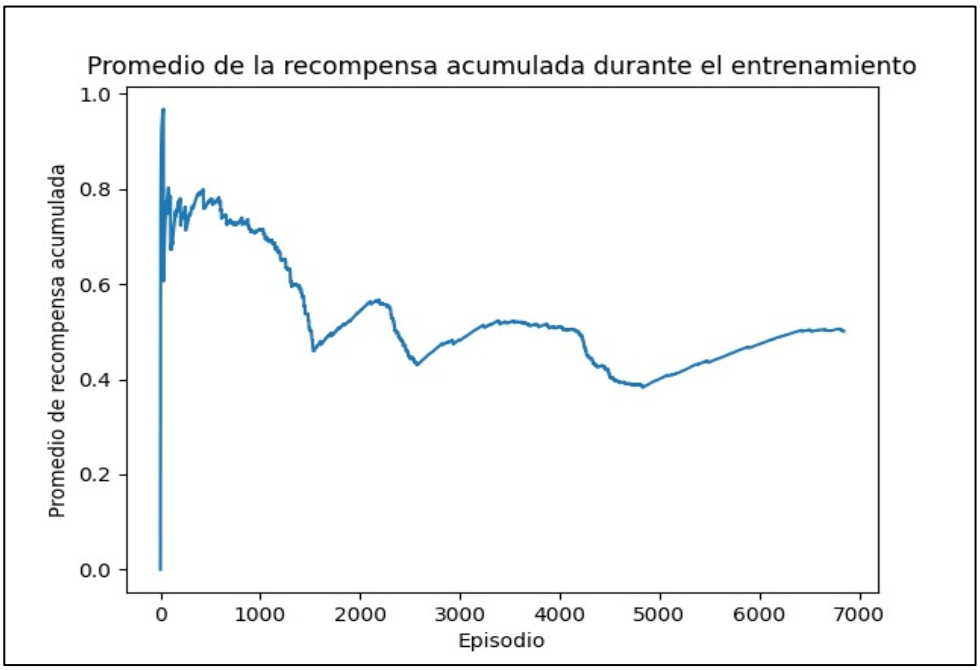
En esta parte se va a desarrollar el analices de los resultados obtenidos de los modelos de redes neuronales que se han diseñado anteriormente, a diferencia de los experimentos realizados con el modelo Qtable se va a seleccionar al segundo experimento, es decir desde la parte de abajo, para ver cómo influye un modelo de redes neuronales y *Deep Learning* a través de los diferentes modelos por medio de sus recompensas.

5.2.1 Modelo DQN para altura

Para este modelo se van a analizar dos resultados, el primero es el entrenamiento del agente, mientras que el segundo será el testeo del entrenamiento. Con esto se podrá analizar cómo ha ido aprendiendo el agente y comprarlo con el testeo. En el siguiente grafico se observa el comportamiento del agente a lo largo de 7 000 episodios.

Figura 63

Resultado de DQN para altura de 7000 episodios de entrenamiento



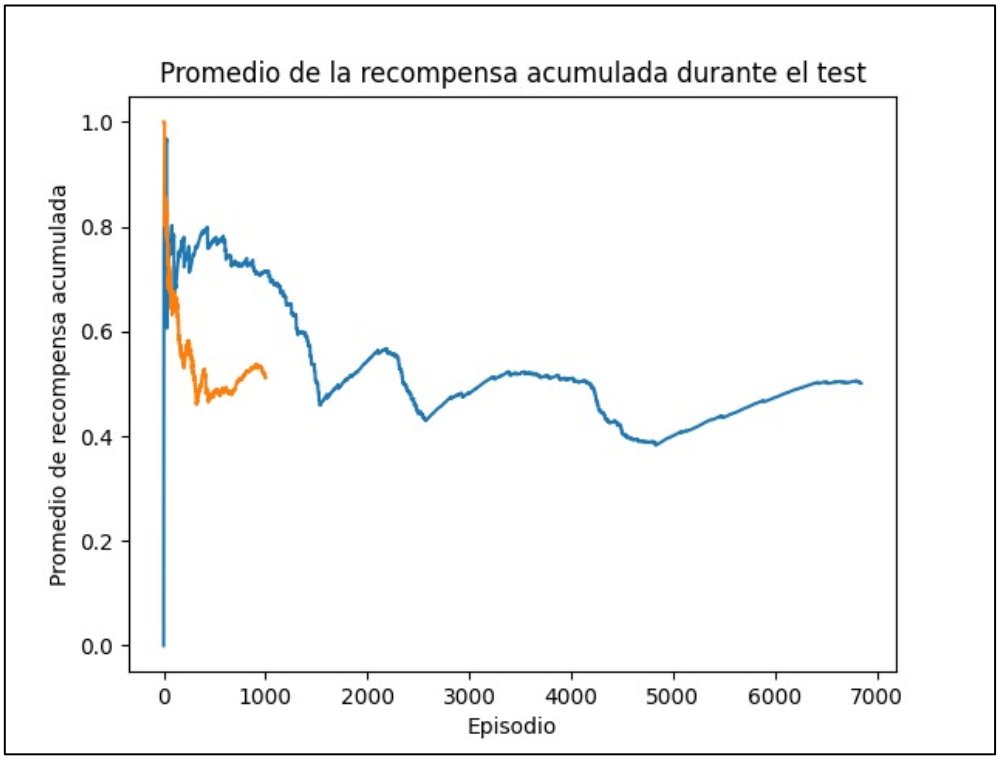
Como se ha observado en el gráfico, la recompensa durante el entramiento es significativamente superior. A pensar que se ha comenzado el prototipo desde abajo, los valores de recompensa han sido entre 0.6 a 0.8 en los primeros 500 episodios. Entre los episodios 1000 y 1500; los valores han ido decayendo, sin embargo, el prototipo se ha ido recuperando en los siguientes 500. Este mismo comportamiento se ha ido presentando a largo del entrenamiento. Se destaca que a partir del episodio 5000, el prototipo ha conseguido presentar un aumento en su recompensa casi de forma constante, lo que implica que las acciones que ha ido tomando han sido las adecuadas, hasta el episodio 7000.

Este comportamiento, se puede deber a que a medida que se vaya realizando un entrenamiento con más episodios, el modelo puede sobre ajustar haciendo la recompensa, o que la red haya tomado decisiones más aleatorias para buscando un mejor rendimiento.

En el siguiente grafico se va a presentar el modelo y los resultados que se ha obtenido con 1000 episodios de testeo, de igual el grafico presenta un promedio de las recompensas que ha ido obteniendo a lo largo de cada episodio.

Figura 64

Resultado de DQN para altura de 1000 episodios de testeo



Como ser observa, el valor al inicio de los episodios de la recompensa en el testo ha sido más alta en comparación al entrenamiento; sin embargo, luego ha ido decayendo, siendo su valor mínimo aproximadamente de 0.5 puntos. Entre el episodio 500 y 1000, se observa que el agente, ha ido incrementado su recompensa significativamente.

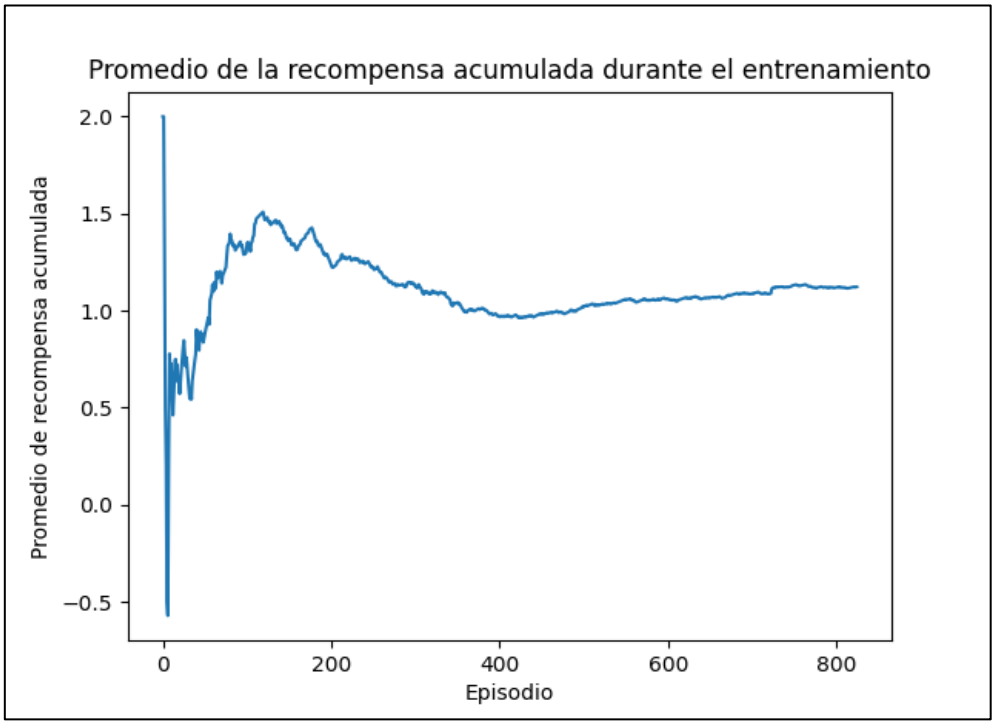
Los resultados obtenidos con este modelo de red neuronal de tres capas presentan valores en su recompensa relativamente mayores con respecto a la recompensa que se ha presentado con el modelo de aprendizaje Qtable. Donde el mínimo valor de este modelo era el mayor valor del modelo con Qtable. Por lo que, si se realiza cambios en la estructura neuronal, se podría llegar a mejores resultados.

5.2.2 Modelo DQN para Roll-Altura

Para este modelo se van a analizar dos resultados, el primero es el entrenamiento del agente, mientras que el segundo será el testeo del entrenamiento como se ha venido trabajando en el modelo anterior. Con esto se podrá analizar cómo ha ido aprendiendo el agente y compararlo con el testeo. El siguiente grafico se observa el comportamiento del agente a lo largo de 800 episodios.

Figura 65

Resultado de DQN para altura y roll de 800 episodios de testeo



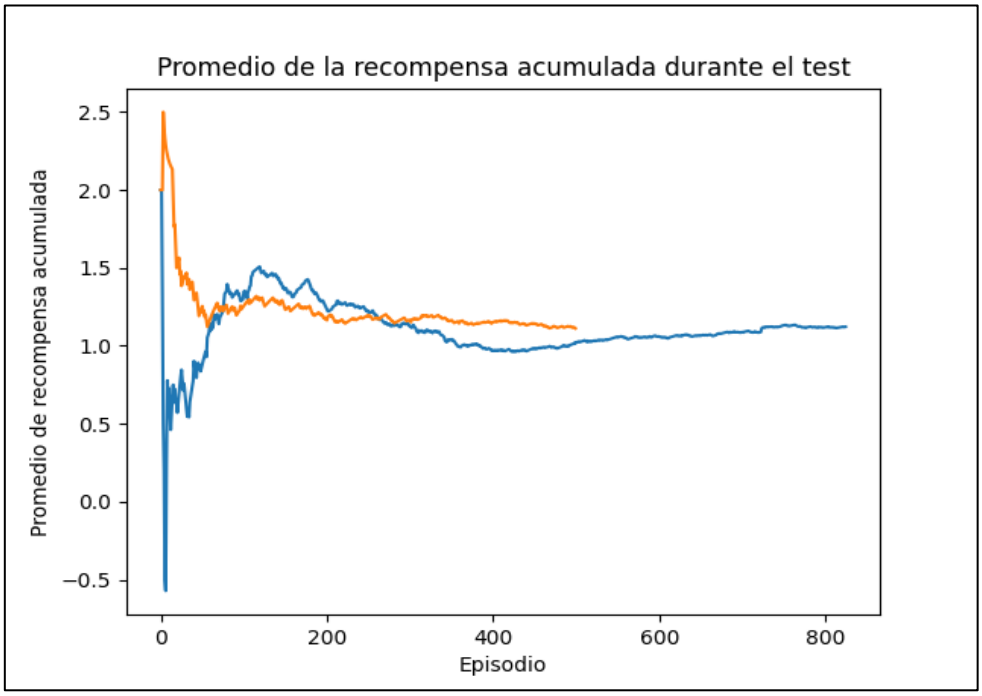
Como se ha podido observar, en los primeros 100 episodios, el prototipo ha ido obteniendo un mayor valor en la recompensa, sin embargo, luego entre el episodio 100 y 200 ha ido decayendo el valor lentamente hasta su valor más bajo en el episodio 400, a partir de este ha ido subiendo constantemente sin ningún tipo de oscilaciones, por lo que el entrenamiento del modelo ha presentado buenos resultados.

Se ha decido disminuir la cantidad episodios de entrenamiento para evitar sobre estimar la red neuronal, por lo que con 800 episodios se ha obtenido buenos resultados en comparación a los 7000 episodios que estuvo trabajando en el modelo anterior.

En el siguiente grafico se va a presentar el modelo y los resultados que se ha obtenido con 500 episodios de testeo, de igual el grafico presenta un promedio de las recompensas que ha ido obteniendo a lo largo de cada episodio. Comparando con el grafico que se ha obtenido en el entrenamiento antes mencionado.

Figura 66

Resultado de DQN para altura y roll de 500 episodios de testeo



Como se ha observado en el gráfico, los valores de testeo del modelo convergen de manera más significativa; entre los 100 y 200 episodios, todavía presentaba oscilaciones de pero no tantas como se ha presentó en el entrenamiento, como se ha visto ese valor tienden al valor que se ha ido proyectando en la fase de entrenamiento. Por lo que el modelo con tres capas de 24 neuronas cada uno, ha obtenido resultados relativamente mejores a los que se ha trabajado en el modelo anterior.

Comparando con el modelo Qtable, el entrenamiento por *Deep Learning* es relativamente mejor, los resultados no son tan dispersos como en el primer experimento con la Qtable y más constante como el segundo experimento. Estas capas se pueden modificar, cambiando los valores de neuronas y las funciones de activación.

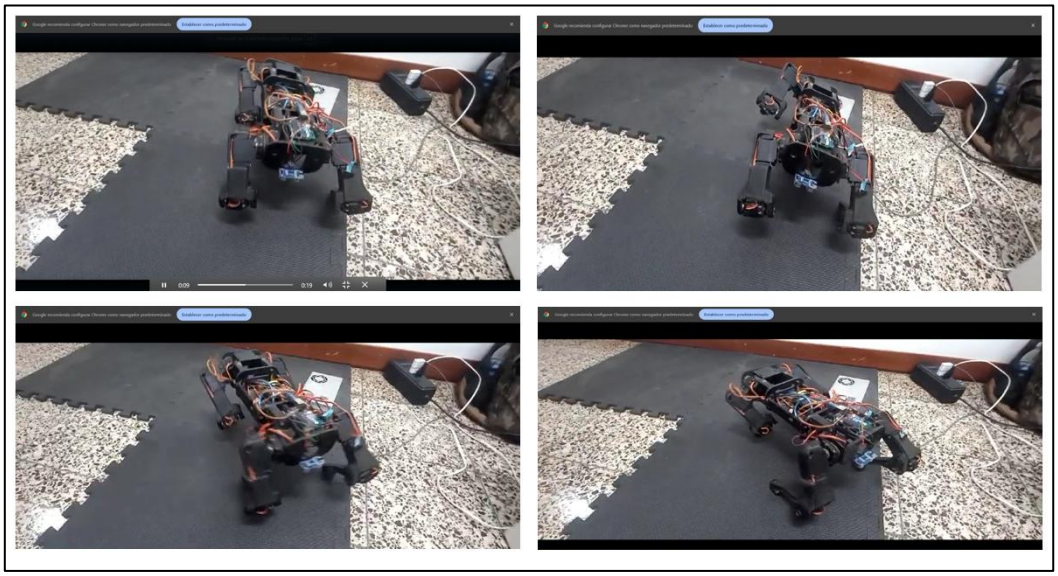
En comparación con el modelo de altura, este modelo con el ángulo Roll y altura, da valores más altos en cuanto a recompensa, sin embargo, esto se debe a que el entorno ha considerado a otro parámetro con mayor peso. A pesar de ello, se destaca que el modelo, teniendo en cuenta que los parámetros han aumentado, pueda seguir el valor adecuado para el control de los servomotores y con ello, la posición del robot.

5.2.3 Modelo DQN para el ángulo Pitch, Roll y Altura

Para este modelo se van a analizar dos resultados, el primero es el entrenamiento del agente, mientras que el segundo será el testeo del entrenamiento como se ha venido trabajando en el modelo anterior. Con esto se podrá analizar cómo ha ido aprendiendo el agente y compararlo con el testeo. El siguiente grafico se observa el comportamiento del agente a lo largo de 800 episodios.

Figura 67

Experimento de resultado de DQN para pitch, roll y altura

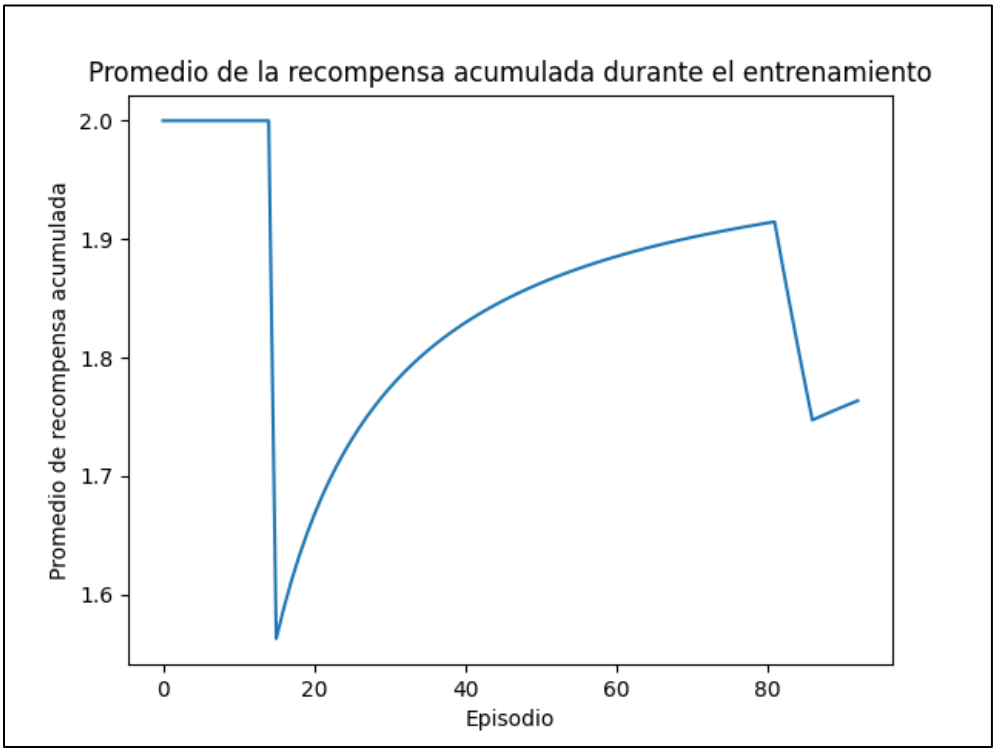


5.2.4 Modelo DQN de traslación

Para este modelo se van a analizar dos resultados, el primero es el entrenamiento del agente, mientras que el segundo será el testeo del entrenamiento como se ha venido trabajando. Con esto se podrá analizar cómo ha ido aprendiendo el agente y compararlo con el aprendizaje de su testeo. Para este experimento se ha tenido en cuenta entrenarlo en 100 episodios, con el fin de ver cómo se comporta a lo largo de este.

Figura 68

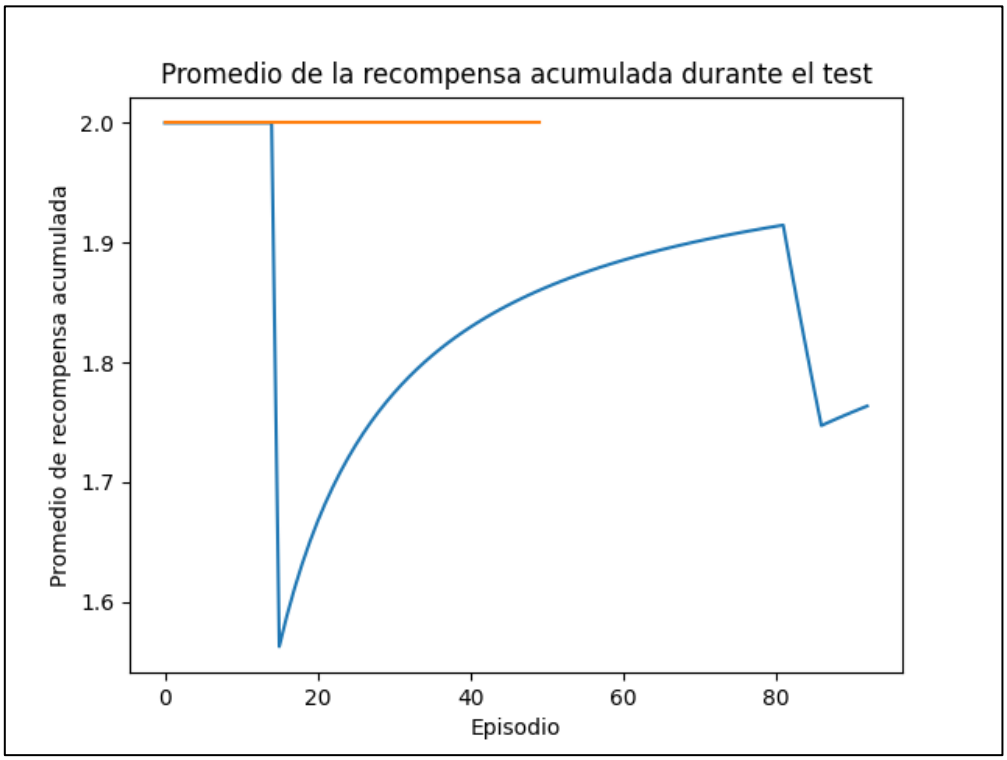
Resultado de DQN para traslación 100 episodios de entrenamiento



Los resultados implican que el comportamiento del robot tiende a ser ideal a medida que este va caminando, sin embargo, la superficie por donde se ha traslado no es lo suficiente para el prototipo. Entre el episodio 10 y el 80 ha ido incrementando constantemente, como se puede observar no se muestra un cambio significativo en cuando los sensores llegan a su valor, adecuado.

Figura 69

Resultado de DQN traslación 50 episodios testeo



Por otra parte, en el testeo se observa que, durante los episodios de testeo, la recompensa se mantiene constante en 2, por lo que se podría asegurar que el entrenamiento ha sido el adecuado.

Conclusiones

El sistema Robot Operating System (ROS), ha sido de mayor importancia para el desarrollo de esta investigación, al tener la facilidad de poder diseñar nodos en los cuales se pueda tener la lógica de programación por separado, ayuda a que no se tenga en un solo script todo el algoritmo implementado. La forma en cómo se obtiene los ángulos de navegación, a través de un bucle While, y estos pasarlos hacia el otro nodo, donde se emplean en el entorno y dependiendo de ello puedan enviar la información necesaria hacia el otro nodo de servomotores. Ha reducido la cantidad de líneas de código y se ha hecho más eficiente en su uso.

El modelo Qtable para altura, realizando los experimentos cuando el prototipo está desde abajo y cuando está desde arriba han presentado más cambios, pero tanto como los que se ha experimentado implementado el modelo *Deep Reforcement Learning* (DQN), el cual presento mejores resultados que la Qtable, tanto en el entrenamiento como en el testeo, por lo que al trabajar con algoritmo de *Machine Learning* para un modelo de un solo estado, ha sido de mucha importancia para el control de este prototipo.

El segundo entorno donde se tomó en cuenta los parámetros de altura y ángulo roll; el modelo de DQN ha presentado mejores resultados en comparación con los de Qtable, a pesar de que se ha mantenido la estructura neuronal con la que se había trabajado el modelo anterior, por lo que, si se desea cambiar o tener un mejor resultado, se debe realizar mejoras en esto.

El tercer entorno al ser un entorno con mayor complejidad, el modelo de Qtable que se ha aplicado es diferente por lo que se tiene que modificar los parámetros en el aprendizaje. Sin embargo, con la ayuda del DQN este entorno ha presentado mejoras significativas a tal punto de poder realizar movimientos fuera de los parámetros de aprendizaje como intentar trasladarse sobre la superficie.

El cuarto entorno enfocado en la traslación del prototipo se ha evidenciado un mejor desempeño de la red neuronal por aprendizaje por refuerzo en comparación con el modelo trabajado por Qtable. El prototipo se ha trasladado de manera adecuada, tanto en una superficie completamente horizontal e intenta hacerlo en una superficie inclinada, haciendo que este sea óptimo para realizar una traslación sobre superficies irregulares para su fin último que sirva como búsqueda y rescate.

Se puede llegar a la conclusión que con los modelos de aprendizaje supervisado que se han estado entrenando al prototipo añadido a la visión artificial, es un candidato óptimo para realizar actividades de búsqueda y rescate, puesto que tendría la capacidad de trasladarse por terrenos no uniformes y siendo capaz de recolectar información por medio de una cámara.

Referencias

Boden, M. A. (2017). *Inteligencia Artificial*. Turner. https://books.google.es/books?hl=es&lr=&id=LCnYDwAAQBAJ&oi=fnd&pg=PT3&dq=inteligencia+artificial&ots=dsQmwWaKo4&sig=AnT0YRvi_yiwb3pR9K-B4CyBIO4#v=onepage&q=inteligencia%20artificial&f=false

Figure 10: (A) Pitch, yaw and roll angles of an aircraft with body... (s. f.). ResearchGate. Recuperado 11 de junio de 2024, de https://www.researchgate.net/figure/a-Pitch-yaw-and-roll-angles-of-an-aircraft-with-body-orientation-O-u-v-original_fig7_348803228

Francois-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An Introduction to Deep Reinforcement Learning. *Foundations and Trends® in Machine Learning*, *11*(3-4), 219-354. https://doi.org/10.1561/2200000071

Gonzalez, J. L. (2020, julio 13). Tipos de aprendizaje automático. *SoldAI*. https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2019). *Deep Reinforcement Learning that Matters* (arXiv:1709.06560). arXiv. https://doi.org/10.48550/arXiv.1709.06560

IBM. (2023, mayo 4). *¿Qué es el aprendizaje no supervisado? | IBM*. https://www.ibm.com/es-es/topics/unsupervised-learning

IBM. (2024, mayo 10). *¿Qué es el aprendizaje supervisado? | IBM*. https://www.ibm.com/es-es/topics/supervised-learning

Julio Cesar Ponce Gallegos, Aurora Torres Soto, Fátima Sayuri Quezada Aguilera, & Antonio Silva Sprock. (2014). *Inteligencia Artificial*. ProyectoLATIn. https://rephip.unr.edu.ar/server/api/core/bitstreams/bb5e5b0c-01b6-482c-a3a4-a469f994c92b/content

LASSE ROUHIAINEN. (2018). Inteligencia artificial_FIN.indd. *Nombre del 2018*, 22.

Navio2 – autopilot HAT for Raspberry Pi. (s. f.). Recuperado 24 de marzo de 2024, de https://navio2.hipi.io/

Quigley, M., Gerkey, B., & Smart, W. D. (2015). *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media, Inc.

Raspberry Pi configuration | Navio2. (s. f.). Recuperado 13 de junio de 2024, de https://emlid.com//navio2/configuring-raspberry-pi/

Sensor Ultrasonido HC-SR04. (s. f.). Naylamp Mechatronics - Perú. Recuperado 11 de junio de 2024, de https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html

Servo MG946R 13kg—Tower Pro Original. (s. f.). Naylamp Mechatronics - Perú. Recuperado 11 de junio de 2024, de https://naylampmechatronics.com/servomotores/23-servo-mg946r-13kg.html

Servomotor MG996R. (s. f.). Electronicos Caldas. Recuperado 12 de junio de 2024, de https://www.electronicoscaldas.com/es/motores-y-servos/546-servo-motor-mg996r.html

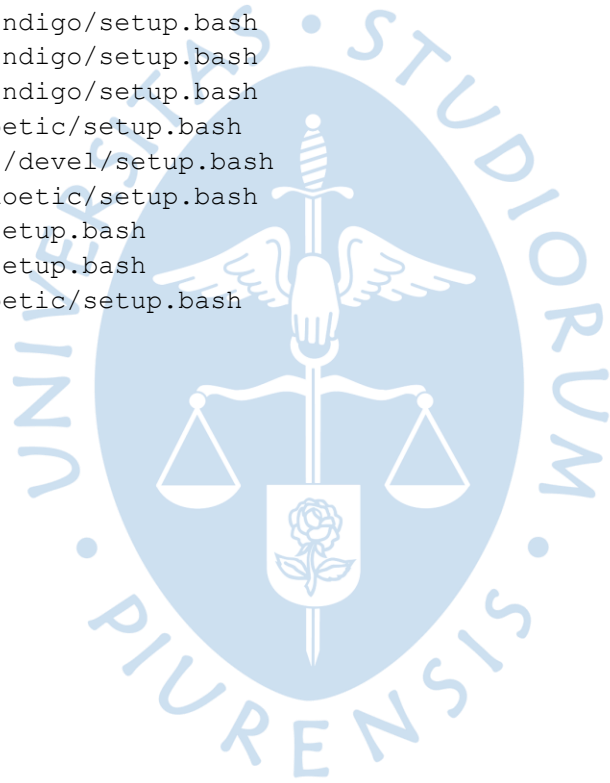
Apéndices

Apéndice A. código de ROS Setup.bash

```
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# If set, the pattern "**" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
#shopt -s globstar

# make less more friendly for non-text input files, see lesspipe(1)
#[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
    xterm-color|*-256color) color_prompt=yes;;
esac

# uncomment for a colored prompt, if the terminal has the capability; turned
# off by default to not distract the user: the focus in a terminal window
# should be on the output of commands, not on the prompt
force_color_prompt=yes

if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null; then
```

```
      # We have color support; assume it's compliant with Ecma-48
      # (ISO/IEC-6429). (Lack of such support is extremely rare, and such
      # a case would tend to support setf rather than setaf.)
      color_prompt=yes
    else
      color_prompt=
    fi
fi

if [ "$color_prompt" = yes ]; then

PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\
033[01;34m\]\w \$\[\033[00m\] '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
    ;;
*)
    ;;
esac

# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval
"$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

# colored GCC warnings and errors
#export
GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote
=01'

# some more ls aliases
#alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
```

```
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
#source /opt/ros/indigo/setup.bash
#source /opt/ros/indigo/setup.bash
#source /opt/ros/indigo/setup.bash
source /opt/ros/noetic/setup.bash
source ~/catkin_ws/devel/setup.bash
#source /opt/ros/noetic/setup.bash
#/opt/ros/noetic/setup.bash
#/opt/ros/noetic/setup.bash
source /opt/ros/noetic/setup.bash
```



Apéndice B. Código de programación

LECTURA_DE_ALTURA.ino

```
const int Trigger = 2;
const int Echo = 3;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(Trigger, OUTPUT);
  pinMode(Echo, INPUT);
  digitalWrite(Trigger, LOW);
}

void loop() {
  // put your main code here, to run repeatedly:
  float t;
  float d;

  digitalWrite(Trigger, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trigger, LOW);

  t = pulseIn(Echo, HIGH);
  d = t/59;

  Serial.println(d);
  delay(100);

}
```

Sensors.py

```
"----------Librerías para dirección----------"
import sys
import os

"----------Cambio dirección----------"
current_directory = os.path.dirname(os.path.abspath(__file__))
navio2_path                                                              =
os.path.join(current_directory,'..','..','..','..','Navio2','Python')
sys.path.append(navio2_path)

"----------Librerías para programa----------"
import time
import math
import serial
import rospy
from std_msgs.msg import String
from std_msgs.msg import Float64
from std_msgs.msg import Int32
from std_msgs.msg import Float32MultiArray
from navio import util
from navio import mpu9250
from navio import lsm9ds1


PuertoSerie =  serial.Serial('/dev/ttyACM0',9600)

rospy.init_node('sensors',anonymous = True)
pub = rospy.Publisher('sensors_topic',Float32MultiArray,queue_size = 10)

"----------Función de ángulos----------"
def angulos(a,b,c):
      r = math.sqrt(b**2 + c**2)
      d = math.atan2(a,r)*180.0/math.pi
      return d
def cos(a):
      b = math.cos(a)
      return b
def sen(a):
      b = math.sin(a)
      return b

"----------Main----------"
def sensors():
      util.check_apm()
      imu = mpu9250.MPU9250()
      imu2 = lsm9ds1.LSM9DS1()
      imu.initialize()
      imu2.initialize()

      #time.sleep(1)
```

```
	rate = rospy.Rate(50)

	p_ang_prev = 0
	r_ang_prev = 0

	count = 0

	while count <= 20:

		sArduino = PuertoSerie.readline()
		m9a,m9g,m9m = imu.getMotion9()
		l9a,l9g,l9m = imu2.getMotion9()

		tiempo_prev = time.time()
		dt = (time.time() - tiempo_prev)

		ax = m9a[0]
		ay = m9a[1]
		az = m9a[2]

		gx = m9g[0]
		gy = m9g[1]
		gz = m9g[2]

		mx = l9m[0]
		my = l9m[1]
		mz = l9m[2]


		"----------ROLL----------"
		roll = angulos(ax,ay,az)
		roll_ang = 0.98*(r_ang_prev + gy*dt) + 0.02*roll
		roll_f = "{:.3f}".format(roll_ang)
		roll_str = "Roll: %s " % roll_f

		"----------PITCH----------"
		pitch = angulos(ay,ax,az)
		pitch_ang = 0.98*(p_ang_prev + gx*dt) + 0.02*pitch
		pitch_f = "{:.3f}".format(pitch_ang)
		pitch_str = "Pitch: %s " % pitch_f

		"----------ALTURA----------"
		alt = sArduino.decode('utf-8')
		alt_str = "Altura: %s" % alt
		altura = float(alt)
		array_msg = Float32MultiArray()

		sensors_data =[roll_ang,pitch_ang,altura,count]

		array_msg.data = sensors_data

		rospy.loginfo(roll_str + pitch_str + ' '+ alt_str+ 'iter:
'+str(count))
```

```
            pub.publish(array_msg)

            count += 1

            rate.sleep()


if __name__== '__main__':

      try:
            sensors()
      except rospy.ROSInterruptException:
            pass
```

Servo.py

```
import sys
import rospy
import os

current_directory = os.path.dirname(os.path.abspath(__file__))
navio2_path                                                                    =
os.path.join(current_directory,'..','..','..','..','Navio2','Python')
sys.path.append(navio2_path)

import time
from navio import pwm
from navio import util
import rospy
import signal
from std_msgs.msg import Float64
from std_msgs.msg import Float32MultiArray


current_data = None

def callback(msg):
      global current_data
      current_data = msg.data

def servo_node():
      rospy.init_node('servo')
      sub = rospy.Subscriber('controlador',Float32MultiArray,callback)

      util.check_apm()

      EFI3 = 10
      EFD3 = 2
      EPD3 = 9
      EPI3 = 3

      EFD2 = 11
      EFI2 = 1
      EPD2 = 8
      EPI2 = 4


      with pwm.PWM(EFI3) as SEFI3, pwm.PWM(EFD3) as SEFD3,pwm.PWM(EPI3) as
SEPI3,pwm.PWM(EPD3) as SEPD3, pwm.PWM(EFI2) as SEFI2,pwm.PWM(EFD2) as
SEFD2,pwm.PWM(EPI2) as SEPI2,pwm.PWM(EPD2) as SEPD2:
            SEFD3.set_period(50)
            SEFI3.set_period(50)
            SEPD3.set_period(50)
            SEPI3.set_period(50)

            SEPI2.set_period(50)
            SEPD2.set_period(50)
```

```
            SEFI2.set_period(50)
            SEFD2.set_period(50)


            SEFI3.enable()
            SEFD3.enable()
            SEPI3.enable()
            SEPD3.enable()

            SEFD2.enable()
            SEPD2.enable()
            SEPI2.enable()
            SEFI2.enable()

            while not rospy.is_shutdown():
                  if current_data is not None:
                        SEFI3.set_duty_cycle(current_data[0])
                        SEFD3.set_duty_cycle(current_data[1])
                        SEPD3.set_duty_cycle(current_data[2])
                        SEPI3.set_duty_cycle(current_data[3])

                        SEPI2.set_duty_cycle(current_data[4])
                        SEFI2.set_duty_cycle(current_data[5])
                        SEPD2.set_duty_cycle(current_data[6])
                        SEFD2.set_duty_cycle(current_data[7])

                        print(current_data)
                  #time.sleep(1)
            #rospy.spin()


if __name__== '__main__':
      try:
            servo_node()
      except rospy.ROSInterruptException:
            pass
```

Enviroment.py

```
#!/usr/bin/env python
"----------Librerias para ROS----------"

import rospy
from std_msgs.msg import String
from std_msgs.msg import Int32MultiArray
from std_msgs.msg import Float32MultiArray

"------Librerias para el programa------"

from random import randint
import gym
from gym import spaces
from gym.spaces import Discrete,Box
import numpy as np
import matplotlib.pyplot as plt
import random
import math
import threading
import time
import itertools

from contextlib import redirect_stdout


"------Librerias para la red neuronal------"
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model

#from tensorflow.keras.optimizers.legacy import Adam

"------Librerias para reforcement learning ------"
from rl.agents import DQNAgent
from rl.policy import BoltzmannQPolicy
from rl.memory import SequentialMemory

"-----------------GYM ENVIROMENT-----------------"
class Enviroment:
    def __init__(self,receiver_node):

        self.combinaciones = list(itertools.product([0, 1, 2], repeat=1))
        self.action_enviroment  =  [list(combinacion)  for  combinacion  in
self.combinaciones]
        self.receiver_node = receiver_node
        self.action_space = Discrete(3)
        self.observation_space = Box(low=np.array([-180.0,-180.0,0.0]), high
= np.array([180.0,180.0,100.0]) )
        while self.receiver_node.get_received_data() is None:
```

```
            time.sleep(1)
        data = self.receiver_node.get_received_data()
        self.roll = data[0]
        self.pitch = data[1]
        self.altura = int(data[2])
        if (self.altura >= 10) and self.altura <= 17:
            self.altura = self.altura
        else:
            self.altura = 10
        #self.altura = 15 + random.randint(-1,1)

        self.state = np.array([self.altura,self.roll,self.pitch])

        #self.timer_length = 10
        self.pub    =    rospy.Publisher('controlador',    Int32MultiArray,
queue_size=10)
        #self.rate = rospy.Rate(1)

    def step(self,action):

        self.altura += self.action_enviroment[action][0] -1

        #self.timer_length -= 1

        while self.receiver_node.get_received_data() is None:
            time.sleep(1)
        data = self.receiver_node.get_received_data()

        altura = data[2]+2.0
        self.pitch =  data[1]
        self.roll = data[0]

        #self.state = np.array([self.altura1,self.altura2])

        if (altura >= 14.0  and altura <= 16.0) :
            reward = 1

        if (self.roll >= -0.5  and self.roll <= 0.5) :
            reward = 2
        if (self.pitch >= -0.5  and self.pitch <= 0.5) :
            reward = 2
        else:
            reward = -1

        if ((altura >= 14.0  and altura <= 16.0) or ((self.roll >= -0.5  and
self.roll <= 0.5) or  (self.pitch >= -0.5  and self.pitch <= 0.5)) ) :
            done =True
            print(" - estado: {} - altura_sensor: {} - roll: {} - pitch:
{}".format(self.state,altura,self.roll,self.pitch) )
            #self.send_to_servo(self.state)
        else:
            done = False
```

```
            print(" - estado: {} - altura_sensor: {} - roll: {} - pitch:
{}".format(self.state,altura,self.roll,self.pitch) )

        """


        if ((altura >= 14.0  and altura <= 16.0) and (self.roll >= 0.0  and
self.roll  <=  0.5)and  (self.pitch  >=  0.0   and  self.pitch  <=  0.5)  )  or
(self.timer_length <= 0) :
            done =True
            print(" time: {} - estado: {} - altura_sensor: {} - roll: {} -
pitch:  {}".format(self.timer_length,self.state,altura,self.roll,self.pitch)
)
            #self.send_to_servo(self.state)
        else:
            done = False
            print(" time: {} - estado: {} - altura_sensor: {} - roll: {} -
pitch:  {}".format(self.timer_length,self.state,altura,self.roll,self.pitch)
)
            #self.send_to_servo(self.state)
        """
        #self.state += random.randint(-1,1)
        #self.send_to_servo(self.state)
        self.state = np.array([self.altura,self.roll,self.pitch])
        info = {}
        time.sleep(0.05)

        array_msg = Int32MultiArray()
        altura_data = np.array([self.altura])
        array_msg.data = altura_data
        self.pub.publish(array_msg)

        return self.state, reward, done, info

    def reset(self):


        #self.altura = 15 + random.randint(-1,1)

        while self.receiver_node.get_received_data() is None:
            time.sleep(1)
        data = self.receiver_node.get_received_data()

        self.roll = data[0]
        self.pitch = data[1]
        self.altura = int(data[2])
        if (self.altura >= 10) and self.altura <= 17:
            self.altura = self.altura
        else:
            self.altura = 10

        self.state = np.array([self.altura,self.roll,self.pitch])
```

```
        #self.timer_length = 10
        array_msg = Int32MultiArray()
        array_msg.data = np.array([self.altura])
        self.pub.publish(array_msg)

        return self.state

    def discretizar(self,valor):
        aux                              =                              ((valor-
self.observation_space.low)/(self.observation_space.high-
self.observation_space.low))*20
        return tuple(aux.astype(np.int32))



    "----------------- Funcion run de Qtable -------------------------- "

    def run(self):

        q_table = np.random.uniform(low = -1, high = 1, size = [20,20,20,3])

        tasa_aprendizaje = 0.1
        factor_descuento = 0.95
        episodios = 50

        listado_recompensa = []
        listado_recompensa_mean = []

        listado_angulo_roll = []
        listado_angulo_roll_mean = []

        listado_angulo_pitch = []
        listado_angulo_pitch_mean = []

        listado_altura = []
        listado_altura_mean = []


        for episodio in range(episodios):
            estado = self.discretizar(self.reset())
            final = False
            recompensa_total = 0
            angulo_roll_total = 0
            angulo_pitch_total = 0
            altura_total = 0

            while not final :
                if randint (0,10)>2:
                    accion = np.argmax(q_table[estado])
                else:
                    accion = randint(0,2)
                nuevo_estado, recompensa, final, info= self.step(accion)
```

```
                q_table[estado][accion]=            q_table[estado][accion]
+tasa_aprendizaje*(recompensa              +              factor_descuento*
np.max(q_table[self.discretizar(nuevo_estado)]) - q_table[estado][accion])

                estado = self.discretizar(nuevo_estado)
                recompensa_total += recompensa

                #array_msg = Int32MultiArray()
                #altura_data = nuevo_estado
                #array_msg.data = altura_data
                #self.pub.publish(array_msg)
                time.sleep(0.1)


            angulo_roll_total += nuevo_estado[2]
            angulo_pitch_total += nuevo_estado[1]
            altura_total += nuevo_estado[0]

            listado_recompensa.append(recompensa_total)
            listado_recompensa_mean.append(np.mean(listado_recompensa))

            listado_angulo_roll.append(angulo_roll_total)
            listado_angulo_roll_mean.append(np.mean(listado_angulo_roll))

            listado_angulo_pitch.append(angulo_pitch_total)
            listado_angulo_pitch_mean.append(np.mean(listado_angulo_pitch))

            listado_altura.append(altura_total)
            listado_altura_mean.append(np.mean(listado_altura))
            time.sleep(1)

#self.pub.publish(np.mean(listado_estado))================================
==============

            if (episodio+1)%1 == 0:
                 print(f"episodio:      {episodio+1}      -      recompensa:
{np.mean(listado_recompensa)}          -          estado_          angulo_roll:
{np.mean(listado_angulo_roll)}-              estado_             angulo_pitch:
{np.mean(listado_angulo_pitch)} - estado_ altura: {np.mean(listado_altura)}
")


        "------------grafica-------------"

        fig, ax1 = plt.subplots(figsize=(22 , 6))

        color1 = 'tab:blue'
        ax1.set_xlabel('Episodio')
        ax1.set_ylabel('Recompensa promedio', color=color1)
        ax1.plot(range(1,    episodios    +    1),    listado_recompensa_mean,
label='Recompensa Promedio', color=color1)
        ax1.tick_params(axis='y', labelcolor=color1)
```

```
        # Crear eje secundario para el angulo
        ax2 = ax1.twinx()
        color2 = 'tab:red'
        ax2.set_ylabel('Angulo roll [deg]', color=color2)
        ax2.plot(range(1,   episodios   +   1),   listado_angulo_roll_mean,
label='roll promedio', color=color2, alpha=0.5)
        ax2.tick_params(axis='y', labelcolor=color2)

        # Mover el eje secundario para la altura un poco a la derecha
        ax3 = ax1.twinx()
        ax3.spines['right'].set_position(('outward',  50))    #  Ajustar  la
posicion
        color3 = 'tab:cyan'
        ax3.set_ylabel('Angulo pitch [deg]', color=color3)
        ax3.plot(range(1,   episodios   +   1),   listado_angulo_pitch_mean,
label='pitch promedio', color=color3, alpha=0.5)
        ax3.tick_params(axis='y', labelcolor=color3)

        # Mover el eje secundario para la altura un poco a la derecha
        ax4 = ax1.twinx()
        ax4.spines['right'].set_position(('outward',  100))    #  Ajustar  la
posicion
        color4 = 'tab:green'
        ax4.set_ylabel('Altura [cm]', color=color4)
        ax4.plot(range(1, episodios + 1), listado_altura_mean, label='Altura
promedio', color=color4, alpha=0.5)
        ax4.tick_params(axis='y', labelcolor=color4)

        fig.suptitle('Recompensa Promedio y Estado real por 50 episodios')
        fig.legend(loc='upper      right',      bbox_to_anchor=(1,      1),
bbox_transform=ax1.transAxes)


        # Guardar la imagen
        plt.savefig('Recompensa_y_Estado_altura-roll-
pitch_por_episodio_Qtabel_obstaculo.png')


        if final:

            rospy.loginfo("Episodio terminado reinicio del entorno...")
            self.reset()



"---------------------------NODO ROS-------------------------------------"


class ReceiverNode:
    def __init__(self):
        rospy.init_node('receiver_node', anonymous=True)
        rospy.Subscriber('sensor_topic', Float32MultiArray, self.callback)
```

```
        self.received_data = None

    def callback(self, data):

        self.received_data = data.data

    def get_received_data(self):
        return self.received_data



"----------------------FUNCION MAIN---------------------"

def main():
    try:

        "----------------- Funcion de la red neuronal / modelo y agente ---
---------------------- "

        def build_model_altura(states, actions):
            model = Sequential()
            model.add(Flatten(input_shape=(1,) + states))
            model.add(Dense(24, activation='relu'))
            model.add(Dense(24, activation='relu'))
            model.add(Dense(actions, activation='linear'))
            return model

        def build_model_altura_roll_pitch(states, actions):
            model = Sequential()
            model.add(Flatten(input_shape=(1,) + states))
            model.add(Dense(9, activation='relu'))
            model.add(Dense(81, activation ='relu'))
            model.add(Dense(6561, activation ='relu'))
            model.add(Dense(81, activation ='relu'))
            model.add(Dense(actions, activation='linear'))
            return model



#=============================================================================
============================================================

        def build_agent(model, actions):
            policy = BoltzmannQPolicy()
            memory = SequentialMemory(limit = 5000, window_length = 1)
            dqn = DQNAgent(model = model, memory = memory, policy = policy,
nb_actions = actions, nb_steps_warmup = 10, target_model_update = 1e-2)
            return dqn

        "----------------- Funcion para graficar fit y test --------------
----------- "

        def plot_fit (scores_fit):
```

```
            episode_rewards = scores_fit.history['episode_reward']

            # Calcular el promedio de la recompensa acumulada hasta cada
episodio
            average_rewards = [sum(episode_rewards[:i+1]) / (i+1) for i in
range(len(episode_rewards))]

            # Graficar el promedio de las recompensas acumuladas durante el
entrenamiento
            plt.plot(average_rewards)
            plt.title('Promedio de la recompensa acumulada durante el
entrenamiento')
            plt.xlabel('Episodio')
            plt.ylabel('Promedio de recompensa acumulada')


plt.savefig('Recompensa_durante_entramiento_altura_caminata.png')
            plt.close


#=================================================================
============================================================

        def plot_test (scores_test):

            episode_rewards = scores_test.history['episode_reward']

            # Calcular el promedio de la recompensa acumulada hasta cada
episodio
            average_rewards = [sum(episode_rewards[:i+1]) / (i+1) for i in
range(len(episode_rewards))]

            # Graficar el promedio de las recompensas acumuladas durante el
entrenamiento
            plt.plot(average_rewards)
            plt.title('Promedio de la recompensa acumulada durante el test')
            plt.xlabel('Episodio')
            plt.ylabel('Promedio de recompensa acumulada')

            plt.savefig('Recompensa_durante_test_altura_caminata.png')
            plt.close




        receiver_node = ReceiverNode()
        gym_env = Enviroment(receiver_node)


        #gym_env.run()
```

```
        "----------------- Funcion run de deep reforcement learning -------
------------------- "

        states = gym_env.observation_space.shape
        actions = gym_env.action_space.n
        model = build_model_altura(states, actions)

        #model = build_model_altura_roll_pitch(states, actions)
        model.summary()

        """
        with open('model_altura_caminata.txt', 'w') as f:
            with redirect_stdout(f):
                model.summary()

        # Leer el resumen del modelo desde el archivo
        with open('model_altura_caminta.txt', 'r') as f:
            model_altura_roll_summary_text = f.read()

        # Renderizar el resumen del modelo como una imagen
        plt.figure(figsize=(10, 5))
        plt.text(0.1,  0.5,  model_altura_roll_summary_text,  fontsize=10,
family='monospace')
        plt.axis('off')

        # Guardar la imagen
        plt.savefig('model_altura_caminita.png',       bbox_inches='tight',
pad_inches=0.1)
        """
        plot_model(model      =      model,show_shapes      =      True,
to_file='model_altura_caminta.png')
        plt.close()

        dqn = build_agent(model,actions)
        dqn.compile(Adam(lr= 1e-3), metrics = ['mse'])
        scores_fit  =  dqn.fit(  gym_env,nb_steps  =  100,  visualize  =  False,
verbose = 1)
        scores_test = dqn.test(gym_env, nb_episodes = 50, visualize = False)

        plot_fit(scores_fit)

        plot_test(scores_test)
        dqn.save_weights("altura_caminata.h5")

        rospy.spin()


    except rospy.ROSInterruptException:
        pass

if __name__ == '__main__':
    main()
```