JOSÉ VARELA-ALDÁS
LUIS F. RECALDE

SIMULACIÓN DE SISTEMAS ROBÓTICOS

Universidad Indoamérica

Simulación de Sistemas Robóticos

Simulación de Sistemas Robóticos

José Varela-Aldás
Luis F. Recalde

Universidad Indoamérica
2024

Fecha de publicación: 15 de julio de 2024

© Autores: José Varela-Aldás[1] y Luis F. Recalde[1]
[1] Centro de Investigación de Ciencias Humanas y de la Educación (CICHE), Universidad Indoamérica, Av. Bolívar y Quito, Ambato, Ecuador.
Correo: josevarela@uti.edu.ec, fernandorecalde@uti.edu.ec

ISBN: 978-9942-821-88-1
Derecho de Autor: QUI-065904

Revisado y aprobado para su publicación por el Comité Editorial de la Universidad Indoamérica (Quito, Ecuador) y por los revisores Dr. Carlos Bran (Universidad Don Bosco) y Dr. David Rivas (Universidad de las Fuerzas Armadas ESPE).

Editor: Ing. Hugo Arias Flores, MBA.

Editorial de la Universidad Indoamérica. Quito – Ecuador.

Para citar este libro: Varela-Aldás, J., Recalde, L. F. (2024). *Simulación de Sistemas Robóticos.* Quito, Ecuador: Editorial Universidad Tecnológica Indoamérica.

Índice de Contenidos

Índice de Figuras

1

Fundamentos Matemáticos para Robótica

Esta sección del libro presenta los conceptos y métodos necesarios utilizados en el desarrollo de un sistema robótico. Los temas abordados en esta sección incluyen vectores, matrices, posición y orientación, cinemática, cinemática diferencial e integral numérica. Además, se presentan casos de estudio que utilizan un robot móvil y un manipulador robótico para consolidar los conceptos presentados.

1.1 Vectores

Los vectores pueden definirse como elementos que poseen magnitud y dirección. Sin embargo, en el campo de la robótica, un vector representa la combinación de varias señales que pueden analizarse, incluyendo, por ejemplo, posición, velocidades, aceleraciones, torques y fuerzas.

Las propiedades asociadas a los vectores resultan de gran utilidad en el modelado y control de sistemas robóticos. En el conjunto del espacio vectorial , los vectores se denotan de la siguiente manera:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^T$$

donde las componentes $x_i \in \mathcal{R}\ \forall i \in \{1, 2, \ldots, n\}$. Así, es importante mencionar que el espacio vectorial se conoce como espacio euclidiano de dimensiones.

De esta manera, es posible definir las siguientes operaciones utilizando vectores.

Suma

La suma de vectores puede ser expresada de la siguiente manera:

$$\mathbf{x} + \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix}$$

donde la suma solo es posible con vectores de igual dimensión. $\mathbf{x}, \mathbf{y} \in \mathcal{R}^n$.

Además, se cumple que la suma entre vectores satisface las siguientes propiedades.

- **Propiedad Conmutativa y Asociativa:**

$$\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$$

$$(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$$

- **El Vector Nulo:** Como su nombre lo indica, todas las componentes de este vector son ceros, es decir, $\mathbf{o} = [0 \quad 0 \ldots \quad 0]^T \in \mathcal{R}^n$, definiendo las siguientes operaciones:

$$\mathbf{x} + \mathbf{0} = \mathbf{0} + \mathbf{x} = \mathbf{x}$$

- **Aditivo Inverso:** Para cada vector $\mathbf{x} \in \mathcal{R}^n$ se puede definir un inverso aditivo definido —x, así:

$$\mathbf{x} - \mathbf{x} = \mathbf{0}$$

Producto por un Escalar

La definición del producto de un vector $\mathbf{x} \in \mathcal{R}^n$ por un escalar $\alpha \in \mathcal{R}$ se expresa de la siguiente manera:

$$\alpha\mathbf{x} = \begin{bmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{bmatrix} = \alpha \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

al considerar los vectores $\mathbf{x}, \mathbf{y} \in \mathcal{R}^n$, y los escalares $\alpha, \omega \in \mathcal{R}$, el producto de un vector por un escalar satisface las siguientes propiedades.

- **Distributiva:**

$$\alpha(\mathbf{x} + \mathbf{y}) = \alpha\mathbf{x} + \alpha\mathbf{y}$$

$$(\alpha + \omega)\mathbf{x} = \alpha\mathbf{x} + \omega\mathbf{x}$$

- **Neutro Unidad:** Se considera que, para cada vector $\mathbf{x} \in \mathcal{R}^n$, existe un elemento neutro unidad:

$$1\mathbf{x} = \mathbf{x}$$

- **Asociativa y Conmutativa:** Se consideran el vector $\mathbf{x} \in \mathcal{R}^n$ y los escalares $\alpha, \omega \in \mathcal{R}$ y se cumple lo siguiente:

$$\alpha(\omega\mathbf{x}) = (\alpha\omega)\mathbf{x} = \omega(\alpha\mathbf{x})$$

Producto Interno de Vectores

Asimismo, se puede definir el producto vectorial interno de la siguiente manera:

$$\mathbf{x}^T\mathbf{y} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i \in \mathcal{R}$$

donde el producto interno de vectores presenta estas propiedades.

- **Asociativa:** Se puede definir la propiedad asociativa respecto al producto interno de vectores, así:

$$\mathbf{x}^T(\mathbf{y} + \mathbf{z}) = \mathbf{x}^T\mathbf{y} + \mathbf{x}^T\mathbf{z} = \mathbf{y}^T\mathbf{x} + \mathbf{z}^T\mathbf{x}$$

Norma de Vectores

La norma de un vector $\mathbf{x} \in \mathcal{R}^n$ es un escalar positivo, el cual representa la magnitud que tiene un vector y se define como:

$$||\mathbf{x}|| = \sqrt{\mathbf{x}^T\mathbf{x}} = \sqrt{\sum_{i=1}^{n}(x_i^2)} \in \mathcal{R}^+$$

de manera similar a las operaciones mencionadas anteriormente, la norma presenta las siguientes propiedades:

- $||\mathbf{x}|| > 0,\ \mathbf{x} \in \mathcal{R}^n$, si $\mathbf{x} \neq \mathbf{0}$
- $||\mathbf{x}|| = 0,\ \mathbf{x} \in \mathcal{R}^n$, si y sólo si $\mathbf{x} = \mathbf{0}$
- Se cumple la desigualdad triangular

  $||\mathbf{x}|| - ||\mathbf{y}|| \leq ||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||,\ \forall \mathbf{x}, \mathbf{y} \in$
- $||\alpha \mathbf{x}|| = |\alpha| ||\mathbf{x}|| \forall \alpha \in \mathcal{R}, \mathbf{x} \in \mathcal{R}^n$
- Desigualmente de Schwarz: $|\mathbf{x}^T \mathbf{y}| \leq ||\mathbf{x}|| ||\mathbf{y}||\ \forall \mathbf{x}, \mathbf{y} \in \mathcal{R}^n$

1.2 Matrices

Se define a una matriz $\mathbf{A} = \{a_{ij}\} \in \mathcal{R}^{n \times m}$, donde $\{a_{ij}\} \in \mathcal{R}$ es un arreglo de números con n filas y m columnas que definen la dimensión de la matriz; además, sus elementos son:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}$$

Matrices Especiales

Existen matrices que ayudan al desarrollo de algoritmos de control en robótica, por ejemplo, la matriz identidad $\mathbf{I} = \{a_{ij}\} \in \mathcal{R}^{n \times n}$, donde $a_{ij} = 1$ si $i = j$ si $i = j$ y $a_{ij} = 0$ si $i \neq j$, definida por:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1_{nn} \end{bmatrix} \in \mathcal{R}^{n \times n}$$

adicionalmente, se tiene una matriz diagonal $\mathbf{D} = \{a_{ij}\} \in \mathcal{R}^{n \times n}$, donde $a_{ij} \in \mathcal{R}$ si $i = j$, y $a_{ij} = 0$ para $i \neq j$, definida así:

$$\mathbf{D} = diag\{a_{11}, a_{22}, \dots a_{nn}\} = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix} \in \mathcal{R}^{n \times n}$$

una matriz simétrica se puede definir por $\mathbf{A} = \{a_{ij}\} \in \mathcal{R}^{n \times n}$ y es igual a su transpuesta; esto es $\mathbf{A} = \mathbf{A}^T$. Mientras que una matriz antisimétrica se representa como $\mathbf{A} = -\mathbf{A}^T$.

Una matriz puede ser definida como positiva $\mathbf{A} \in \mathcal{R}^{n \times n}$; de esta manera, sin necesidad de ser simétrica, cumple con la siguiente expresión:

$$\mathbf{x}^T \mathbf{A} \mathbf{x} > 0 \ \forall \mathbf{x} \neq \mathbf{0} \in \mathcal{R}^n$$

además, una matriz puede ser semidefinida positiva, aunque $\mathbf{A} \in \mathcal{R}^{n \times n}$ no necesariamente sea simétrica, de la siguiente manera:

$$\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0 \ \forall \mathbf{x} \in \mathcal{R}^n$$

con todo lo explicado anteriormente, es posible definir operaciones y propiedades de las matrices que son comúnmente utilizadas en el campo de la robótica.

Suma de Matrices

La suma en matrices solo es posible entre matrices de igual dimensión. Si se considera $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times m}$, la suma se puede definir así:

$$\mathbf{A}+\mathbf{B} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1m}+b_{1m} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2m}+b_{2m} \\ \vdots & & \ddots & \vdots \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \dots & a_{nm}+b_{nm} \end{bmatrix}$$

la suma de matrices cumple las siguientes propiedades, teniendo en cuenta estos elementos $\mathrm{A}, \mathrm{B}, \mathrm{C} \in \mathcal{R}^{n \times m}$, tal que:

- **Propiedad Conmutativa:**

$$\mathrm{A}+\mathrm{B}+\mathrm{C} = \mathrm{B}+\mathrm{A}+\mathrm{C} = \mathrm{A}+\mathrm{C}+\mathrm{B} = \mathrm{C}+\mathrm{A}+\mathrm{B} = \mathrm{C}+\mathrm{B}+\mathrm{A}$$

- **Propiedad Asociativa:**

$$\mathbf{A}+\mathbf{B}+\mathbf{C} = \mathbf{A}+(\mathbf{B}+\mathbf{C}) = (\mathbf{A}+\mathbf{B})+\mathbf{C}$$

- **Elemento Neutro Aditivo:** Se define por una matriz nula $\mathbf{0} = \{0_{ij}\} \in \mathcal{R}^{n \times m}$, donde $0_{ij} = 0$, y se tiene la siguiente propiedad:

$$\mathbf{0}+\mathbf{A} = \mathbf{A}+\mathbf{0} = \mathbf{A}$$

Transpuesta de una Matriz

Para definir la operación traspuesta, se considera la matriz $\mathbf{A} = \{a_{ij}\} \in \mathcal{R}^{n \times m}$, donde la operación transpuesta sobre esta matriz se expresa como $\mathbf{A}^T = \{a_{ji}\} \in \mathcal{R}^{m \times n}$, de la siguiente manera:

$$\mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \vdots & & \ddots & \vdots \\ a_{1m} & a_{2m} & \dots & a_{mn} \end{bmatrix} \in \mathcal{R}^{m \times n}$$

además, es importante mencionar que la operación transpuesta cumple con ciertas propiedades que son útiles al diseñar algoritmos de control. Estas propiedades se definen de la siguiente forma:

- $(\mathbf{A}^T)^T = \mathbf{A}$
- $((\mathbf{A}^T)^T)^T = \mathbf{A}^T$
- $(\alpha\mathbf{A})^T = \alpha\mathbf{A}^T = \mathbf{A}^T\alpha$, donde $\alpha \in \mathcal{R}$
- $(\mathbf{I})^T = \mathbf{I}$, donde $\mathbf{I} \in \mathcal{R}^{n \times n}$ es una matriz identidad.

- Si se tiene la siguiente matriz diagonal $\mathbf{D} \in \mathcal{R}^{n \times n}$, se tiene que $\mathbf{D}^T = \mathbf{D}$
- $(\mathbf{A} + \mathbf{B} + \mathbf{C})^T = \mathbf{A}^T + \mathbf{B}^T + \mathbf{C}^T$
- $(\mathbf{A}^T + \mathbf{B}^T + \mathbf{C}^T)^T = \mathbf{A} + \mathbf{B} + \mathbf{C}$

Resta de Matrices

Para definir la resta, se consideran las siguientes matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times m}$, donde la sustracción se define como $\mathbf{C} = \mathbf{A} - \mathbf{B}$,

donde $\mathbf{C} \in \mathcal{R}^{n \times m}$

Para definir la resta, se toman en cuenta las siguientes matrices $\mathbf{A} \in \mathcal{R}^{n \times m}$ y $\mathbf{B} \in \mathcal{R}^{n \times m}$ La sustracción se define como $\mathbf{C} = \mathbf{A} - \mathbf{B} \in \mathcal{R}^{n \times m}$ de la siguiente manera:

$$\mathbf{A}-\mathbf{B}=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}+\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix}=\begin{bmatrix} a_{11}-b_{11} & a_{12}-b_{12} & \dots & a_{1m}-b_{1m} \\ a_{21}-b_{21} & a_{22}-b_{22} & \dots & a_{2m}-b_{2m} \\ \vdots & & \ddots & \vdots \\ a_{n1}-b_{n1} & a_{n2}-b_{n2} & \dots & a_{nm}-b_{nm} \end{bmatrix}$$

además, la resta de matrices no es conmutativa; por lo tanto, $\mathbf{A}-\mathbf{B}\neq\mathbf{B}-\mathbf{A}$.

Finalmente, la operación de resta de matrices cumple con las siguientes propiedades:

- $\mathbf{A}-\mathbf{B}-\mathbf{C}=-\mathbf{B}+\mathbf{A}-\mathbf{C}=\mathbf{A}-\mathbf{C}-\mathbf{B}=-\mathbf{C}+\mathbf{A}-\mathbf{B}$
- $\mathbf{A}-\mathbf{B}-\mathbf{C}=\mathbf{A}-(\mathbf{B}+\mathbf{C})=(\mathbf{A}-\mathbf{B})-\mathbf{C}=\mathbf{A}-(\mathbf{C}+\mathbf{B})$
- $(\mathbf{A}-\mathbf{B}-\mathbf{C})^T=\mathbf{A}^T-\mathbf{B}^T-\mathbf{C}^T$
- $(\mathbf{A}^T-\mathbf{B}^T-\mathbf{C}^T)^T=\mathbf{A}-\mathbf{B}-\mathbf{C}$

Multiplicación de Matrices

Para definir la multiplicación, se consideran las siguientes matrices definidas $\mathbf{A}\in\mathcal{R}^{n\times m}$ y $\mathbf{B}\in\mathcal{R}^{m\times n}$, donde el número de columnas de A es igual al número de filas de **B**. Por lo tanto, el producto se define como $\mathbf{C}=\mathbf{AB}\in\mathcal{R}^{n\times n}$, así:

$$\mathbf{AB}=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}=\begin{bmatrix} \sum_{i=1}^{m} a_{1i}b_{i1} & \sum_{i=1}^{m} a_{1i}b_{i2} & \dots & \sum_{i=1}^{m} a_{1i}b_{in} \\ \sum_{i=1}^{m} a_{2i}b_{i1} & \sum_{i=1}^{m} a_{2i}b_{i2} & \dots & \sum_{i=1}^{m} a_{2i}b_{in} \\ \vdots & & \ddots & \vdots \\ \sum_{i=1}^{m} a_{ni}b_{i1} & \sum_{i=1}^{m} a_{ni}b_{i2} & \dots & \sum_{i=1}^{m} a_{ni}b_{in} \end{bmatrix}$$

bajo estas consideraciones, se determina que el producto de matrices no es conmutativo definido por $\mathbf{AB}\neq\mathbf{BA}$.

Además, considerando las siguientes matrices $\mathbf{A}\in\mathcal{R}^{n\times}$, $\mathbf{B}\in\mathcal{R}^{p\times m}$ $\mathbf{D}\in\mathcal{R}^{p\times m}$, y el escalar $\alpha\in\mathcal{R}$, se tienen estas propiedades asociadas al producto de matrices.

- Ley asociativa $\mathbf{C} = \mathbf{A}(\mathbf{BE}) = (\mathbf{AB})\mathbf{E},\ \mathbf{C} \in \mathcal{R}^{n \times n}$
- Ley distributiva $\mathbf{C} = \mathbf{A}(\mathbf{B} + \mathbf{D}) = (\mathbf{AB} + \mathbf{AD}) \in \mathcal{R}^{n \times m}$

- $\mathbf{C} = (\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T \in \mathcal{R}^{m \times n}$
- $\mathbf{C} = (\mathbf{ABE})^T = \mathbf{E}^T\mathbf{B}^T\mathbf{A}^T \in \mathcal{R}^{n \times n}$
- Se tiene $\mathbf{0} \in \mathcal{R}^{p \times m}$, por lo tanto $\mathbf{A0} = \mathbf{0} \in \mathcal{R}^{n \times m}$
- $\mathbf{C} = \alpha\mathbf{A} = \mathbf{A}\alpha \in \mathcal{R}^{n \times p}$
- Se tiene que $\alpha = 0$, por lo tanto $\alpha\mathbf{A} = \mathbf{0} \in \mathcal{R}^{n \times p}$

Dado la matriz $\mathbf{F} \in \mathcal{R}^{n \times n}$ la matriz identidad definida por $\mathbf{I} \in \mathcal{R}^{n \times n}$, se tiene que $\mathbf{FI} = \mathbf{IF} = \mathbf{F}$

Determinante de una Matriz

El determinante puede ser definido considerando una matriz cuadrada $\mathbf{A} = \{a_{ij}\} \in \mathcal{R}^{n \times m}$, donde $|\mathbf{A}|$ se conoce como el determinante y se lo expresa de forma general como.

$$|\mathbf{A}| = \sum_{j=1}^{n}((-1)^{i+j}a_{ij}|\mathbf{M}_{ij}|)$$

donde a_{ij} son elementos de la matriz $\mathbf{A}$ y $\mathbf{M}$ es una submatriz que se forma al eliminar los elementos de la fila i y la columna j.

Adjunta de una Matriz

Para el cálculo de la adjunta, se considera una matriz cuadrada definida por $\mathbf{A} = \{a_{ij}\} \in \mathcal{R}^{n \times m}$, donde su adjunta $adj(\mathbf{A})$ viene dada por los elementos $\{c_{ij}\} = (-1)^{i+j}|\mathbf{M}_{ij}|$ de la matriz de cofactores $\mathbf{C}$, considerando $\mathbf{M}_{ij}$ como una submatriz que se forma al eliminar los elementos de la fila i y la columna j.

Finalmente, la matriz adjunta se puede obtener generando la transpuesta de la matriz de cofactores C; esta operación se define de la siguiente forma:

$$adj(\mathbf{A}) = \mathbf{C}^T$$

Inversa de una Matriz

Para definir la operación de inversa, se considera una matriz cuadrada $\mathbf{A} = \{a_{ij}\} \in \mathcal{R}^{n \times m}$, donde su inversa se expresa como $\mathbf{A}^{-1}$. De llegar a existir inversa, se considera la siguiente expresión:

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

además, para definir la inversa de una matriz de debe considerar que el $|\mathbf{A}| \neq 0$ y la inversa se define así:

$$\mathbf{A}^{-1} = \frac{adj(\mathbf{A})}{|\mathbf{A}|}$$

con las formulaciones presentadas anteriormente, las siguientes propiedades se pueden asociar a la inversa de matrices considerando $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$:

- $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$
- $(\mathbf{AB})^{-1}\mathbf{AB} = \mathbf{B}^{-1}\mathbf{A}^{-1}\mathbf{AB} = \mathbf{B}^{-1}(\mathbf{A}^{-1}\mathbf{A})\mathbf{B} = \mathbf{B}^{-1}\mathbf{IB} = \mathbf{I}$

Para una explicación aún más detallada de vectores y matrices, se recomienda al lector los trabajos de (Strang, n.d.) titulado *Vector Spaces* y (Lang, 1987).

1.3 Pose en 2 Dimensiones

Orientaciones en 2 Dimensiones

Para definir la orientación, se establece un punto arbitrario $\mathbf{p} = [x \quad y]^T \in \mathcal{R}^2$, el cual puede ser referido respecto a diferentes sistemas de referencia $\{\mathcal{V}\}$ o $\{\mathcal{B}\}$. Por lo tanto, este vector puede expresarse con respecto al marco $\{\mathcal{V}\}$ formado por ${}^V\mathbf{p} = [{}^Vx \quad {}^Vy]^T$, utilizando su posición relativa con respecto a $\{\mathcal{B}\}$, definido por ${}^B\mathbf{p} = [{}^Bx \quad {}^By]^T$, de la siguiente manera:

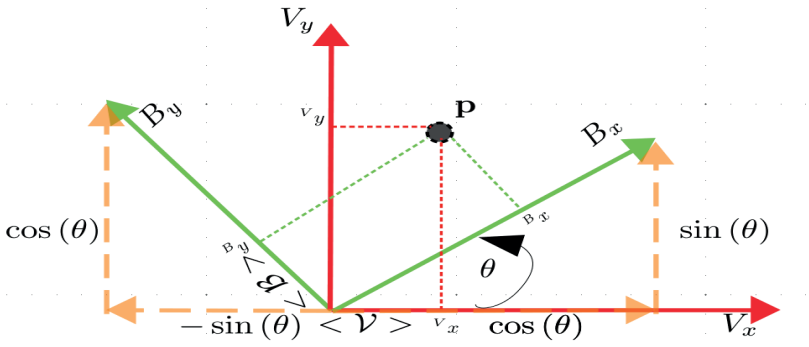
$$\begin{bmatrix} {}^Vx \\ {}^Vy \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} {}^Bx \\ {}^By \end{bmatrix}$$

$${}^V\mathbf{p} = {}^V\mathbf{R}_B \, {}^B\mathbf{p}$$

donde ${}^V\mathbf{R}_B$ es una matriz de rotación girada en sentido antihorario un ángulo θ, que transforma el vector $\mathbf{p} = [x \quad y]^T$ del sistema de referencia hacia una mejor representación se muestra en la **Figura 1.**

Figura 1

Concepto de rotación aplicado a sistemas de referencia, fundamental en geometría, física y ciencias de la ingeniería



Adicionalmente, este vector respecto al *frame* $\{\mathcal{B}\}$ puede ser expresado de la siguiente forma:

$$\begin{bmatrix} {}^{B}x \\ {}^{B}y \end{bmatrix} = ({}^{V}\mathbf{R}_{B})^{-1} \begin{bmatrix} {}^{V}x \\ {}^{V}y \end{bmatrix} = ({}^{V}\mathbf{R}_{B})^{T} = ({}^{B}\mathbf{R}_{V}) \begin{bmatrix} {}^{V}x \\ {}^{V}y \end{bmatrix}$$

Orientación y Desplazamiento en 2 Dimensiones

Para representar la pose (posición y orientación) de un punto, es necesario considerar la traslación y rotación que se presenta entre los sistemas de referencia desde $\{\mathcal{B}\}$ hacia $\{\mathcal{V}\}$, lo cual puede ser escrito de la siguiente forma:

$$\begin{bmatrix} {}^{V}x \\ {}^{V}y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} {}^{B}x \\ {}^{B}y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} {}^{V}x \\ {}^{V}y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \end{bmatrix} \begin{bmatrix} {}^{B}x \\ {}^{B}y \\ 1 \end{bmatrix}$$

Esta expresión puede ser escrita de forma compacta así:

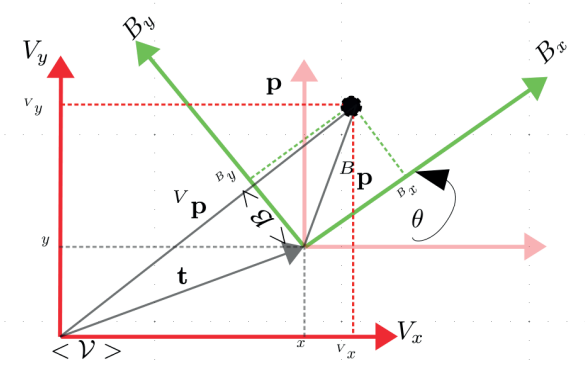
$$ {}^{V}\bar{\mathbf{p}} = \begin{bmatrix} {}^{V}\mathbf{R}_B & \mathbf{t} \\ \mathbf{0}_{1\times 1} & 1 \end{bmatrix} {}^{B}\bar{\mathbf{p}} $$

$$ {}^{V}\bar{\mathbf{p}} = {}^{V}\mathbf{H}_B \, {}^{B}\bar{\mathbf{p}} $$

donde $\mathbf{t} = [x \quad y]^T$ y ${}^{V}\mathbf{R}_B$ es el vector de traslación y rotación del *frame* $\{\mathcal{B}\}$ medido desde $\{\mathcal{V}\}$. Adicionalmente, ${}^{V}\bar{\mathbf{p}} = [{}^{V}x \quad {}^{V}y \quad 1]^T$ es el vector homogéneo de ${}^{V}\mathbf{p}$, ${}^{B}\bar{\mathbf{p}} = [{}^{B}x \quad {}^{B}y \quad 1]^T$ es el vector homogéneo de ${}^{B}\mathbf{p}$. Finalmente, ${}^{V}\mathbf{H}_B$ es una matriz de transformación homogénea, una mejor representación se muestra en la **Figura 2.**

Figura 2

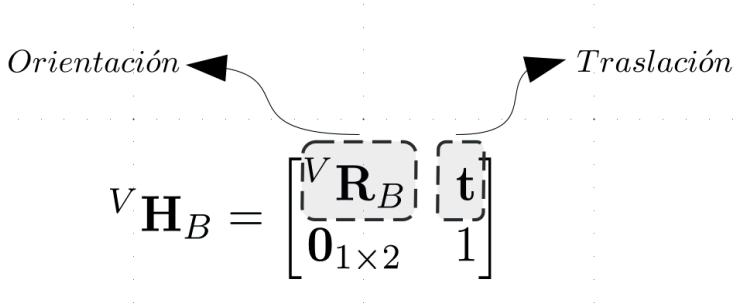
Combinación de rotación y traslación aplicada a sistemas de referencia



La matriz de transformación homogénea (**ver Figura 3**) representa la posición y orientación del sistema de referencia $\{\mathcal{B}\}$ respecto al sistema $\{\mathcal{V}\}$.

Figura 3

Representación de una matriz de transformación homogénea en dos dimensiones. Esta matriz es fundamental en gráficos computacionales y geometría



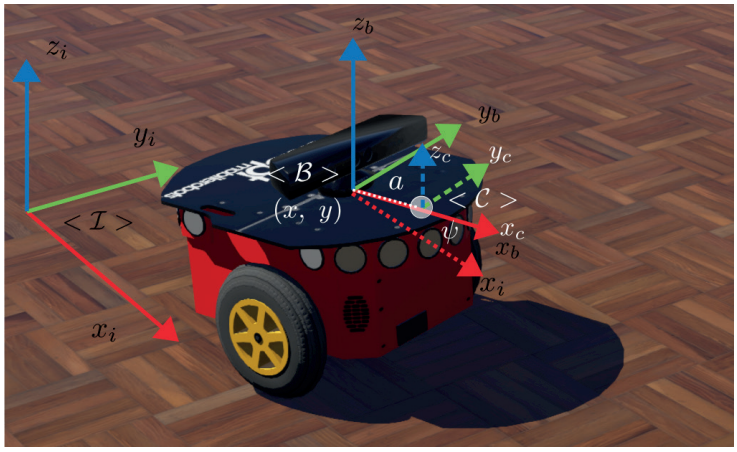
Información adicional sobre matrices de transformación y rotación en 2 dimensiones se presenta en el siguiente trabajo (Corke, 2017).

Caso Estudio Robot Móvil

Con las formulaciones presentadas, es posible describir la orientación y traslación de un robot móvil tipo diferencial. Para ello, se establecen una serie de sistemas de referencia que servirán para generar esta formulación, como se muestra en la **Figura 4.**

Figura 4

Representación de un robot móvil con los respectivos sistemas de referencia



Con base en lo descrito, la ubicación y orientación del sistema de referencia $\{\mathcal{C}\}$se pueden definir en función de matrices de transformación homogénea, como se muestra a continuación:

$$ {}^{I}\mathbf{H}_{B} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & x \\ \sin(\psi) & \cos(\psi) & y \\ 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{B}\mathbf{H}_{C} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{I}\mathbf{H}_{C} = {}^{I}\mathbf{H}_{B}\ {}^{B}\mathbf{H}_{C} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & a\cos(\theta) + x \\ \sin(\theta) & \cos(\theta) & a\sin(\theta) + y \\ 0 & 0 & 1 \end{bmatrix} $$

una vez encontrada esta representación, se puede conocer la posición y orientación; esto es útil para aplicaciones de control. Además, si se cuentan con las medidas de algún sensor, las cuales están referenciadas al sistema de coordenadas $\{\mathcal{C}\}$, estas pueden ser proyectadas al sistema $\{\mathcal{I}\}$ utilizando la matriz de transformación homogénea ${}^{I}\mathbf{H}_{C}$.

Para un mejor entendimiento de cómo generar y operar con matrices de transformación homogénea, se han implementado dichas operaciones en el lenguaje de programación Python.

Para obtener más información, se puede ingresar en el enlace https://bit.ly/3zCYNJk o ir directamente a la sección anexos (Anexo 1).

Además, se recomienda al lector revisar el trabajo de (Grafarend & Kühnel, 2011) titulado *A minimal atlas for the rotation group.*

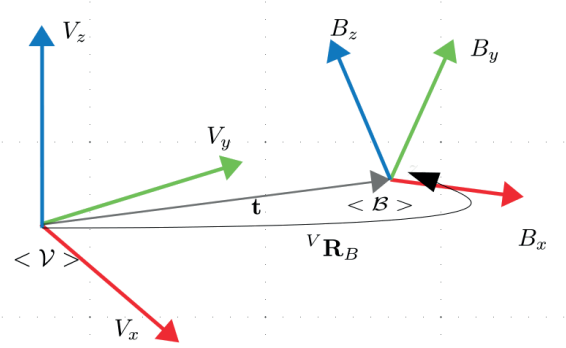
1.4 Pose en 3 Dimensiones

El uso de tres dimensiones es una extensión de lo descrito anteriormente, donde solo se consideraron dos dimensiones. En este caso, se agrega un eje adicional, generalmente denotado por Z, el cual es ortogonal respecto a los ejes X y Y. Para la correcta ubicación de los ejes, se debe considerar la regla de la mano derecha.

Para un mejor entendimiento, se presenta la **Figura 5**, que muestra dos sistemas de referencia, $\{\mathcal{V}\}$ y $\{\mathcal{B}\}$.

Figura 5

Representación de sistemas de referencia que han sido trasladados y rotados en un entorno tridimensional



donde $\mathbf{t} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ y $^{V}\mathbf{R}_B$ es el desplazamiento y la rotación arbitraria del sistema $\{\mathcal{B}\}$ respecto a $\{\mathcal{V}\}$.

Orientaciones en 3 Dimensiones

En el transcurso de la historia, los matemáticos han desarrollado diversas formas para representar rotaciones, como matrices ortonormales de rotación, ángulos de Euler, ángulos de rotación sobre ejes y cuaterniones unitarios. Todas estas formulaciones pueden expresarse como vectores o matrices, de los cuales se presenta introducción.

Las matrices ortonormales de rotación permiten especificar rotaciones θ respecto a diferentes ejes. A lo largo de los ejes x, y y z, se presentan las matrices definidas de la siguiente manera:

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

es importante mencionar que las rotaciones en tres dimensiones no son conmutables; por lo tanto, el orden en que se realicen estas operaciones produce un resultado diferente.

Cuando se busca describir la orientación total de un cuerpo rígido, la secuencia de rotaciones utilizada en este trabajo es la siguiente:

$$\mathbf{R} = \mathbf{R}_x(\phi)\mathbf{R}_y(\theta)\mathbf{R}_z(\psi)$$

donde ϕ, θ y ψ se conocen como los ángulos de roll, pitch y yaw, respectivamente.

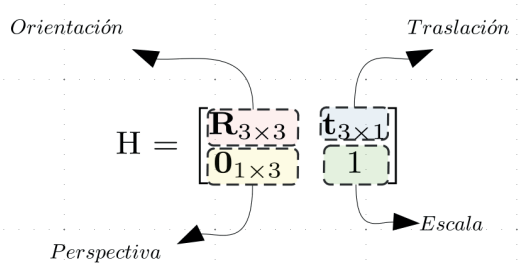
Cabe mencionar que en la literatura existen un sin número de secuencias de rotación, tales como *XYZ*, *ZYX*, *YXZ*, entre muchas más. Para comprender mejor las matrices de rotación, se ha implementado su formulación en el lenguaje de programación Python. Para obtener más información, se puede ingresar en el enlace https://bit.ly/4bxKBP6 o ir directamente a la sección anexos (Anexo 2).

Desplazamiento y Orientación en 3 Dimensiones

Para representar la posición y orientación de un sistema de referencia en el espacio 3D, es necesario utilizar matrices homogéneas $\mathbf{H} \in \mathcal{R}^{4\times4}$, una formulación análoga a la presentada anteriormente para el caso de 2D. Se asume que no hay transformación de perspectiva y que el escalado es unitario, lo que resulta en una matriz homogénea de la siguiente forma:

Figura 6

Matriz transformación homogénea 3D



La Figura 6 representa la traslación y orientación de un sistema de referencia arbitrario $\{\mathcal{B}\}$ respecto a otro sistema de referencia $\{\mathcal{V}\}$. Además, esta matriz homogénea puede ser utilizada para expresar las coordenadas de un punto $\mathbf{p} = [x \quad y \quad z]^T \in \mathcal{R}^3$ respecto al sistema $\{\mathcal{V}\}$, tal que ${}^V\mathbf{p} = [{}^Vx \quad {}^Vy \quad {}^Vz]^T$ o respecto al sistema $\{\mathcal{B}\}$ representado por ${}^B\mathbf{p} = [{}^Bx \quad {}^By \quad {}^Bz]^T$. De esta manera, se define la siguiente transformación:

$$ {}^V\bar{\mathbf{p}} = {}^V\mathbf{H}_B \, {}^B\bar{\mathbf{p}} $$

donde ${}^V\bar{\mathbf{p}}$ y ${}^B\bar{\mathbf{p}}$ son vectores homogéneos.

Bajo estas consideraciones, se puede analizar detalladamente el uso de matrices homogéneas como herramientas para representar la posición de objetos en el espacio 3D y realizar proyecciones.

Para el caso en el cual solo se presente la traslación de un sistema de referencia $\{\mathcal{B}\}$ respecto a otro $\{\mathcal{V}\}$, la matriz de transformación homogénea puede definirse de la siguiente manera:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{t}_{3\times 1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}$$

donde $\mathbf{t} = [t_x \quad t_y \quad t_z] \in \mathcal{R}^3$, es el vector de traslación respecto al sistema $\{\mathcal{V}\}$.

En el caso en el que el sistema $\{\mathcal{B}\}$se encuentra solo rotado respecto a $\{\mathcal{V}\}$, la matriz ${}^{V}\mathbf{R}_{B}$ será la que defina esta transformación. En consecuencia, se pueden definir tres matrices homogéneas básicas de rotación:

$$\mathbf{H}(x,\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}(y,\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}(z,\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 & 0 \\ \sin(\psi) & \cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Cuando se utilizan matrices de transformación homogénea, es importante tener en cuenta si primero se realiza la rotación y después la traslación, o viceversa. Esto se debe a que son transformaciones espaciales no conmutativas, por lo que se obtendrán resultados distintos dependiendo del orden de las operaciones.

Rotación y Traslación

Cuando se realiza una rotación sobre uno de los ejes respectivos y luego una traslación, se generan las matrices homogéneas que se muestran a continuación:

$$\mathbf{H}((x,\phi),\mathbf{t}) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & \cos(\phi) & -\sin(\phi) & t_y \\ 0 & \sin(\phi) & \cos(\phi) & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}((y,\theta),\mathbf{t}) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & t_x \\ 0 & 1 & 0 & t_y \\ -\sin(\theta) & 0 & \cos(\theta) & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}((z,\psi),\mathbf{t}) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 & t_x \\ \sin(\psi) & \cos(\psi) & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Traslación y Rotación

Cuando se realiza primero una traslación seguida de una rotación sobre los ejes respectivos, se generan las siguientes matrices de transformación homogénea:

$$\mathbf{H}(\mathbf{t},(x,\phi),) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & \cos(\phi) & -\sin(\phi) & t_y\cos(\phi) - t_z\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) & t_y\sin(\phi) + t_z\cos(\phi) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}(\mathbf{t},(y,\theta)) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & t_x\cos(\theta) + t_z\sin(\theta) \\ 0 & 1 & 0 & t_y \\ -\sin(\theta) & 0 & \cos(\theta) & t_z\cos(\theta) - t_x\sin(\theta) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}(\mathbf{t},(z,\psi)) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 & t_x\cos(\psi) - t_y\sin(\psi) \\ \sin(\psi) & \cos(\psi) & 0 & t_x\sin(\theta) + t_y\cos(\psi) \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Composición de Matrices Homogéneas

Debido a la capacidad de representar orientación y traslación respecto a diferentes sistemas de referencia, las matrices homogéneas pueden utilizarse para representar múltiples traslaciones y rotaciones.

Por ejemplo, si se tiene una rotación ϕ sobre el eje *x,* seguida de una rotación θ sobre el eje y; y, finalmente un desplazamiento angular de ψ sobre el eje *z*, esto puede expresarse mediante la siguiente composición de matrices homogéneas:

$$\mathbf{H} = \mathbf{H}(z,\psi)\mathbf{H}(y,\theta)\mathbf{H}(x,\phi) = \begin{bmatrix} \cos(\psi)\cos(\theta) & \sin(\phi)\sin(\theta)\cos(\psi) - \sin(\psi)\cos(\phi) & \sin(\phi)\sin(\psi) + \sin(\theta)\cos(\phi)\cos(\psi) & 0 \\ \sin(\psi)\cos(\theta) & \sin(\phi)\sin(\psi)\sin(\theta) + \cos(\phi)\cos(\psi) & -\sin(\phi)\cos(\psi) + \sin(\psi)\sin(\theta)\cos(\phi) & 0 \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

sin embargo, dado que el producto de matrices no es conmutativo, la composición de matrices homogéneas tampoco posee esta característica. Si se invierte el orden de las operaciones previamente mencionadas, se obtiene el siguiente resultado:

$$\mathbf{H} = \mathbf{H}(z,\psi)\mathbf{H}(y,\theta)\mathbf{H}(x,\phi) = \begin{bmatrix} \cos(\psi)\cos(\theta) & \sin(\phi)\sin(\theta)\cos(\psi) - \sin(\psi)\cos(\phi) & \sin(\phi)\sin(\psi) + \sin(\theta)\cos(\phi)\cos(\psi) & 0 \\ \sin(\psi)\cos(\theta) & \sin(\phi)\sin(\psi)\sin(\theta) + \cos(\phi)\cos(\psi) & -\sin(\phi)\cos(\psi) + \sin(\psi)\sin(\theta)\cos(\phi) & 0 \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Para comprender mejor las matrices homogéneas, se ha implementado su formulación y las operaciones asociadas en el lenguaje de programación Python. Si se desea obtener más información sobre este tema, se puede ingresar en el enlace https://bit.ly/3Wcgx6j o revisar la sección anexos (Anexo 3).

Si se desea obtener más información acerca de matrices de transformación para rotaciones y traslaciones, se puede revisar el trabajo de (Corke, 2017) titulado *Vision and Control.*

1.5 Cinemática Directa de Robots

La cinemática de un robot permite estudiar el movimiento de un punto o sistema de interés respecto a un sistema de referencia y, en particular, la cinemática directa consiste en describir la posición y orientación del extremo operativo o punto de interés en función de la configuración de los estados internos del robot (Siciliano, 2016).

Debido a que un robot puede considerarse como una cadena cinemática formada por eslabones unidos entre sí mediante articulaciones, es posible situar un sistema de referencia en la base del robot y describir la localización de cada uno de los eslabones respecto a la base. Por lo tanto, el problema de la cinemática directa se reduce a encontrar las matrices de transformación homogéneas en función de las articulaciones que relacionan la posición del extremo operativo respecto a un sistema fijo.

Para ejemplificar el uso de matrices homogéneas de transformación, se proponen dos ejemplos: un manipulador

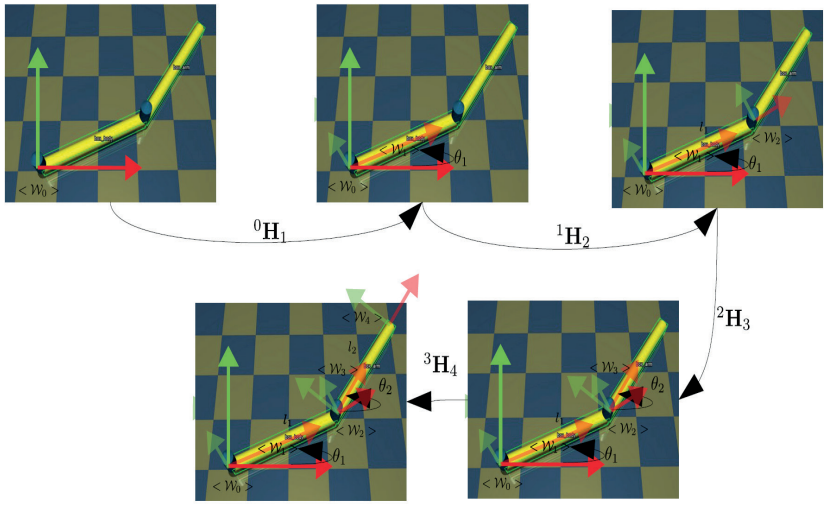
robótico planar, capaz de moverse solo en un plano, y un manipulador robótico de 3 grados de libertad, capaz de realizar movimientos en el espacio 3D.

Manipulador Robótico Planar

El primer caso consiste en un robot de dos grados de libertad que se mueve en el plano $(x - y)$. En la **Figura 7** se muestra el uso de las matrices de transformación homogéneas para representar la posición y orientación del punto final del manipulador robótico. Esta transformación es esencial en robótica para describir y controlar la posición y orientación del robot.

Figura 7

Proceso de transformación de los sistemas de referencia en un robot planar en dos dimensiones



Las matrices de transformación homogéneas, en función de las articulaciones y dimensiones del robot, se presentan a continuación:

$$ {}^{0}\mathbf{H}_{1} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}\mathbf{H}_{2} = \begin{bmatrix} 1 & 0 & 0 & l_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}\mathbf{H}_{3} = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}\mathbf{H}_{4} = \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

una vez definidas estas matrices, es posible describir la transformación de $\mathcal{W}_0$ a $\mathcal{W}_4$, de la siguiente manera:

$$ {}^{0}\mathbf{H}_{4} = {}^{0}\mathbf{H}_{1}\,{}^{1}\mathbf{H}_{2}\,{}^{2}\mathbf{H}_{3}\,{}^{3}\mathbf{H}_{4} = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & l_1\cos(\theta_1)+l_2\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & l_1\sin(\theta_1)+l_2\sin(\theta_1+\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

en este caso, es posible obtener la posición del efector final utilizando la información de traslación en la matriz homogénea, lo cual puede expresarse de la siguiente forma:

$$ \begin{bmatrix} h_x \\ h_y \end{bmatrix} = \begin{bmatrix} l_1\cos(\theta_1)+l_2\cos(\theta_1+\theta_2) \\ l_1\sin(\theta_1)+l_2\sin(\theta_1+\theta_2) \end{bmatrix} $$

luego de obtener la posición del efector final, se puede utilizarla para formular estructuras de control, donde se desee que el efector final realice una tarea predefinida.

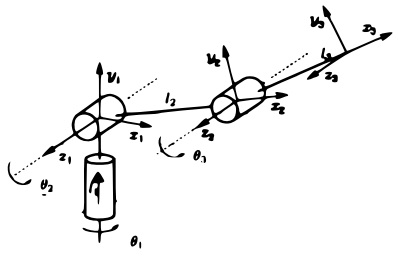
Para un mejor entendimiento de este ejemplo, se han implementado dichas operaciones en el lenguaje de programación Python. Si se requiere más información, se puede ingresar en el enlace https://bit.ly/4cQFLxp o ir directamente a la sección anexos (Anexo 4).

Manipulador Robótico

Adicionalmente, se puede analizar el caso de un robot manipulador de 3 grados de libertad que se mueve en el espacio. El sistema se presenta en la figura a continuación.

Figura 8

Representación de un manipulador robótico en 3D



Para encontrar la posición y orientación del sistema de referencia W^3, respecto a W^0 se deben formular las diferentes matrices homogéneas en función de las articulaciones del sis-

tema robótico que definen cada una de las transformaciones respectivas, definidas de la siguiente manera:

$$
{}^{0}\mathbf{H}_{1} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{1}\mathbf{H}_{2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{2}\mathbf{H}_{3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{3}\mathbf{H}_{4} = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{4}\mathbf{H}_{5} = \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{5}\mathbf{H}_{6} = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & 0 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{6}\mathbf{H}_{7} = \begin{bmatrix} 1 & 0 & 0 & l_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

una vez definidas estas matrices, es posible describir la transformación de a, así:

$$
{}^{0}\mathbf{H}_{4} = {}^{0}\mathbf{H}_{1}\ {}^{1}\mathbf{H}_{2}\ {}^{2}\mathbf{H}_{3}\ {}^{3}\mathbf{H}_{4}\ {}^{4}\mathbf{H}_{5}\ {}^{5}\mathbf{H}_{6}\ {}^{6}\mathbf{H}_{7}
$$

$$
{}^{0}\mathbf{H}_{7} = \begin{bmatrix} \cos(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3)\cos(\theta_1) & \sin(\theta_1) & (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\cos(\theta_1) \\ \sin(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_1)\sin(\theta_2+\theta_3) & -\cos(\theta_1) & (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\sin(\theta_1) \\ \sin(\theta_2+\theta_3) & \cos(\theta_2+\theta_3) & 0 & l_1+l_2\sin(\theta_2)+l_3\sin(\theta_2+\theta_3) \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

donde es posible obtener la posición y orientación del extremo operativo del sistema robótico:

$$
\begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix} = \begin{bmatrix} (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\cos(\theta_1) \\ (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\sin(\theta_1) \\ l_1+l_2\sin(\theta_2)+l_3\sin(\theta_2+\theta_3) \end{bmatrix}
$$

$$
{}^{0}\mathbf{R}_{3} = \begin{bmatrix} \cos(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3)\cos(\theta_1) & \sin(\theta_1) \\ \sin(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_1)\sin(\theta_2+\theta_3) & -\cos(\theta_1) \\ \sin(\theta_2+\theta_3) & \cos(\theta_2+\theta_3) & 0 \end{bmatrix}
$$

con la definición de la posición y orientación se puede encontrar la cinemática diferencial del sistema robótico; esto es de mucha utilidad para el desarrollo de algoritmos de control avanzados.

Para un mejor entendimiento de este ejemplo, se han implementado dichas operaciones en el lenguaje de programación Python. A fin de obtener más información, se puede ingresar en el enlace https://bit.ly/3XWqAPn no ir directamente a la sección anexos (Anexo 5).

También se recomienda la lectura del trabajo de (Waldron & Schmiedeler, 2008) titulado *Kinematics*

1.6 Cinemática Diferencial de Robots

La ubicación del efector final, además de poseer posición y orientación, dispone de velocidades de traslación y rotación

que le permiten realizar movimientos en el espacio cartesiano. Es importante mencionar que las velocidades del efector final son resultados de las velocidades de cada uno de los estados internos del robot. Para representar la velocidad del extremo operativo, se utiliza una herramienta matemática conocida como matriz Jacobiana, que permite realizar un mapeo lineal del espacio de estados internos al espacio de trabajo donde se desarrollan los movimientos del manipulador (Siciliano, 2009).

Esta matriz se define como la derivada del vector respecto a otro vector arbitrario. Por lo tanto, si se tiene la siguiente función vectorial $\mathbf{y} = \mathbf{f}(\mathbf{x})$, donde $\mathbf{x} \in \mathcal{R}^n$ y $\mathbf{y} \in \mathcal{R}^m$, se define la matriz Jacobiana de la siguiente forma:

$$\mathbf{J} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \in \mathcal{R}^{m \times n}$$

con base en esta formulación de la matriz, es posible determinar la relación entre las velocidades de las articulaciones y las del efector final del robot manipulador.

En función de la posición del efector final $\mathbf{h} = \begin{bmatrix} h_x & h_y & h_z \end{bmatrix} \in \mathcal{R}^3$, se encuentra el Jacobiano asociado a h, definido de la siguiente forma:

$$\frac{\partial \mathbf{h}}{\partial \mathbf{q}} = \mathbf{J}(\mathbf{q})$$

donde $\mathbf{q} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \end{bmatrix} \in \mathcal{R}^3$, es el vector de articulaciones del robot y es la matriz Jacobiana del sistema. En consecuencia, se tiene que:

$$\partial \mathbf{h} = \mathbf{J}(\mathbf{q}) \partial \mathbf{q}$$

$$\frac{\partial \mathbf{h}}{dt} = \mathbf{J}(\mathbf{q}) \frac{\partial \mathbf{q}}{dt}$$

$$\dot{\mathbf{h}} = \mathbf{J}(\mathbf{q}) \dot{\mathbf{q}}$$

la expresión presentada nos permite utilizar la matriz Jacobiana como un operador lineal entre el espacio de las velocidades articulares y el espacio de control o movimiento del robot manipulador.

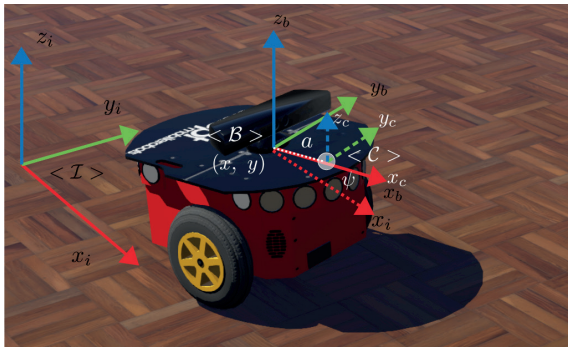
A continuación, se presentan ejemplos aplicados del robot móvil en el espacio y de un robot manipulador en el espacios, con los cuales se logrará entender a profundidad los conceptos de cinemática diferencial y Jacobiano del sistema.

Robot Móvil

Para definir la cinemática diferencial de un robot móvil, se considera la posición del sistema$\{\mathcal{C}\}$ respecto a$\{\mathcal{I}\}$. En la **Figura 9** se muestra cómo se representan los sistemas de referencia asociados a un robot móvil.

Figura 9

Representación de un Robot Móvil junto con sus respectivos sistemas de referencia



donde $\mathbf{h} = [h_x \quad h_y] \in \mathcal{R}^2$ representa la posición del punto de interés del robot móvil, expresado de una manera más clara así:

$$\begin{bmatrix} h_x \\ h_y \end{bmatrix} = \begin{bmatrix} a\cos(\psi) + x \\ a\sin(\psi) + y \end{bmatrix}$$

el vector de estados del sistema es $\mathbf{q} = [x \quad y \quad \psi] \in \mathcal{R}^3$ su derivada temporal $\dot{\mathbf{q}} = [\mu\cos(\psi) \quad \mu\sin(\psi) \quad \omega] \in \mathcal{R}^3$, que es el resultado de la proyección de la velocidad del robot móvil en el sistema $\{\mathcal{C}\}$ hacia $\{\mathcal{I}\}$. Bajo estas consideraciones, la cinemática diferencial se describe de la siguiente forma:

$$\begin{bmatrix} \dot{h}_x \\ \dot{h}_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & -a\sin(\psi) \\ 0 & 1 & a\cos(\psi) \end{bmatrix} \begin{bmatrix} \mu\cos(\psi) \\ \mu\sin(\psi) \\ \omega \end{bmatrix}.$$

$$\begin{bmatrix} \dot{h}_x \\ \dot{h}_y \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -a\sin(\psi) \\ \sin(\psi) & a\cos(\psi) \end{bmatrix} \begin{bmatrix} \mu \\ \omega \end{bmatrix}$$

esta descripción se escribe de manera sintética así:

$$\dot{\mathbf{h}} = \mathbf{J}(\psi)\dot{\mathbf{q}}$$

donde $\dot{\mathbf{h}} = [\dot{h}_x \quad \dot{h}_y] \in \mathcal{R}^2$ es la velocidad del robot móvil respecto al sistema de referencia $\{\mathcal{I}\}$ y $\mathbf{J}(\psi) \in \mathcal{R}^{2\times 2}$ y es la matriz Jacobiana, que permite realizar un mapeo lineal desde el espacio de control $\dot{\mathbf{q}}$ hacia h (Waldron & Schmiedeler, 2008).

Para un mejor entendimiento de este ejemplo se han implementado dichas operaciones en el lenguaje de programación Python. A fin de obtener más información, se puede ingresar

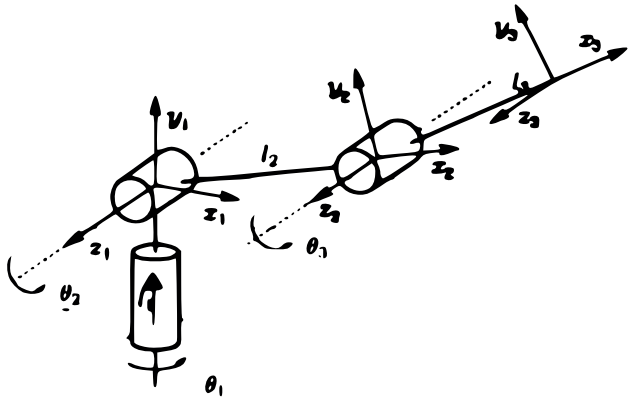
en el siguiente enlace https://bit.ly/45ZAguf o ir directamente a la sección anexos (Anexo 6).

Robot Manipulador de 3 Grados de Libertad

Para definir la cinemática diferencial de un robot manipulador de 3 grados de libertad, es necesario conocer la posición del extremo operativo en relación con el sistema de referencia $\mathcal{W}_3$ respecto a $\mathcal{W}_0$. El robot manipulador se presenta en la figura que se ubica a continuación.

Figura 10

Representación visual de un robot manipulador en un entorno tridimensional



donde $\mathbf{h} = [h_x \quad h_y \quad h_z] \in \mathcal{R}^3$ representa la ubicación del extremo operativo definido por:

$$\begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix} = \begin{bmatrix} (l_2 \cos(\theta_2) + l_3 \cos(\theta_2 + \theta_3)) \cos(\theta_1) \\ (l_2 \cos(\theta_2) + l_3 \cos(\theta_2 + \theta_3)) \sin(\theta_1) \\ l_1 + l_2 \sin(\theta_2) + l_3 \sin(\theta_2 + \theta_3) \end{bmatrix}$$

el vector de estados internos del robot es $\mathbf{q} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \end{bmatrix} \in \mathcal{R}^3$, mientras que la derivada temporal del vector de estado es $\dot{\mathbf{q}} = \begin{bmatrix} \dot{\theta}_1 & \dot{\theta}_2 & \dot{\theta}_3 \end{bmatrix} \in \mathcal{R}^3$. Con esta información, la cinemática diferencial puede ser escrita como:

$$\begin{bmatrix} \dot{h}_x \\ \dot{h}_y \\ \dot{h}_z \end{bmatrix} = \begin{bmatrix} -(l_2 \cos(\theta_2) + l_3 \cos(\theta_2 + \theta_3)) \sin(\theta_1) & -(l_2 \cos(\theta_2) + l_3 \cos(\theta_2 + \theta_3)) \sin(\theta_1) & -l_3 \sin(\theta_2 + \theta_3) \cos(\theta_1) \\ (l_2 \cos(\theta_2) + l_3 \cos(\theta_2 + \theta_3)) \cos(\theta_1) & (l_2 \cos(\theta_2) + l_3 \cos(\theta_2 + \theta_3)) \cos(\theta_1) & -l_3 \sin(\theta_1) \sin(\theta_2 + \theta_3) \\ 0 & 0 & l_3 \cos(\theta_2 + \theta_3) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix}$$

la formulación presentada se puede escribir de forma compacta así:

$$\dot{\mathbf{h}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$$

donde $\dot{\mathbf{h}} = \begin{bmatrix} \dot{h}_x & \dot{h}_y & \dot{h}_z \end{bmatrix} \in \mathcal{R}^3$ es la velocidad del extremo operativo respecto al sistema de referencia $\{\mathcal{I}\}$ y $\mathbf{J}(\mathbf{q}) \in \mathcal{R}^{3\times3}$ y es la matriz Jacobiana, que permite realizar un mapeo lineal desde el espacio de control $\dot{\mathbf{q}}$ hacia $\dot{\mathbf{h}}$.

Para un mejor entendimiento de este ejemplo se han implementado dichas operaciones en el lenguaje de programación Python. A fin de obtener más información, se puede ingresar en el enlace https://bit.ly/4cIvFPTb o ir directamente a la sección anexos (Anexo 7).

1.7 Integral Numérica

En sistemas robóticos, como manipuladores, robots móviles y vehículos aéreos, se tienen ecuaciones diferenciales no lineales para las que no existe una solución analítica.

En general, no es posible resolver ecuaciones diferenciales no lineales, las cuales pueden tener la siguiente forma general:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$$

solo en el caso especial de que sean lineales, es posible encontrar soluciones a ecuaciones diferenciales. Por lo tanto, es preciso conocer dichos sistemas y cómo evolucionan a lo largo del tiempo. Para ello, es necesario utilizar sistemas computacionales y representar el sistema en tiempo discreto, de manera que un computador pueda obtener la solución a estas ecuaciones diferenciales no lineales (Waldron & Schmiedeler, 2008).

Ecuaciones Diferenciales Discretas

Las ecuaciones diferenciales tienen su equivalente en tiempo discreto, el cual puede ser expresado de la siguiente manera:

$$\mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x}_k)$$

donde $\mathbf{f}_d$ es el equivalente de f en tiempo discreto. Además, existen diferentes formas para obtener $\mathbf{f}_d$, la más fácil y explícita de todas es la integración de Euler explícita (Griffiths & Higham, 2010). Así, se tiene que:

$$\mathbf{f}_d(\cdot) = \mathbf{x}_k + t_s\mathbf{f}(\mathbf{x}_k)$$

a esta formulación se la conoce como integración de Euler hacia delante.

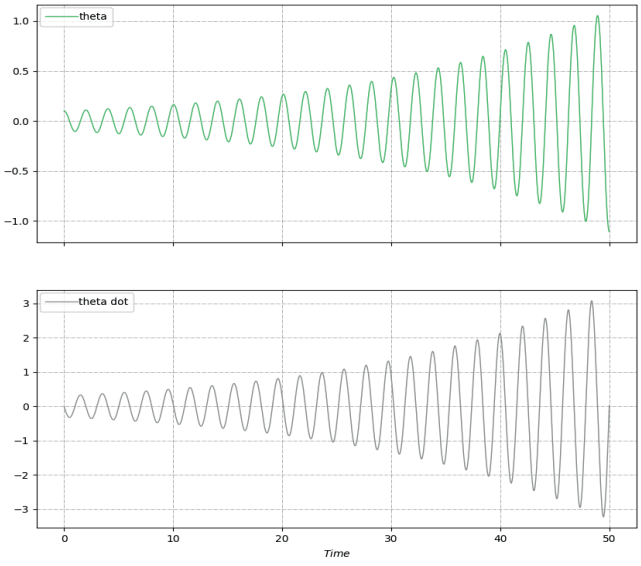
Se presenta un ejemplo práctico donde se puede ver el funcionamiento de la integración de Euler hacia delante, utilizando el modelo matemático de un péndulo en ecuaciones diferenciales que se definen así:

$$\ddot{\theta} = -\frac{g}{l} \sin(\theta)$$

donde g y l corresponden a la gravedad y la longitud del péndulo, respectivamente y la longitud del péndulo, respectivamente. Los estados del sistema se pueden representar como θ, $\dot{\theta}$ y $\ddot{\theta}$, y es el desplazamiento, velocidad y aceleración angular del péndulo.

Figura 11

Resultados numéricos obtenidos utilizando el método de integración de Euler



En la Figura 11 se presentan los resultados de la evolución del sistema propuesto, utilizando la integración de hacia delante de Euler.

A pesar de que la integración de Euler hacia adelante es comúnmente utilizada en el área de la robótica, presenta problemas relacionados con la estabilidad y la divergencia de los estados a medida que la simulación avanza.

Para un mejor entendimiento de este ejemplo, se han implementado dichas operaciones en el lenguaje de programación Python. A fin de obtener más información, se puede ingresar en el enlace https://bit.ly/3W7bgxU ir directamente a la sección anexos (Anexo 8).

Como ya se mencionó, lo expuesto anteriormente presenta problemas en el ámbito de la simulación de sistemas robóticos y control; por lo tanto, existe una alternativa al método de Euler y se trata de otro método, el de Runge-Kutta de cuarto orden (Ixaru & Vanden Berghe, 2004).

Así, según el método de Runge-Kutta de cuarto orden, se presenta la aproximación de $\mathbf{f}_d$:

$$\mathbf{f}_d(\cdot) = \mathbf{x}_k + \tfrac{1}{6} t_s(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

$$\mathbf{k}_1 = \mathbf{f}(\mathbf{x}_k)$$

$$\mathbf{k}_2 = \mathbf{f}(\mathbf{x}_k + \tfrac{1}{2}\mathbf{k}_1 t_s)$$

$$\mathbf{k}_3 = \mathbf{f}(\mathbf{x}_k + \tfrac{1}{2}\mathbf{k}_2 t_s)$$

$$\mathbf{k}_4 = \mathbf{f}(\mathbf{x}_k + \mathbf{k}_3 t_s)$$

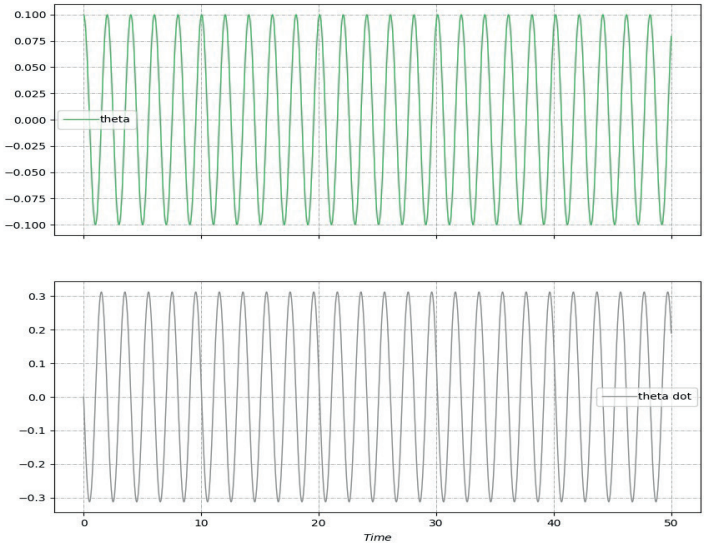
Para verificar la funcionalidad y ventajas de este método numérico de integración, se emplea el modelo matemático del péndulo en ecuaciones diferenciales. Además, se proporciona la formulación en código, que contribuirá a una mejor comprensión de la presente formulación https://bit.ly/3WaXL0f o también se puede ir directamente a la sección anexos (Anexo 9).

En la **Figura 12** se presentan los resultados utilizando la formulación de Runge-Kutta de cuarto orden.

Debido a la aproximación que se utiliza en el método de Runge-Kutta, el análisis iterativo de la función es capaz de encontrar una mejor solución a las ecuaciones diferenciales ordinarias. Asimismo, no se presentan problemas relacionados con la divergencia de los estados del sistema a medida que avanza el tiempo de simulación. Este método es ampliamente utilizado en robótica avanzada, donde una simulación de alta precisión es de vital importancia.

Figura 12

Resultados numéricos obtenidos utilizando el método de Runge-Kutta de cuarto orden



2

Simulador Robótico Webots

Los simuladores robóticos han presentado contribuciones de alta importancia para la comunidad robótica, principalmente en el área de reducción de costos de experimentación, ahorro de tiempo y la versatilidad para experimentar con diferentes formulaciones de control. Además, permiten la detección fácil de posibles errores en algoritmos de control en entornos simulados, evitando así pérdidas materiales en los sistemas robóticos (Román-Ibáñez, 2018).

El principal objetivo de los simuladores robóticos es reducir el tiempo de desarrollo y permitir al usuario cometer errores dentro de un ambiente altamente controlado. En la literatura se presentan varios simuladores que pueden ser utilizados, por ejemplo, CoppeliaSim (conocido como V-REP), Gazebo y Mujoco. Estos simuladores han demostrado sus ventajas y desventajas en diversas investigaciones (Koenig & Howard, 2004).

Sin embargo, las plataformas de simulación mencionadas anteriormente presentan una curva de aprendizaje compleja que, al momento de desarrollar sistemas robóticos, genera una alta inercia, aumentando así los tiempos de producción (Andaluz V. , 2016).

Con base en lo mencionado, se presenta Webots, un simulador robótico de código abierto desarrollado por Cyber-

botics Limited. Este simulador proporciona las herramientas necesarias para prototipado, modelación, programación y simulación de sistemas robóticos basados en Open Dynamics Engine. Además, en estos sistemas se puede usar una amplia gama de sensores y actuadores. Finalmente, las tareas de programación de sistemas robóticos se pueden realizar en diferentes lenguajes como Matlab, C++ y Python (Michel, 2004).

Webots es una plataforma que presenta un buen desempeño para propósitos de investigación y educación relacionados con las siguientes áreas:

- Prototipado de robots móviles.
- Locomoción de robots abarcando áreas de humanoides, cuadrúpedos y manipuladores robóticos.
- Robots colaborativos o múltiples sistemas robóticos.
- Enseñanza de sistemas robóticos

Adicionalmente, el conocimiento previo que el usuario debe poseer para usar de forma correcta el simulador robótico se contempla en los siguientes enunciados:

- Un entendimiento básico de uno o más de los siguientes lenguajes de programación MATLAB, Python, Java y C + +, con lo cual se desarrollan los controladores de los sistemas robóticos.
- Diseño y modelado 3d, con los que se pueden desarrollar robots a medida del usuario y realizar pruebas dentro del entorno de simulación.

Por esto, en el siguiente capítulo se presenta la correcta instalación del simulador robótico Webots, la configuración de variables de entorno que permiten el uso correcto de Python como lenguaje de programación para sistemas robóticos, una

breve introducción al simulador que abarca funcionalidades básicas, así como los sensores y actuadores disponibles.

2.1 Instalación de Webots

Para llevar a cabo la instalación de Webots se recomienda utilizar un sistema operativo basado en GNU-Linux, ya que la mayoría de los sistemas robóticos de investigación se fundamentan en dicha arquitectura, dado que presenta un mayor soporte y una mayor abstracción. Por lo tanto, se recomienda el uso del sistema operativo Ubuntu LTS (Focal Fossa), que puede ser descargado directamente del enlace https://bit.ly/4cjR95f. Una alternativa a utilizar un sistema operativo nativo en Linux es el uso del Subsistema de Windows para Linux (WSL), el cual puede llegar a ser configurado bajo una serie de tutoriales disponibles en la web.

Una vez se disponga el correcto sistema operativo, se procede a descargar la versión Webots 2023a, disponible en el enlace https://bit.ly/3XQAHVH. Luego de descargar el archivo, se instala el simulador; el proceso de instalación puede llevarse a cabo de diferentes formas, una de las recomendadas es el uso del terminal. A continuación, se presentan los pasos que debe seguir el usuario para realizar una correcta instalación:

- En la terminal de Linux, es necesario ubicar el archivo descargado. Los comandos 'cd' y 'ls' permiten al usuario moverse entre directorios y mostrar los archivos de dicho directorio, respectivamente. En la **Figura 13** se muestra el directorio de trabajo y el archivo descargado (webots_2023a_amd64.deb).

Figura 13

Ubicación del archivo de instalación desde la terminal en el espacio de trabajo

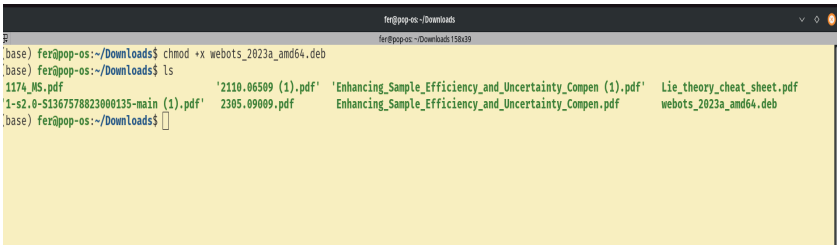
Nota. Esta imagen muestra cómo se accede a la ubicación del archivo de instalación desde la terminal dentro del espacio de trabajo.

A continuación, se presentan los comandos utilizados:

- cd Downloads
- Ls
- Chmod -x webots_2023a_amod64.deb
    - Seguidamente, se conceden permisos de ejecución al archivo con el siguiente comando 'chmod +x webots_2023a_amd64.deb'. De esta manera, el archivo cambiará a color verde, demostrando que ahora es posible ejecutarlo dentro del sistema operativo. El resultado se presenta en la **Figura 14**. Esta acción es crucial para permitir que el archivo de instalación se ejecute correctamente en el sistema operativo.

Figura 14

Incluyendo los permisos de ejecución del archivo de instalación

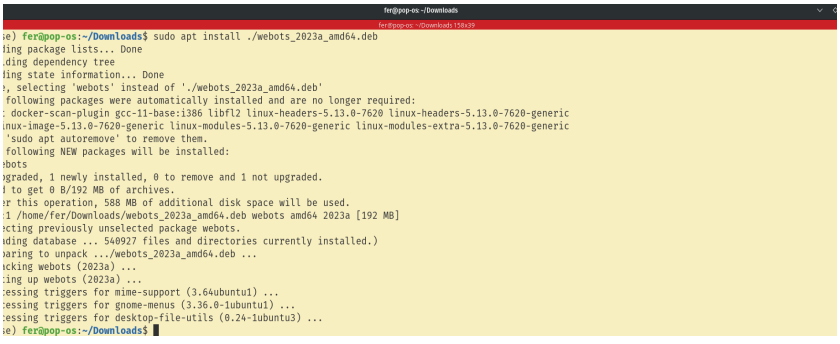


A continuación, se presentan los comandos utilizados:

- Chmod -x webots_2023a_amod64.deb
- Ls
    - Para completar la instalación, se debe ejecutar el comando 'sudo apt install ./webots_2023a_amd64.deb'. Con este comando, el simulador robótico será instalado. Puedes verificar este procedimiento en la **Figura 15** que se presenta a continuación.

Figura 15

Proceso de instalación de Webots a través de la terminal



Nota. Esta acción implica la utilización de comandos específicos en la terminal para instalar el *software Webots* en el sistema operativo.

A continuación, se presentan los comandos utilizados:

- Sudo apt install ./webots_2023a_amd64.deb
    - Finalmente, es posible ejecutar el simulador robótico ingresando el siguiente comando en el terminal 'webots'. El simulador se ejecutará inmediatamente, permitiendo al usuario utilizar las distintas funcionalidades; esto se aprecia en la figura ubicada a continuación.

Figura 16

Proceso que implica utilizar comandos específicos en la terminal para iniciar la ejecución del software Webots en el sistema operativo

A continuación, se presentan los comandos utilizados:

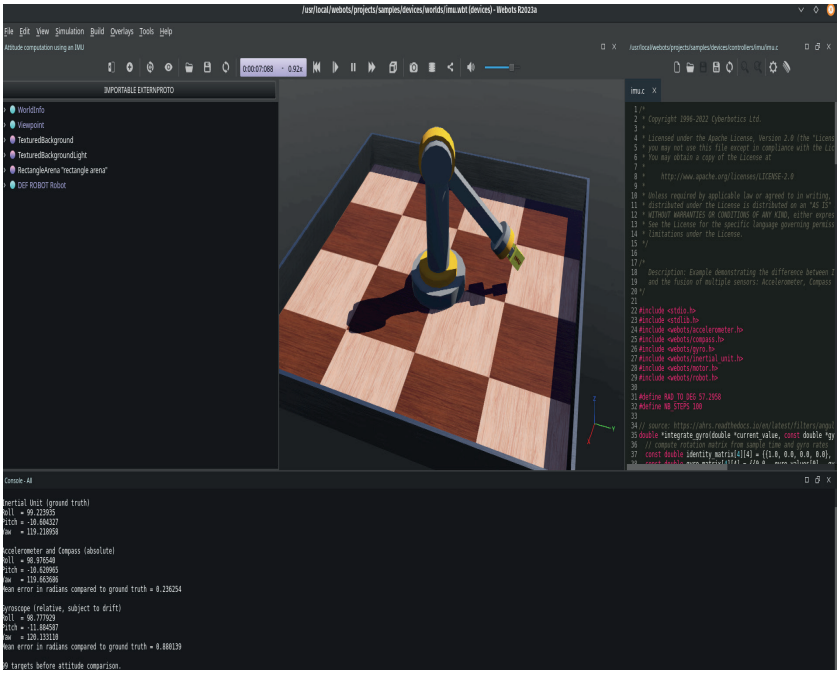
- Webots

Luego de realizar los pasos presentados anteriormente, se puede usar el simulador. El usuario no tendrá problemas para familiarizarse con la interfaz, ya que Webots es una plataforma de simulación muy intuitiva y no presenta la misma curva de aprendizaje en comparación con otros simuladores. En la **Figu-**

ra 17 se presenta el panel principal del simulador robótico. En secciones posteriores se abordará en profundidad cada una de las secciones del panel.

Figura 17

Entorno de simulación de Webots



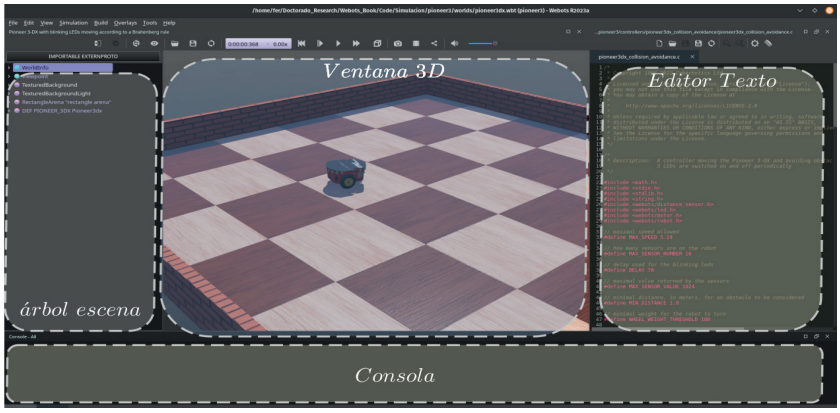
2.2 Interfaz de Usuario

En esta sección, se presenta en detalle la interfaz de usuario de Webots, la cual está compuesta por cuatro ventanas principales: la **ventana 3D**, que permite al usuario interactuar con la simulación; el **árbol de escena**, que representa jerárquicamente los elementos presentes dentro del simulador; el **editor de**

texto, que permite editar los controladores de código fuente de la simulación en diferentes lenguajes de programación; y finalmente, la **consola**, donde se pueden mostrar resultados de compilación y de controladores. En la **Figura 18** se pueden ver mejor los elementos de la interfaz de usuario.

Figura 18

Ventanas de navegación que acompañan al entorno de simulación de Webots



Adicionalmente, la interfaz de usuario presenta ocho menús fundamentales: **File, Edit, View, Simulation, Build, Overlays, Tools** y **Help**. El nombre asignado a estos elementos puede variar según el idioma seleccionado en el simulador Webots. En la siguiente sección se presenta una breve descripción de cada uno de los menús mencionados previamente.

Menú File

En el **menú File** se presentan elementos que le permiten al usuario guardar modificaciones y añadir nuevos elementos, entre otras funcionalidades.

- Dentro de este menú se encuentra un submenú llamado **New**, que tiene estas funcionalidades:
- **New Project Directory:** Permite al usuario crear un nuevo directorio de trabajo, que incluye elementos como **world, controlador, plugins** y **elementos de simulación**.
- **New World File:** Permite al usuario crear nuevos ambientes de simulación en el proyecto de simulación.
- **New Robot Controller:** El usuario puede generar nuevos controladores para los robots dentro del ambiente de simulación, seleccionando entre los diversos lenguajes de programación compatibles con Webots, tales como Matlab, C++, Java y Python.
- **New Physics Plugin:** Posibilita la creación de complementos de física dentro del proyecto de simulación.

A continuación, se explican brevemente los submenús del **menú File**.

- **Open World:** Permite al usuario seleccionar entornos de simulación.
- **Open Recent World:** Almacena una lista de entornos de simulación ejecutados recientemente; esto permite al usuario trabajar eficientemente entre diferentes entornos de simulación.
- **Open Sample World:** Despliega una lista que contiene los ejemplos disponibles dentro del simulador Webots.
- **Save World:** Guarda la información del entorno de simulación. Es importante tener en cuenta que el

contenido se sobrescribirá con el mismo nombre del archivo, por lo que no se tendrán respaldos de modificaciones previas. Por lo tanto, esta opción debe ser utilizada con precaución.

- **Save World As:** Guarda la información del entorno de simulación y le proporciona al usuario la posibilidad de asignar un nuevo nombre al proyecto.
- **Reload World:** Reinicia todos los parámetros e inicia la simulación bajo las condiciones iniciales respectivas. Esta funcionalidad es de gran importancia cuando se realizan modificaciones dentro del simulador, como las características de los sensores.
- **Reset Simulation:** Restaura la simulación a las condiciones iniciales. A diferencia de **Reload World**, esta funcionalidad no reinicia completamente la simulación, pero se utiliza con más frecuencia debido a la velocidad de respuesta del simulador.
- **New Text File:** Permite la creación de un archivo de texto plano que puede utilizarse para los controladores del entorno de simulación.
- **Open Text File:** El usuario puede abrir archivos de controladores, ya que son archivos de texto con diversas extensiones correspondientes a distintos lenguajes de programación.
- **Save Text File:** Es posible guardar archivos de texto plano con diversas extensiones, por ejemplo '.python,' '.java.'
- Finalmente, dentro del **menú File** se disponen de funcionalidades como '**Take Screenshot**', '**Make Movie**' y '**Share**', con las cuales es posible guardar los ambientes de simulaciones en formato multimedia.

Menú Edit

En este menú se presentan funcionalidades relacionadas estrechamente con la edición de archivos de texto, como copiar, pegar y cortar.

Menú View

En esta sección del menú, el usuario puede modificar la representación visual de la simulación, como diferentes puntos de vista, renderizado de ciertos elementos, entre otros. A continuación, se presenta una breve descripción de las funcionalidades presentes en este menú.

Follow Object: Permite al usuario utilizar una cámara fija como punto de vista para hacer un seguimiento de un objeto dentro del entorno de simulación. Por ejemplo, si se tiene un robot que se mueve sobre un plano, al usar Follow Object, dicho robot siempre se mantendrá perfectamente centrado.

Restore Viewpwoint: Restablece los valores por defecto del punto de vista hacia el usuario.

Change View: Permite al usuario mover el punto de vista hacia la posición más conveniente, detectada por el simulador Webots.

Projection: Permite al usuario seleccionar entre Perspective Projection y Orthographic Projection. La Perspective Projection corresponde a una representación natural donde los obje-

tos aparecen más pequeños mientras están más lejos del punto de vista. En cambio, la Orthographic Projection no se ve afectada por la distancia desde el punto de vista a los objetos, manteniendo su tamaño constante. Este modo de operación suele ser utilizado durante la fase de modelado.

Rendering: Ofrece al usuario la posibilidad de seleccionar entre Plain Rendering y Wireframe. En Plain Rendering, los objetos se representan con sus materiales, texturas y colores correspondientes. Mientras que en Wireframe Rendering, solo se muestran las figuras primitivas sin colores ni texturas, esto resulta útil al diseñar mallas de contacto para los objetos.

Optional Rendering: En esta sección del menú, se brinda la posibilidad de agregar u omitir información o elementos en el ambiente de simulación. Esta opción se utiliza, comúnmente, para comprender el funcionamiento de ciertos sensores presentes en el simulador robótico, como cámaras y sus respectivas proyecciones en el plano de la imagen digital. A continuación, se detallan los elementos que se pueden agregar u omitir en la simulación:

- **Show Coordinate System:** Esta opción muestra u oculta el sistema de referencia global utilizando los colores: rojo para el eje "x", verde para el eje "y" y azul para el eje "z".
- **Show All Bounding Objects:** Esta opción permite al usuario visualizar las mallas de colisión de los objetos dentro del ambiente de simulación. Las mallas de colisión están representadas por líneas blancas y cambian a azul cuando se detecta una colisión con estos elementos.

- **Show Contact Points**: Esta opción permite mostrar u ocultar los puntos de colisión de los elementos presentes en la simulación. Los puntos de contacto que no generen fuerzas de contacto no son representados. Las fuerzas de contacto solo son generadas por objetos con físicas, es decir, aquellos que tienen masa y matriz de inercia dentro del simulador.
- **Show Connector Axes**: Esta opción permite representar los ejes de conexión, mostrando las alineaciones de rotación de ciertos elementos, como ruedas o elementos de un manipulador robótico.
- **Show Joint Axes**: Esta opción muestra u oculta los ejes de conexión entre elementos.
- **Show RangeFinder Frustums:** Permite visualizar el rango de operación de sensores como cámaras y sensores Lidar, así facilita la comprensión de dichos sensores por parte del usuario.
- **Show Lidar Rays Paths:** Permite al usuario mostrar u ocultar los rayos del sensor Lidar, así como el campo de operación; esto facilita la comprensión de este tipo de sensor.
- **Show Lidar Point Cloud:** Muestra u oculta la proyección del sensor Lidar sobre los objetos en color azul, proporcionando al usuario información sobre la ubicación del objeto dentro del campo de operación del sensor Lidar.
- **Show Distance Sensor Rays:** Muestra u oculta los rayos del sensor de distancia, que pueden o no estar asociados a un sensor Lidar, cámaras con profundidad y sensores de distancia.

- **Show Center Of Mass:** Muestra u oculta el centro de masa de los elementos dentro del ambiente de simulación, representado como un punto azul oscuro.

Finalmente, el último elemento del **menú View** es el submenú **Scene Interactions**, donde se pueden habilitar o deshabilitar ciertas interacciones de la escena 3D con el usuario, como la selección de objetos para su posterior traslado a una nueva ubicación. A continuación, se muestran las opciones disponibles en este submenú:

- **Lock Viewpoint:** Cuando esta opción está habilitada, previene el cambio de posición y orientación del punto de vista del usuario. Esto resulta particularmente útil si no se desean cambios indeseables en el punto de vista del usuario de la simulación.
- **Disable Object Move:** Previene que se realicen movimientos o traslaciones indeseadas de los elementos de la simulación. La pose de dichos elementos puede ser modificada directamente en el **árbol de escena**.
- **Disable Force and Torque:** Cuando esta opción se encuentra habilitada, previene que se apliquen fuerzas o torques directamente a los objetos. Esto es particularmente útil en competiciones donde los participantes deben completar una serie de desafíos sin hacer trampa o mover objetos manualmente.

Menú Simulation

Este menú se utiliza para controlar la simulación bajo diferentes opciones. A continuación, se describen brevemente cada uno de sus elementos:

- **Pause:** Detiene la simulación, conservando la información en ese instante.
- **Step:** Ejecuta un paso de tiempo en el entorno de simulación. La duración de este paso de simulación puede ser definida por el usuario según su conveniencia y necesidades.
- **Real-time:** Ejecuta la simulación en tiempo real hasta que el usuario la pause o reinicie.
- **Fast:** El usuario tiene la opción de ejecutar la simulación tan rápido como sea posible, lo cual es útil en situaciones de entrenamiento reforzado donde se realizan ciclos de entrenamiento a altas velocidades.
- **Rendering:** Permite al usuario habilitar o deshabilitar el renderizado de gráficos. Desactivar los gráficos permite ejecutar simulaciones más rápidamente de lo habitual; esto es muy útil en algoritmos de entrenamiento reforzado.

Menú Build

Esta sección del menú está relacionada con el proceso de compilación de controladores de robots escritos en lenguaje de programación C++ o C, los cuales requieren un proceso de compilación antes de su ejecución. En caso de que el usuario esté utilizando un lenguaje de programación que no requiera compilación previa, esta sección del menú no será utilizada.

Menú Overlays

Esta sección del menú presenta opciones relacionadas con dispositivos como cámaras, pantallas y sensores de distancia basados

en “Rangefinder”. Algunas de las opciones de este menú solo están activas cuando los elementos son seleccionados desde la ventana 3D. A continuación, se muestran las opciones disponibles:

- **Camera Devices:** En este submenú se muestran todos los dispositivos tipo cámara que se encuentran en el robot u objeto. Permite al usuario habilitar o deshabilitar el campo de funcionamiento de dichos dispositivos.
- **RangeFinder Devices:** Aquí se encuentran todos los dispositivos de tipo rangefinder que están en el robot u objeto. Además, permite activar o desactivar el campo de operación de dichos dispositivos para que el usuario tenga un mejor entendimiento de ellos.
- **Display Devices:** Dentro del simulador Webots, es posible simular dispositivos como pantallas, actuando como monitores. Este submenú permite al usuario identificar la cantidad de dispositivos de este tipo presentes en un robot.

Menú Tools

Esta sección del menú permite al usuario abrir o cerrar las distintas ventanas del simulador, tales como:

- **3D View:** Muestra u oculta la ventana 3D, que permite interactuar con la simulación.
- **Scene Tree:** Muestra el Scene Tree, donde el usuario puede editar los elementos presentados en el ambiente de simulación.
- **Text Editor:** Este submenú abre el editor de texto de Webots, que puede ser usado para editar y compilar controladores.

- **Documentation:** Muestra la documentación oficial de Webots, donde es posible realizar consultas puntuales del simulador.
- **Restore Layout:** Es posible que por accidente el usuario haya ocultado alguna de las ventanas del simulador; por lo tanto, con este submenú podrá recuperar la configuración predeterminada de la distribución de ventanas del simulador.
- **Clear all Consoles:** Limpia la salida de consola de los entornos de simulación.
- **New Console:** Crea una nueva consola mostrando por defecto los resultados de los controladores.
- **Edit Physics Plugin:** Permite al usuario abrir el código fuente de los complementos de físicas del ambiente de simulación.

Menú Help

En esta sección del menú, el usuario podrá acceder a la documentación e información general del simulador Webots.

Hasta aquí se han presentado los menús básicos que permiten al usuario utilizar adecuadamente el simulador Webots. Además, en esta sección se detalla la interfaz de usuario de Webots, la cual consta de la **ventana 3D**, el **árbol de escena**, el **editor de texto** y, finalmente, la **consola**.

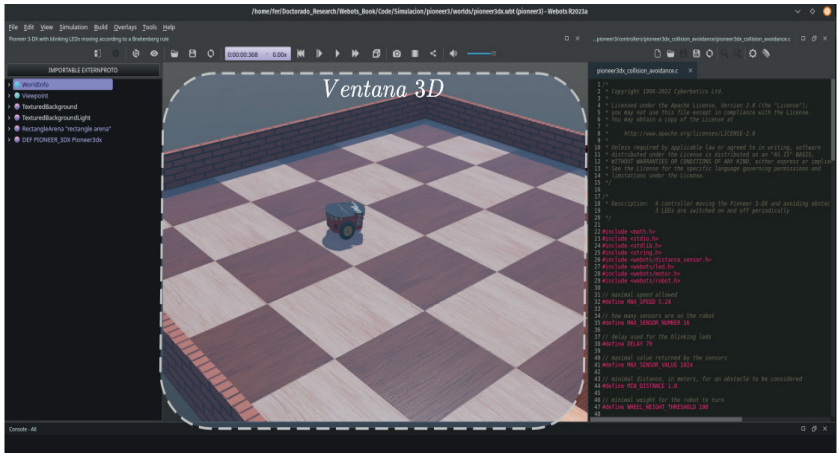
Comprender correctamente la interfaz de usuario facilita un flujo de trabajo adecuado y acelera la generación de entornos de simulación.

Ventana 3D

A continuación, se detallan las funcionalidades presentes en la **Ventana 3D**. En la **Figura 19** se representa la **Ventana 3D** en la interfaz de usuario.

Figura 19

Interfaz de Webots, con especial énfasis en las funcionalidades 3D



Seleccionando objetos

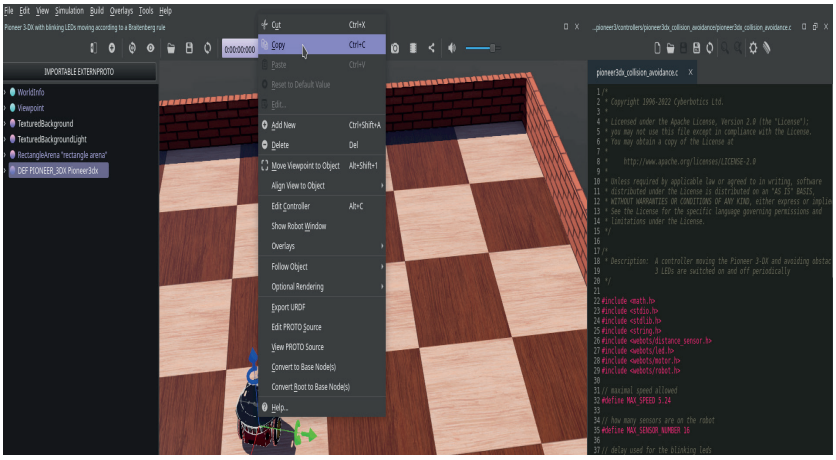
El usuario puede seleccionar objetos con un simple clic izquierdo del ratón sobre los elementos del entorno de simulación. Una vez que el objeto ha sido seleccionado, se resalta con líneas blancas en su contorno. Estas líneas se vuelven púrpuras en caso de colisiones y, finalmente, azules si no se respetan, indicando intersecciones entre diferentes objetos.

Tan pronto como el usuario seleccione los objetos, se desplegará un menú en el que podrá realizar diversas acciones sobre dicho objeto, como copiar, mover, entre otras. El despliegue

de este menú se muestra en la Figura 20. Esta función permite a los usuarios seleccionar objetos y ajustar sus propiedades dentro del entorno de simulación de Webots.

Figura 20

Selección de objetos y propiedades en Webots



El usuario puede modificar la representación de la escena al desplazar el ratón mientras presiona uno de sus botones. A continuación, se detallan los posibles movimientos:

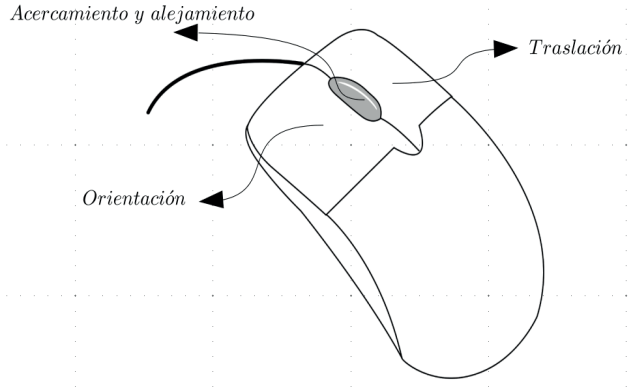
- **Rotación de la cámara:** Dentro de la ventana 3D, al desplazar el ratón, presione sobre uno de los objetos con el botón izquierdo. Esta secuencia de pasos rotará el punto de vista hacia el objeto seleccionado, permitiendo tener diferentes perspectivas. Si el usuario no presiona un objeto y selecciona la escena de fondo, la cámara rotará en su propio eje, lo que permitirá al usuario realizar rotaciones a lo largo de los diferentes ejes.

- **Traslación de la cámara:** Dentro de la ventana 3D, al desplazar el ratón, presione el botón derecho. Esto provocará la traslación del punto de vista de la cámara.
- **Acercamiento de la escena:** Dentro de la ventana 3D, al desplazar el ratón, presione el botón central. Esto resultará en un acercamiento a la escena o su efecto inverso.

En la **Figura 21** se ilustra gráficamente la ubicación de los botones: izquierdo, derecho y central, así como la función que desempeñan dentro del simulador Webots. Esta es una herramienta importante para navegar y manipular objetos en el entorno de simulación.

Figura 21

Atajos del ratón utilizados en Webots



Desplazando objetos

Actualmente, Webots ofrece dos formas de desplazar y rotar objetos dentro del ambiente de simulación. La primera

consiste en utilizar los ejes asociados a cada objeto, y la segunda implica el uso de atajos mediante el uso del teclado.

Cuando se selecciona un objeto en la ventana 3D, además de las líneas blancas que muestran el contorno del objeto, se presenta un sistema de referencia adicional en el centro de masa de dicho elemento. Este sistema de referencia puede utilizarse para trasladar y rotar el objeto a lo largo de los diferentes ejes. El usuario debe seleccionar uno de estos y trasladar o rotar el objeto a la posición u orientación deseada. En la **Figura 22** se ofrece una representación más clara de este procedimiento. Esta función permite a los usuarios mover y girar objetos dentro del entorno de simulación de Webots.

Figura 22

Traslación y rotación de objetos en Webots

Los atajos mediante el uso del teclado, que permiten trasladar y rotar objetos dentro del simulador, se pueden clasificar en:

- **Traslación:** El usuario puede trasladar los objetos realizando movimientos paralelos al suelo al ejecutar

la siguiente secuencia: mantener presionada la tecla 'Shift', presionar el botón izquierdo del *mouse* y mover este último.

- **Rotación:** El usuario puede rotar los objetos siguiendo la siguiente secuencia: mantener presionada la tecla 'Shift', presionar el botón derecho del *mouse* y moverlo, lo que generará rotaciones en el objeto.

Aplicar fuerzas a los objetos

Para aplicar fuerzas a los objetos, el usuario debe posicionar el puntero del *mouse* en el lugar donde desee aplicar la fuerza y mantener presionada la tecla 'Alt' mientras acciona el botón izquierdo y desplaza el *mouse*. Este desplazamiento será proporcional a la fuerza aplicada. En algunos sistemas de Linux, es necesario presionar las teclas 'Ctrl' y 'Alt' simultáneamente para aplicar fuerzas a los objetos.

Aplicar torques a los objetos

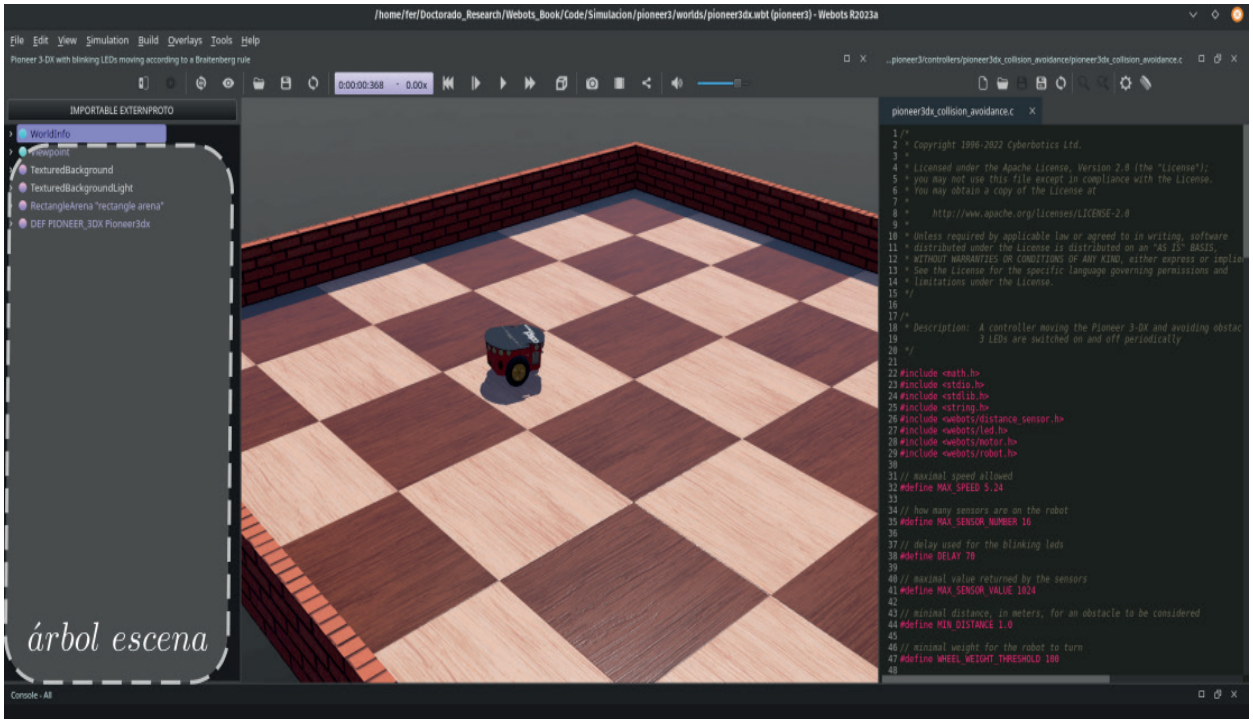
El usuario puede aplicar torques a los objetos posicionando el puntero del *mouse* en el lugar deseado para el torque y manteniendo presionada la tecla 'Alt' mientras acciona el botón derecho y desplaza el *mouse*. Este desplazamiento será proporcional al torque aplicado. En algunos sistemas de Linux, será necesario presionar las teclas 'Ctrl' y 'Alt' simultáneamente para aplicar torques a los objetos.

Árbol de escena

A continuación, se detallan las funcionalidades presentes en el **árbol de escena**, que le permite al usuario conocer los elementos que conforman la simulación, como robots y elementos del ambiente. En la **Figura 23**. se muestra la interfaz de Webots junto con la sección del árbol de escena que corresponde a los elementos y objetos presentes en la simulación.

Figura 23

Webots y la sección correspondiente en el árbol de escena

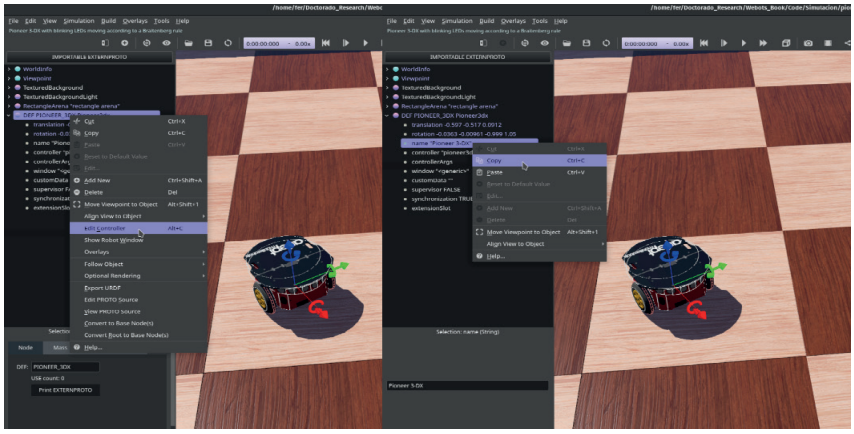


El árbol de escena de Webots se fundamenta en la estructura VRML97, donde se muestra una lista de nodos que contienen archivos de diversos tipos, numéricos y de texto, así como otros nodos. En esta sección, se proporcionará una breve introducción al árbol de escena con su respectivo formato VRML97.

Esta sección incluye un submenú que puede contener diversas funcionalidades, desde cortar y copiar elementos hasta reiniciar parámetros del ambiente de simulación. Si el nodo seleccionado es un robot, es posible acceder a toda la información del mismo, así como a sus controladores, en el editor de texto. En la **Figura 24** se proporciona más información sobre el mencionado submenú aplicado tanto a un objeto del ambiente como a un robot.

Figura 24

Submenú del árbol de escena en Webots aplicado tanto a un elemento como a un robot en la simulación

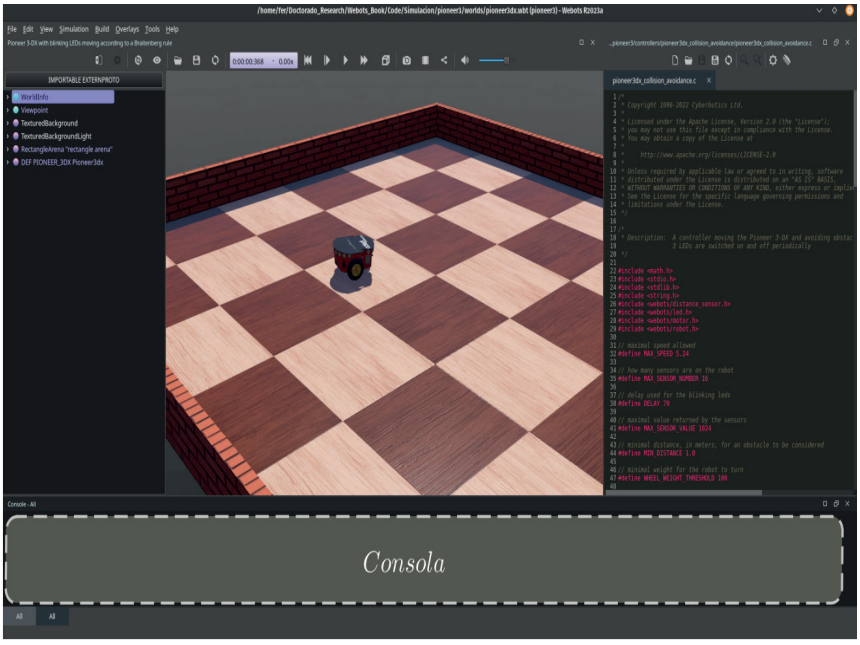


Consola

La consola permite al usuario visualizar posibles advertencias, errores de compilación e, incluso, advertencias dentro del sistema de físicas de Webots. Para asegurar que la información se presente correctamente en la consola, los datos provenientes de los controladores no se muestran al instante. En cambio, se espera y se agrupan por controladores, mostrándose al final de un paso de simulación. En la **Figura 25** se puede encontrar más información sobre la consola dentro de la interfaz del simulador.

Figura 25

Interfaz de Webots junto con la sección de consola o terminal correspondiente



En consecuencia, si se tienen un robot A y un robot B, la información por consola del robot A se presenta antes que la información del robot B. Por defecto, hay una sola consola disponible dentro del simulador, pero se puede abrir una nueva consola en el **menú Tools**. Cada una de las consolas son completamente independientes y tienen sus propios nombres y procesos adjuntos a ellas.

2.3 Conceptos Básicos Dentro de Webots

Al iniciar el uso de Webots, los usuarios deben familiarizarse con ciertos conceptos que se utilizarán de manera extensa a lo largo del texto.

Mundo "world"

Dentro del entorno de Webots, se encuentra el archivo world ".wbt", en el que se definen uno o varios robots dentro de un ambiente de simulación. Este archivo contiene información y descripciones (diseño 3D, posición, orientación y geometrías) de cada uno de los robots presentes, así como los controladores de movimiento de cada robot. Por último, incluye complementos de física que nos permiten modificar o agregar comportamientos adicionales a la simulación, como fluidos y vientos.

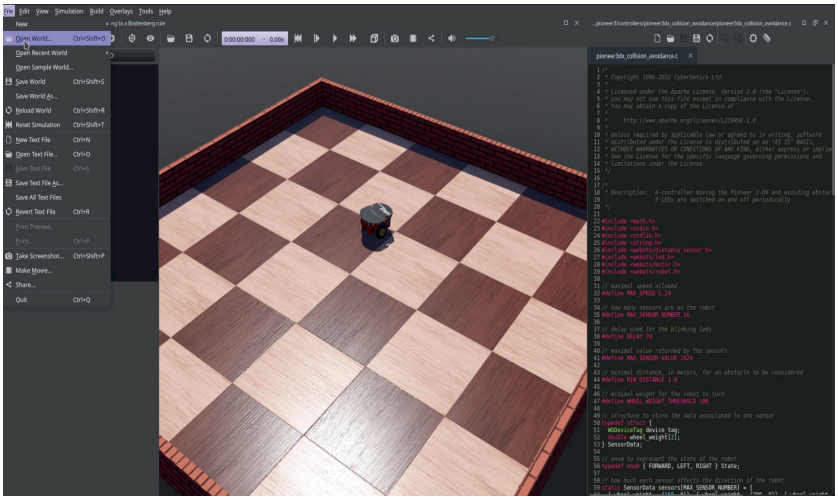
Para abrir un entorno de simulación que tenga en cuenta el archivo "world" de Webots, es preciso seguir los siguientes pasos:

- Una vez que esté ejecutando el simulador Webots, haga clic en el menú **'File'** y luego en el submenú **'Open World'**, lo que le permitirá seleccionar el archivo ".wbt" deseado. En la **Figura 26** se muestra la ejecución de las medidas previamente mencionadas. Se muestra el proceso de acceder a un nuevo archivo

dentro de Webots, una acción fundamental para cargar modelos, configuraciones o escenarios en el entorno de simulación.

Figura 26

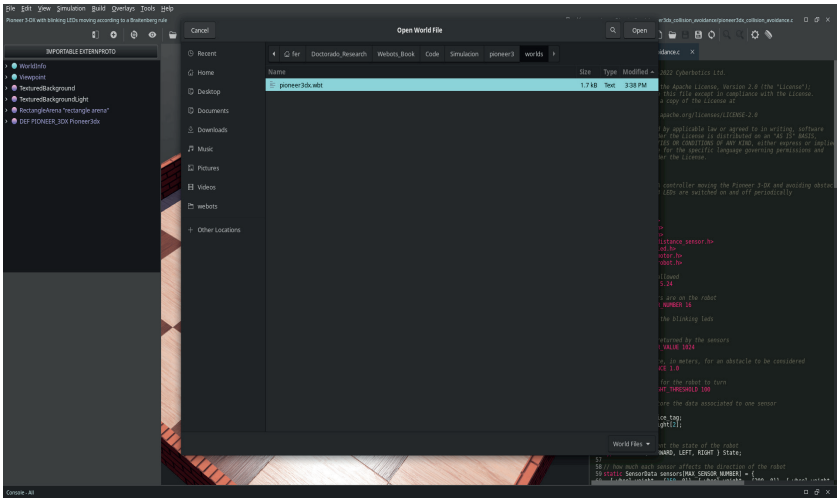
Ilustración del acceso a un nuevo archivo en Webots



Posteriormente, proceda a seleccionar el archivo ".wbt" correspondiente. En este caso, el archivo está disponible en el enlace https://bit.ly/45RcIHP. Después, el usuario debe ubicarse dentro de la carpeta 'Simulación', luego en 'pioneer3' y, finalmente, seleccionar el archivo "pioneer3dx.wbt". En la **Figura 27** se proporcionan detalles adicionales.

Figura 27

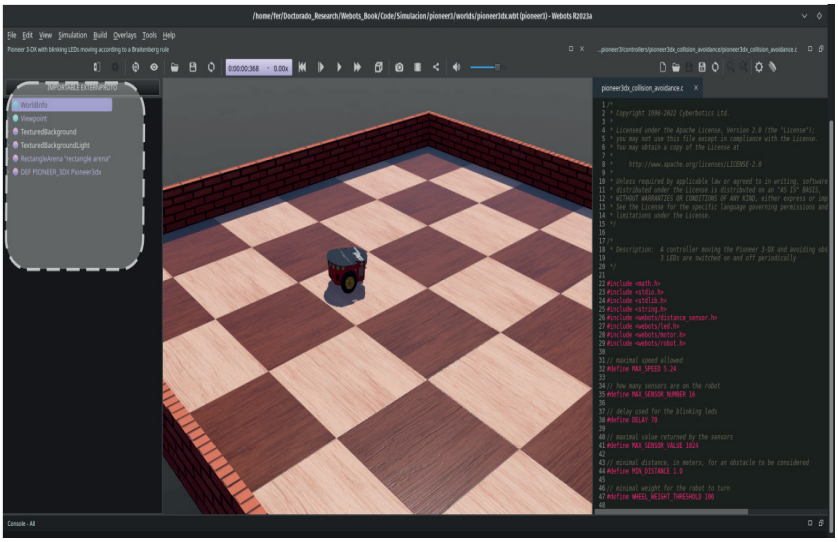
Ventana auxiliar se emplea en Webots para facilitar la elección de un nuevo archivo, como modelos de robots o escenarios de simulación



Una vez completado este paso, es posible visualizar los elementos presentes dentro del entorno de simulación. En la **Figura 28** se muestran dichos elementos del árbol de escena. Este término se refiere a los atributos y objetos que están representados en la estructura jerárquica del árbol de escena.

Figura 28

Propiedades y elementos presentes en el árbol de escena del entorno de simulación



Controlador

Un controlador es un programa capaz de comandar a los robots dentro del entorno de simulación. Este programa puede ser escrito en diferentes lenguajes de programación como MATLAB, Python, Java y C++. Al iniciar la simulación, Webots ejecuta los controladores para cada robot; además, es importante destacar que diferentes robots pueden ejecutar el mismo controlador.

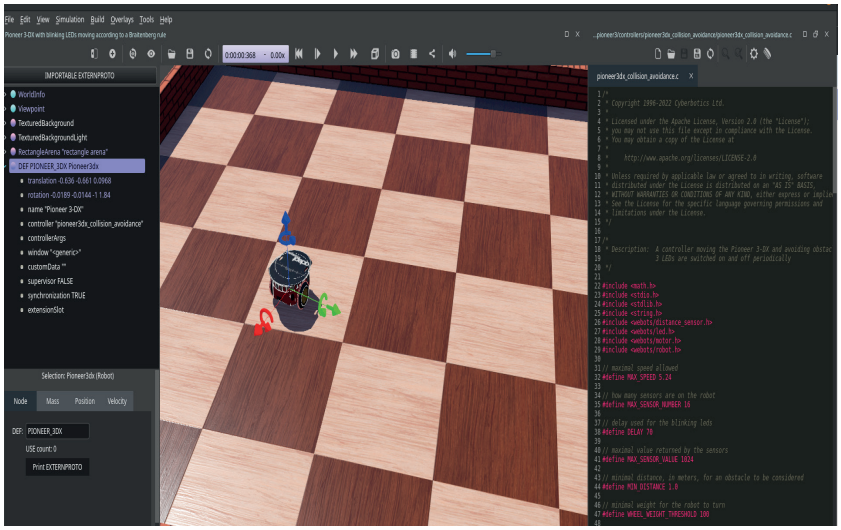
Para identificar y asignar diferentes controladores a un sistema robótico, se deben seguir los siguientes pasos:

- Una vez seleccionado el archivo de trabajo 'pioneer-3dx.wbt', haga clic sobre el robot. Este robot se en-

cuentra dentro del árbol de escena. En la **Figura 29** se proporciona más información.

Figura 29

Acción de asignar controladores a objetos o elementos específicos dentro de la estructura jerárquica del árbol de escena



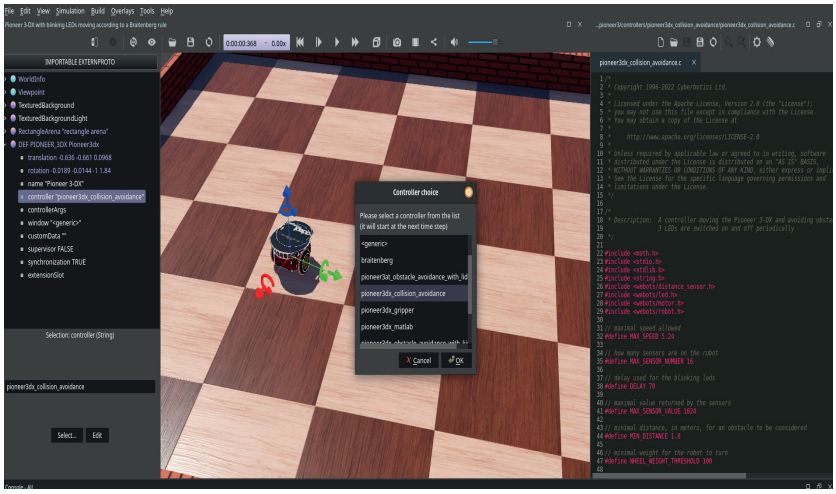
Una vez completado este paso, se despliega una serie de pestañas donde se puede visualizar información del robot, como la traslación o posicionamiento respecto al sistema inercial, orientación o rotación respecto a un sistema de referencia, controlador y muchas otras características más.

- La pestaña de controlador nos permite conocer qué controlador ha sido asignado al robot y nos brinda la posibilidad de cambiarlo o reemplazarlo por algún otro controlador diseñado por el usuario. En la **Figu-**

ra 30. se puede encontrar información detallada sobre lo descrito.

Figura 30

Ventana auxiliar para asignar un nuevo controlador



En esta figura se puede observar que el usuario tiene la capacidad de asignar otro controlador al robot, así como también utilizar controladores desarrollados por el mismo usuario.

Tanto el archivo “.wbt” como los controladores son elementos fundamentales que el lector de este trabajo o el usuario debe tener muy claros, ya que se utilizarán frecuentemente.

2.4 Configuración en el Lenguaje de Programación Python

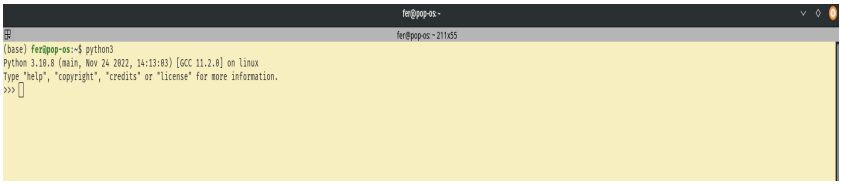
En esta sección, se presentarán al lector los pasos necesarios para configurar el entorno de programación dentro de Webots

y así utilizar Python como lenguaje predefinido sin llegar a tener conflictos relacionados con la versión de Python. A continuación, se presentan los pasos necesarios para realizar una correcta configuración:

- Como paso inicial, es necesario verificar que el ordenador que se va a utilizar disponga de Python 3 como lenguaje de programación. Para hacerlo, se debe ejecutar el siguiente comando en el terminal: "python3" o simplemente "python" en algunos casos. En la **Figura 31** consta la representación de este paso.

Figura 31

Accedo al lenguaje 'Python' a través de la terminal



A continuación, se presentan los comandos utilizados:

- Python3
- Este comando nos coloca en un entorno de programación de Python, donde podemos realizar una pequeña prueba de suma de variables para verificar que Python se esté ejecutando correctamente. En la **Figura 32** se muestra una posible representación de lo descrito.

Figura 32

Operaciones matemáticas dentro de "Python"

A continuación, se presentan los comandos utilizados:

- Python3
- Es importante mencionar que la versión de Python debe ser la "3" en adelante; este libro no aborda el uso de Python 2. Una vez cumplidos estos requisitos, se procede a verificar la ubicación donde está instalado Python. Para ello, se ejecuta el siguiente comando dentro del entorno de programación: 'import sys'. Esto incluye la librería de "sys", la cual proporciona acceso a algunas variables utilizadas o mantenidas por el intérprete. Posteriormente, se debe ejecutar el comando 'print(sys.executable)', lo cual dará la ubicación del intérprete de Python y permitirá conocer exactamente dónde fue instalado. En este caso, el resultado es "/home/fer/miniconda3/bin/python3". En la **Figura 33** se presentan los resultados de estos comandos.

Figura 33

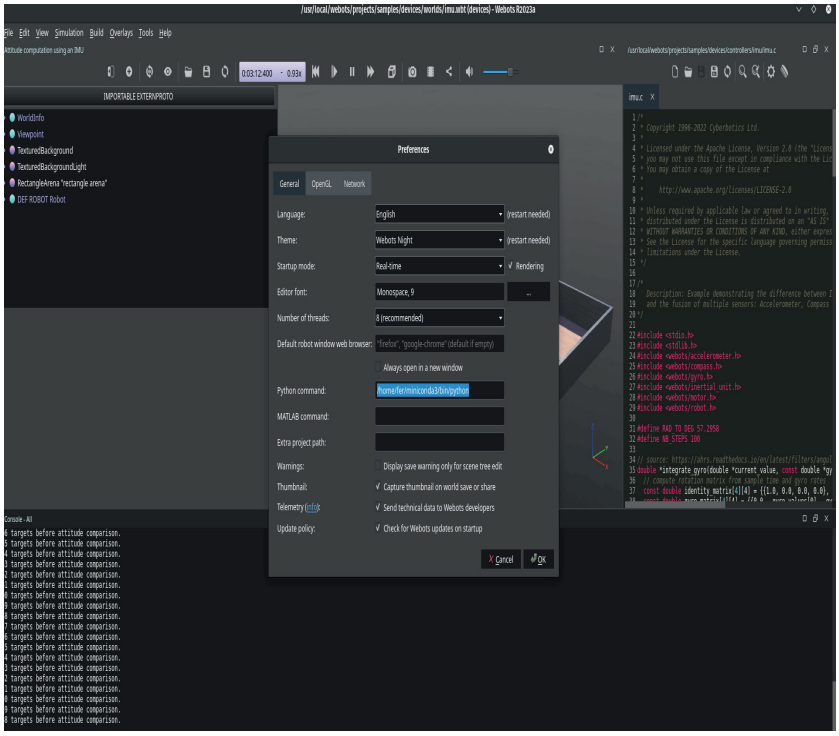
Ubicación del directorio de instalación de Python en el sistema operativo

Una vez obtenida la dirección que se muestra en esta figura, se puede configurar el lenguaje de programación en Webots. Para ello, nos ubicamos en la pestaña de 'Tools' y luego continuamos con la pestaña de 'Preferences'.

- Luego de encontrarse en esta ventana, se procede a copiar la dirección dentro del Python command y aparece la información que consta en la siguiente **Figura 34**. Esta acción se refiere al proceso de especificar la ubicación del intérprete de Python dentro del entorno de desarrollo de Webots.

Figura 34

Configuración de la ubicación de 'Python' dentro de Webots



Posteriormente, se reinicia el ambiente de simulación, donde el intérprete usado por el simulador será el que se especificó por el usuario.

Una vez finalizados estos pasos, Python quedará configurado correctamente dentro de Webots. De esta manera, cada vez que se genere un controlador basado en Python, se ejecutará con la versión correcta, evitando problemas de versiones.

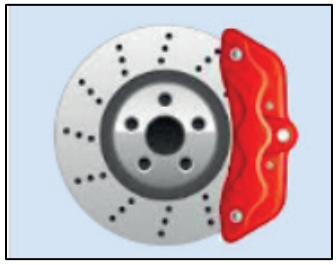
2.5 Actuadores

El simulador robótico Webots dispone de una amplia variedad de actuadores robóticos. Es importante destacar que algunos actuadores pueden llegar a utilizar subactuadores en su estructura. A continuación, se mostrarán actuadores genéricos dentro del sistema de simulación:

- **Freno:** El simulador Webots permite simular sistemas de freno mecánicos, comunes en la mayoría de automóviles. El icono correspondiente se encuentra en la siguiente figura.

Figura 35

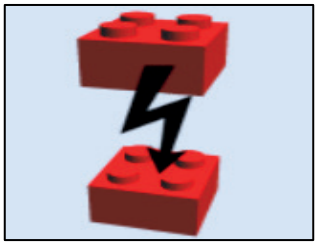
Freno simulado en Webots



- **Conector:** Permite simular objetos sujetos a deformación o ruptura de partes. En la **Figura 36** se muestra una posible representación de un conector.

Figura 36

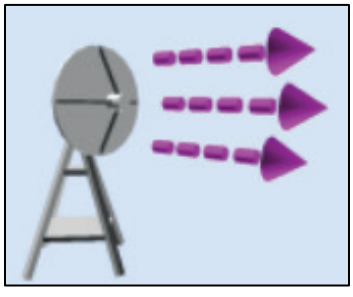
Conexión de elementos dentro de Webots.



- **Emisor:** Permite simular señales de radio, serial e infrarroja para el envío de información entre robots dentro del simulador. En la **Figura 37** se muestra una representación gráfica de lo descrito.

Figura 37

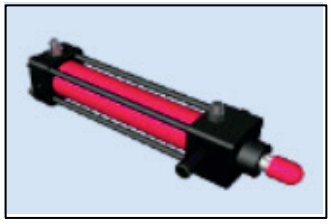
Objeto emisor dentro de Webots



- **Actuador Lineal:** Simula un actuador lineal, ampliamente utilizado en celdas de automatización en sectores industriales. En la **Figura 38** se muestra su representación.

Figura 38

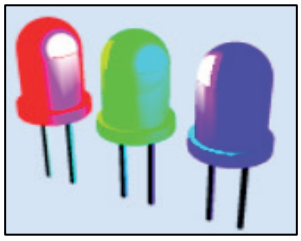
Actuador lineal dentro de Webots



- **Led:** Simula la luz emitida por un diodo led. En la **Figura 39** se muestra la representación de un led en el simulador.

Figura 39

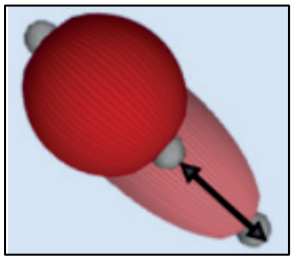
Leds dentro de Webots



- **Músculos:** Simula la forma y el comportamiento de músculos biológicos. En la **Figura 40** se muestra la representación de este actuador.

Figura 40

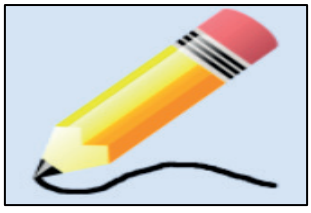
Actuador tipo musculo dentro de Webots



- **Lápiz:** Simula el comportamiento de un lápiz, que es capaz de dibujar sobre cualquier geometría dentro del simulador robótico. En la **Figura 41** se muestra su respectiva representación.

Figura 41

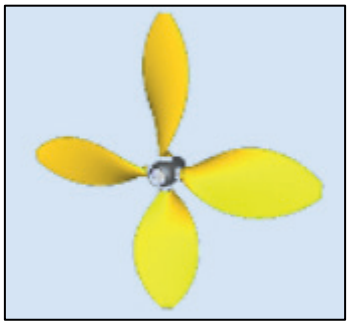
Posible representación de un actuador tipo lápiz dentro de Webots



- **Propelas o hélices:** Este modelo simula el comportamiento de hélices en vehículos aéreos o marinos. En la **Figura 42** se muestra su representación.

Figura 42

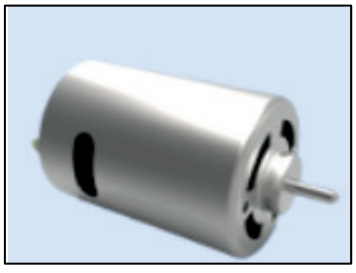
Hélices dentro de Webots



- **Actuador Rotacional:** Este componente replica el comportamiento de un actuador rotacional, como un motor de corriente continua. En la **Figura 43** se muestra una posible representación.

Figura 43

Motor o actuador dentro de Webots



- **Parlantes o bocinas:** Este componente simula el comportamiento de un dispositivo de reproducción de audio. En la **Figura 44** se muestra su representación.

Figura 44

Posible representación de bocinas dentro de Webots

2.6 Sensores

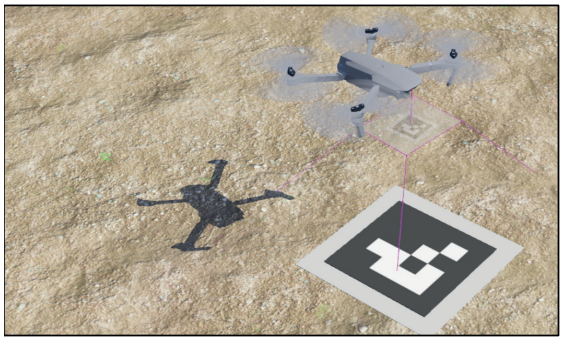
Dentro de Webots, es posible simular una amplia variedad de sensores estándar en el campo de la robótica. Así, se dispone de una serie de sensores genéricos que pueden combinarse para generar sistemas de sensores aún más complejos, como cámaras y sensores Lidar. A continuación, se presenta una breve explicación de los tipos de cámaras y Lidar disponibles en este simulador.

Cámaras

En Webots, es posible utilizar cámaras y modificar sus características, como la resolución de la imagen, el campo de visión y el ruido presente en la imagen digital. Además, el *software* ofrece diversos efectos que pueden agregarse a la imagen, como difuminado al movimiento, diferentes modelos de ruido, distorsión de la imagen, y finalmente, varios tipos de proyección, entre los que se incluyen la planar, cilíndrica y esférica. La **Figura 45** muestra el funcionamiento del sensor de cámara dentro de un ambiente de simulación.

Figura 45

Representación de un sensor tipo cámara en Webots

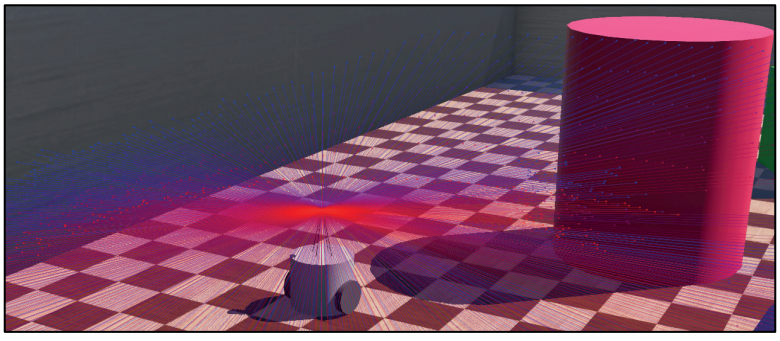


Lidar

Dentro del simulador Webots, es posible simular un amplio rango de sensores Lidar comerciales, como el Hokuyo URG-04LX, el Robotis LDS-01, el Velodyne Puck y muchos más. En la **Figura 46** se muestra el funcionamiento de un sensor Lidar dentro del ambiente de simulación.

Figura 46

Representación de un sensor tipo Lidar en Webots



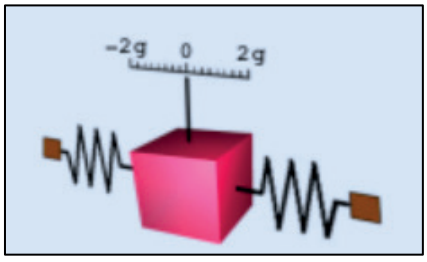
Sensores Genéricos

Webots presenta una amplia gama de sensores genéricos. A continuación, se detalla información relevante para el usuario:

- **Acelerómetro:** Este componente simula el comportamiento de un acelerómetro, que mide aceleraciones relativas. En la **Figura 47** se muestra una posible representación.

Figura 47

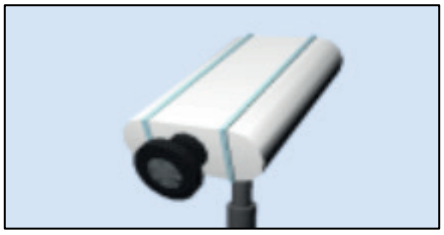
Acelerómetro dentro de Webots



- **Cámara:** El usuario puede simular el comportamiento de cámaras con información en RGB, escala de grises e incluso de lente tipo *fisheye.* En la **Figura 48** se muestra una posible representación de este dispositivo dentro del simulador.

Figura 48

Representación gráfica de una cámara en Webots



- **Compass:** Este componente permite simular el comportamiento de un sensor magnético, mediante el cual se mide la dirección relativa al norte. En la **Figura 49** se muestra su representación.

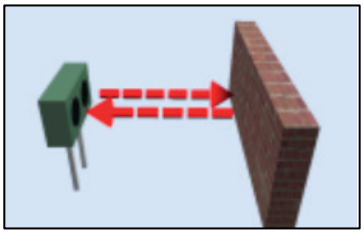
Figura 49

Sensor magnético dentro de Webots

- **Sensor de Distancia:** Este componente replica el comportamiento de un sensor de distancia basado en luz infrarroja, eco de sonar o rayo láser. En la **Figura 50** se muestra su posible representación.

Figura 50

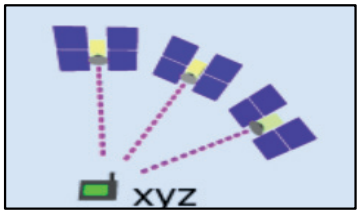
Sensores de distancia dentro de Webots



- **GPS:** Este componente permite simular el comportamiento de un sensor capaz de medir la posición absoluta dentro del simulador con respecto al sistema de referencia global. En la **Figura 51** se muestra una posible representación.

Figura 51

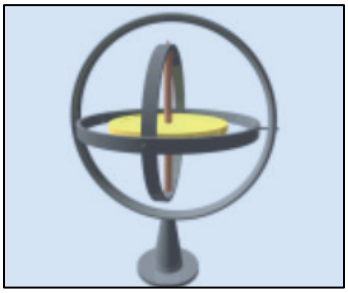
Representación del funcionamiento del GPS en Webots



- **Giroscopio:** Este componente simula un sensor de tipo giroscopio, mediante el cual es posible medir las velocidades angulares del objeto al que esté sujeto. En la **Figura 52** se muestra la representación de dicho sensor.

Figura 52

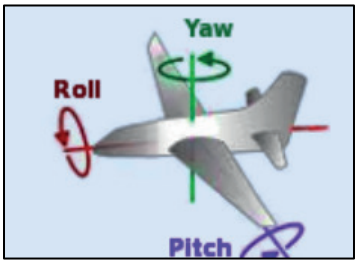
Ilustración de un giroscopio en Webots



- **Unidad Inercial:** Este componente replica el comportamiento de una unidad inercial con la cual se puede estimar el valor del desplazamiento angular de los objetos. En la **Figura 53** se muestra su representación gráfica.

Figura 53

Representación de las medidas otorgadas por la unidad inercial



- **Lidar:** Este componente simula el comportamiento de un sensor Lidar, permite al usuario realizar modificaciones como aumentar el número de líneas del sensor y ajustar el ángulo de resolución. En la **Figura 54** se muestra un sensor Lidar comercial, el SICK LMS 291 Lidar.

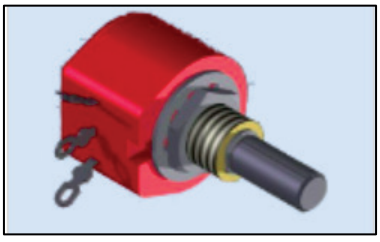
Figura 54

Ilustración de un sensor Lidar en Webots

- **Sensor de posición angular:** Este tipo de sensor permite medir el desplazamiento angular, como un sensor tipo encoder o un potenciómetro. En la **Figura 55** se muestra su representación gráfica.

Figura 55

Sensor de desplazamiento angular



- **Cámara con profundidad:** Este componente simula el comportamiento de una cámara con información de profundidad. Un modelo muy popular es el Microsoft Kinect. En la **Figura 56** se muestra su representación.

Figura 56

Sensor tipo cámara con información de profundidad

- **Sensor de fuerza:** Este componente permite al usuario simular sensores de fuerza, los cuales pueden colocarse en el extremo de un manipulador robótico para interactuar con el entorno.

Con la breve información de cada uno de los sensores, el usuario logrará un mejor flujo de trabajo dentro del simulador Webots. En las siguientes secciones, se tratarán temas como la simulación de sistemas móviles y manipuladores, donde la información presentada en esta sección es de gran importancia.

3

Primeros pasos dentro del Simulador

En esta sección, se presentan una serie de tutoriales para que los usuarios comprendan los conceptos fundamentales en Webots. Esto les posibilitará la creación de sus propias simulaciones, familiarizarse con la interfaz, crear objetos simples dentro del entorno, controlar los robots y comprender de manera más profunda las mecánicas de la física dentro de Webots.

Adicionalmente, en este apartado se aborda la modelación de robots y su entorno de simulación dentro de Webots, así como en la programación de sus respectivos controladores. El usuario no necesitará conocimientos adicionales durante esta sección; no obstante, contar con un poco de conocimiento en robótica, matemáticas y modelación 3D puede resultar de gran ayuda.

3.1 Primera Simulación en Webots

El objetivo de este tutorial es familiarizarse con la interfaz de usuario y algunos conceptos básicos presentes dentro del simulador. Para ello, se dispone de un entorno de simulación muy simple con elementos como un campo de trabajo, algunas cajas, un robot e-puck y, finalmente, un controlador que permitirá al robot moverse de forma autónoma.

Iniciar Webots

Para iniciar el simulador robótico, se debe ingresar el siguiente comando en el terminal: 'webots'. El simulador se ejecutará inmediatamente, lo que permite al usuario utilizar las distintas funcionalidades.

Crear un Nuevo "world"

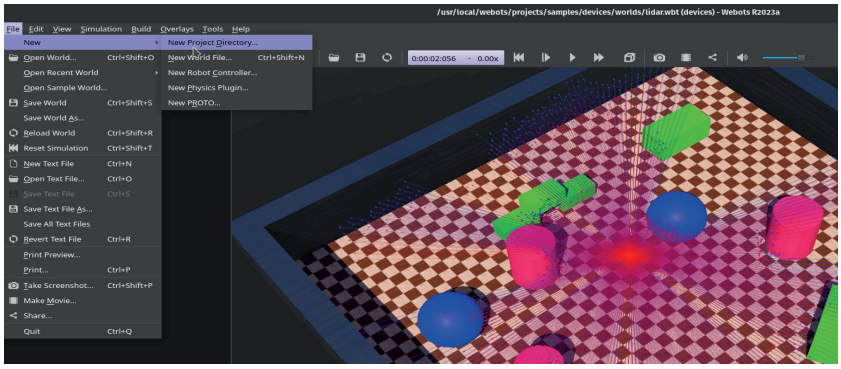
Un archivo **'world'** contiene toda la información del entorno de simulación, como su representación, la forma de interacción entre los elementos del entorno, los colores, las masas de los objetos, entre otros. Los diferentes objetos son llamados **nodos** y están organizados en el á**rbol de escena.** Un archivo **'world**' se almacena con la extensión .wbt.

Para crear un nuevo proyecto de simulación, es necesario seguir los siguientes pasos:

- Primero, es preciso seleccionar la siguiente opción dentro del menú de usuario **File/New/New Project Directory**, la correcta representación de este paso se presenta en la **Figura 57**. Estos directorios son utilizados para organizar y almacenar archivos relacionados con proyectos de simulación, lo que ayuda a mantener la estructura de trabajo ordenada y facilita la gestión de archivos y recursos en el entorno de Webots.

Figura 57

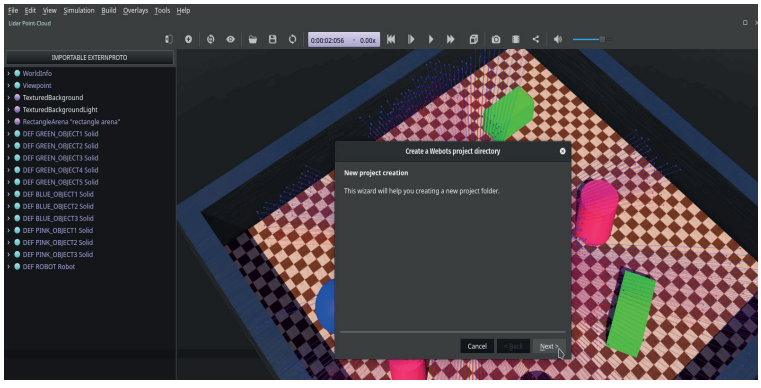
Proceso de creación de un nuevo directorio de trabajo en Webots



- Luego, se presenta una nueva ventana para crear un nuevo directorio de trabajo; para ello, se debe seleccionar la opción “Next”. En la **Figura 58** consta la representación de este paso.

Figura 58

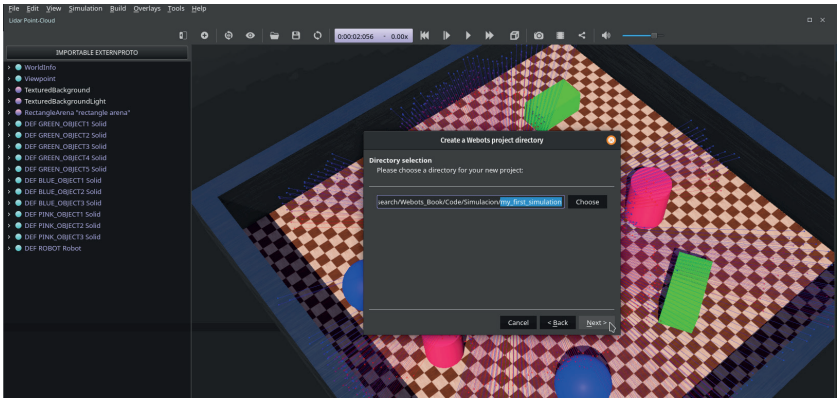
Ventana de asistencia para la creación de un nuevo directorio de trabajo



- Después, se debe dirigir al directorio de trabajo deseado y crear un directorio llamado 'my_first_simulation'. Luego, hay que seleccionar la opción 'Next'. En la **Figura 59** existe más información sobre este paso.

Figura 59

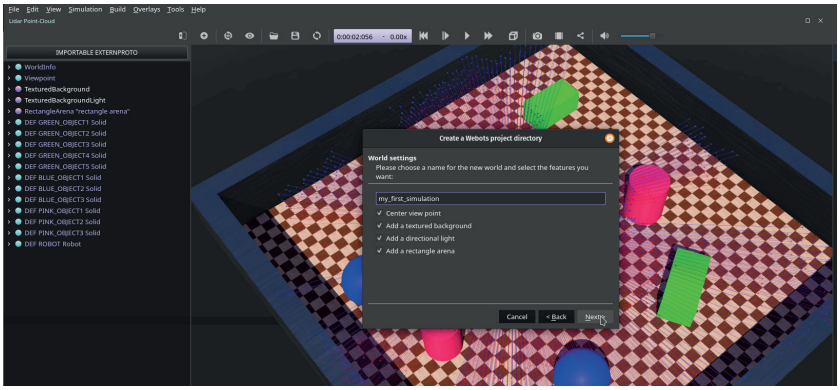
Creación del directorio llamado 'my_first_simulation'



- A continuación, aparece una ventana donde el usuario debe ingresar el nombre del archivo 'world'; escriba 'my_first_simulation' y, finalmente, seleccione todas las características, incluida un área de trabajo rectangular. En la **Figura 60** se muestra más información acerca de lo descrito.

Figura 60

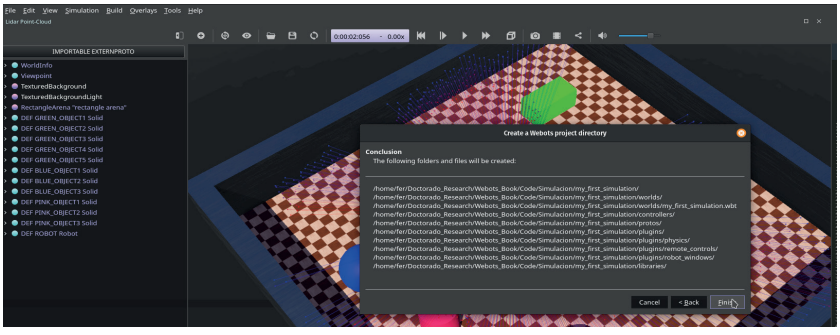
Ventana de asistencia para incluir archivos del tipo "world" y el área de trabajo rectangular



- Finalmente, Webots muestra una ventana con las carpetas y archivos creados, y solo resta seleccionar la opción 'Finish'. En la **Figura 61** consta la información detallada sobre este paso.

Figura 61

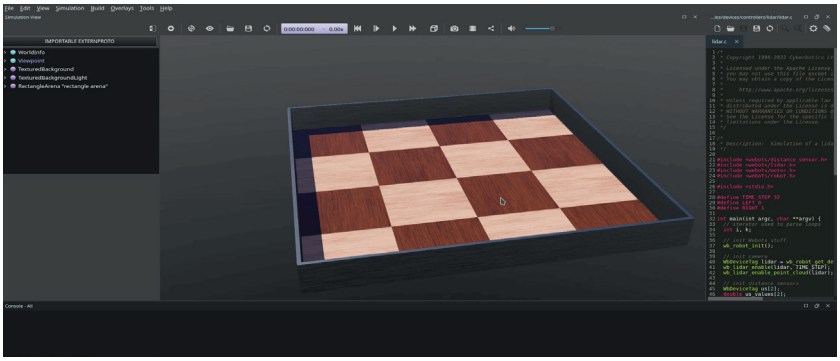
Ventana que proporciona una vista detallada de los elementos que han sido creados en el entorno de trabajo



- ¡Felicidades, has creado tu propio ambiente de simulación! En la **Figura 62** se presenta el resultado de todos estos pasos.

Figura 62

Resultado de la creación de un directorio de trabajo en Webots



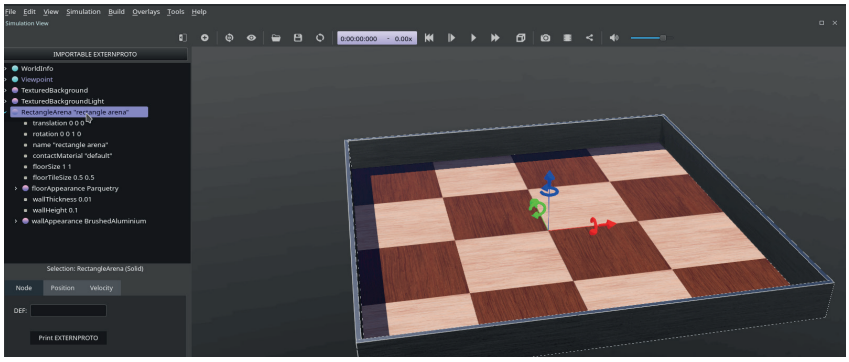
Tras la culminación de los pasos previamente mencionados, el usuario visualizará un área de trabajo rectangular, donde será posible moverse utilizando las funcionalidades de desplazamiento mencionadas anteriormente en esta sección.

Dentro del árbol de escena se pueden modificar los nodos y se presentan los siguientes elementos: WorldInfo, Viewpoint, TexturedBackground, TexturedBackgroundLight y RectangleArena. Cada elemento dentro del árbol de escena tiene propiedades que se pueden modificar. Por ejemplo, para cambiar las dimensiones del área de trabajo (RectangleArena), se deben seguir los siguientes pasos:

- Hacer doble clic sobre 'RectangleArena' en el árbol de escena, lo cual mostrará todos los campos presentes en este nodo. En la **Figura 63** consta más información sobre este paso.

Figura 63

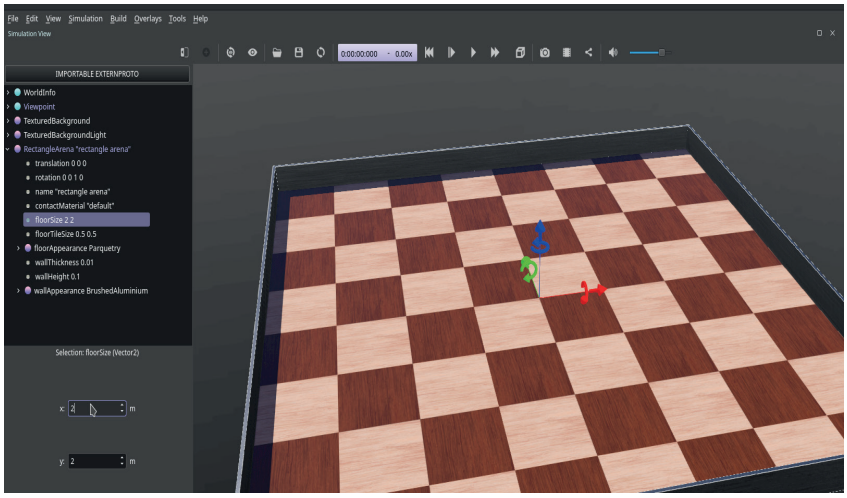
Modificación de las propiedades del 'RectangleArena' en el árbol de escena



- Seleccionar el campo 'floorSize' y modificar los valores a 2 en cada eje. En la **Figura 64** se muestra la representación de este paso.

Figura 64

Asignación de nuevas dimensiones para "RectangleArena"



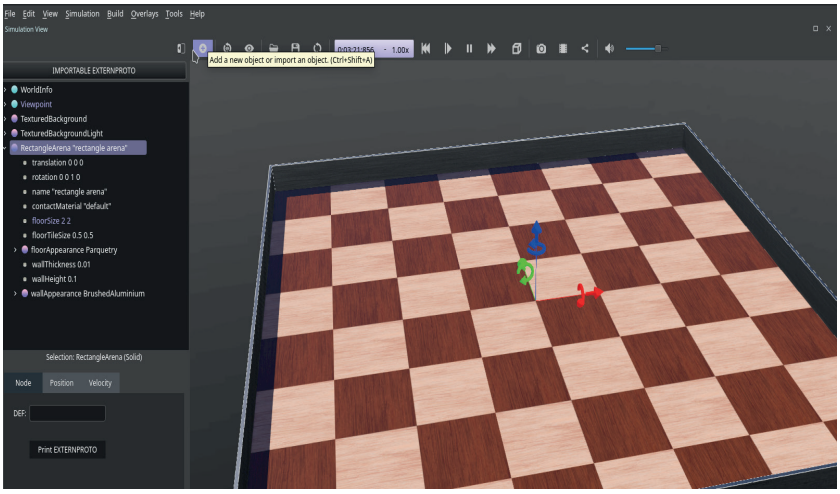
Al aumentar su dimensión, el usuario puede visualizar cómo el campo de trabajo se modifica.

Además, es posible agregar objetos al ambiente de simulación para incrementar su realismo. En los siguientes pasos se mostrará cómo agregar objetos adicionales al ambiente de simulación.

- Hacer doble clic en 'RectangleArena' y luego presionar el botón 'Agregar', ubicado en la parte superior del árbol de escena. La ejecución de este paso se muestra en la **Figura 65**. Este proceso implica agregar elementos adicionales a la escena tridimensional en Webots.

Figura 65

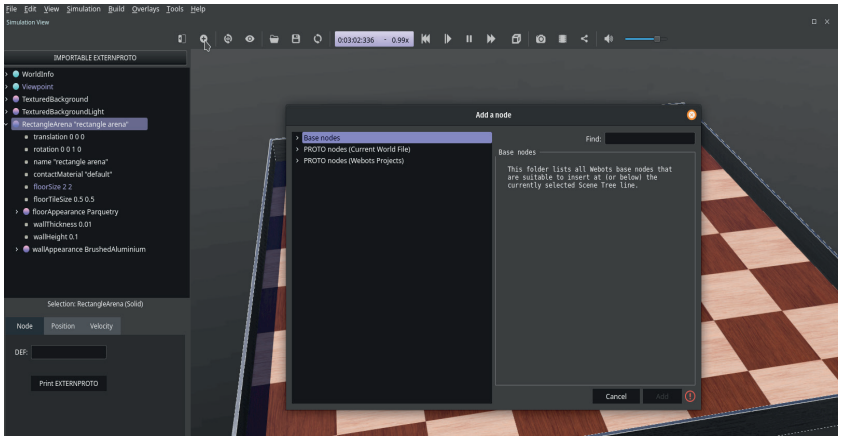
Incorporación de nuevos objetos a la escena 3D



- Al hacer clic, se desplegará una nueva ventana en la que el usuario tendrá acceso a todos los objetos y robots dentro del simulador Webots. En la **Figura 66** se puede visualizar esta paso.

Figura 66

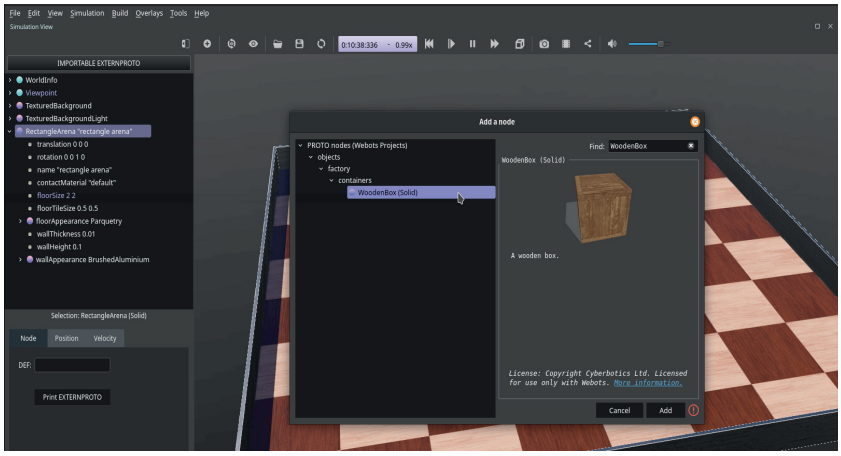
Ventana auxiliar que proporciona una interfaz para seleccionar el objeto que se desea agregar a la escena en Webots



- Dentro del pequeño buscador, se debe introducir 'WoodenBox' y así el sistema encontrará automáticamente el objeto que deseamos agregar al ambiente de simulación. Este paso permite elegir el objeto denominado 'WoodenBox' dentro de la ventana auxiliar en Webots. En la **Figura 67** se indican más detalles sobre lo descrito.

Figura 67

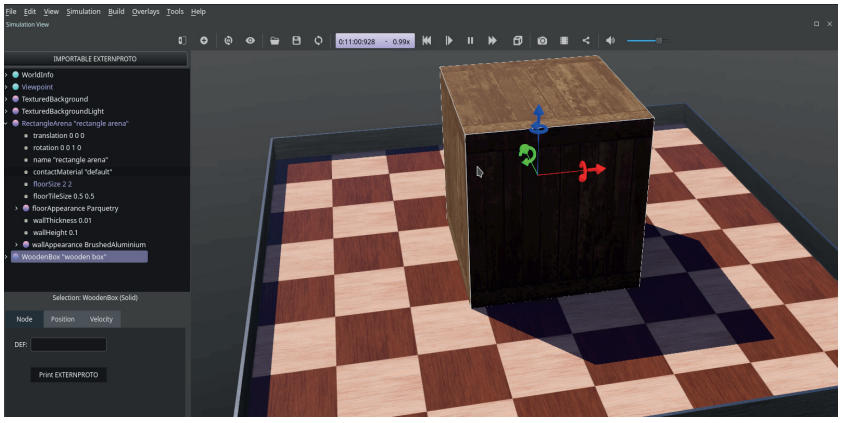
Selección del elemento 'WoodenBox' en la ventana auxiliar



- Finalmente, es preciso seleccionar la opción 'Add' para que el objeto se agregue al ambiente de simulación. En la **Figura 68** se muestra este paso.

Figura 68

Visualización del elemento 'WoodenBox' en el espacio 3D



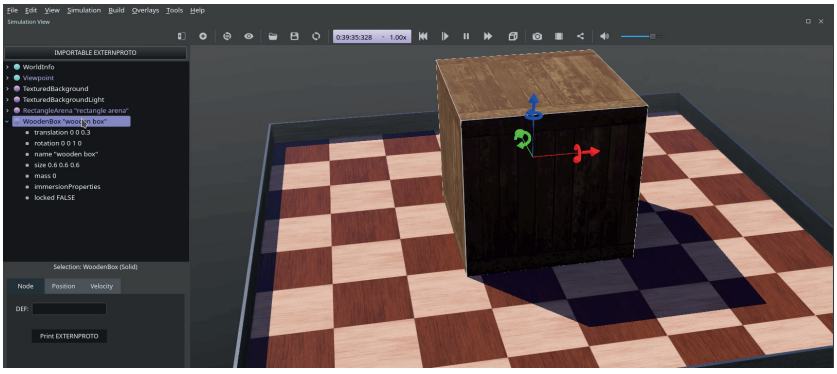
Una vez agregado el objeto deseado, el usuario puede notar que las dimensiones no son las correctas para nuestra área de trabajo. Por lo tanto, a continuación, se definen una serie de pasos que nos permiten cambiar las dimensiones del objeto, duplicarlos, crear más objetos y trasladarlos a posiciones deseadas por el usuario.

Para realizar ajustes en las dimensiones del objeto, es preciso seguir los siguientes pasos:

- Hacer doble clic sobre el nodo 'WoodenBox' dentro del árbol de escena; esto desplegará los campos que este objeto dispone. En la **Figura 69** se muestra lo descrito.

Figura 69

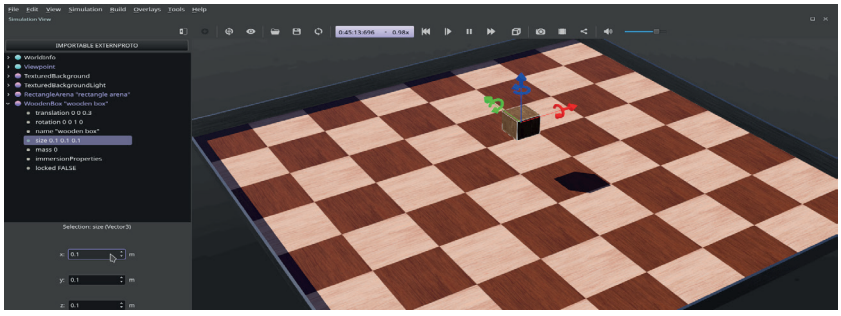
Propiedades del elemento 'WoodenBox' en el árbol de escena



- Dentro del campo 'Size' se pueden cambiar las dimensiones del objeto a 0.1 en cada uno de los ejes. En la **Figura 70** se muestra este paso.

Figura 70

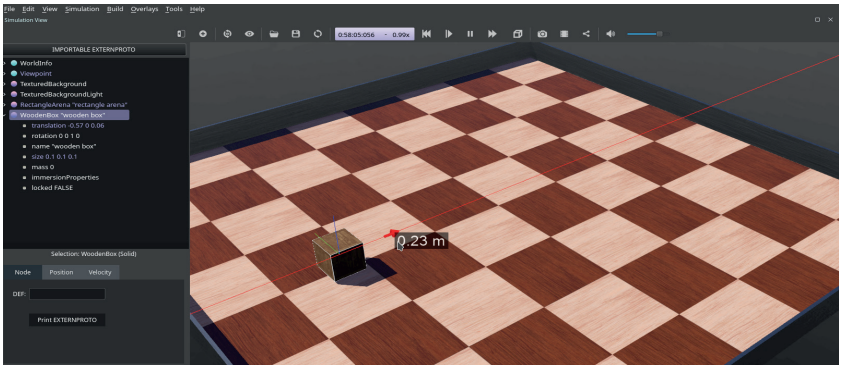
Asignación de nuevas dimensiones al elemento



A continuación, hay que desplazar el objeto a la posición deseada por el usuario utilizando el sistema de referencia generado en el objeto. Para ello, se siguen los pasos mencionados en la sección anterior. En la **Figura 71** se muestra una representación de este procedimiento.

Figura 71

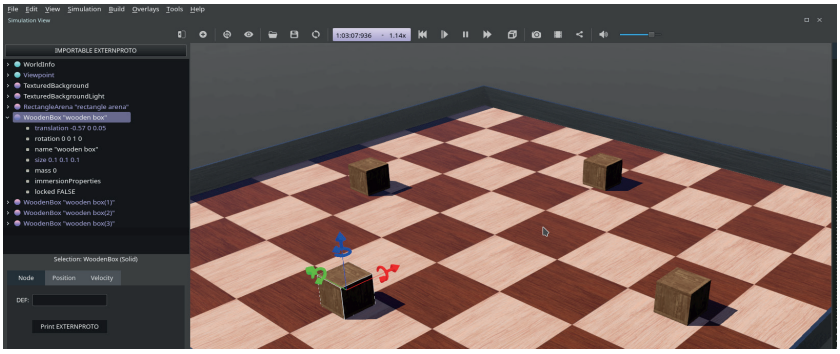
Representación de posibles desplazamientos del elemento 'WoodenBox' dentro del entorno de simulación



Finalmente, se va a copiar y pegar dicho objeto simplemente presionando las teclas 'Ctrl' + 'c' para copiar y 'Ctrl' + 'v' para pegar el objeto (ver **Figura 72**). Con esto, el usuario es capaz de replicar el objeto previamente generado, agilizando el flujo de trabajo dentro del simulador.

Figura 72

Ilustración de varios elementos 'WoodenBox' dentro del ambiente de simulación

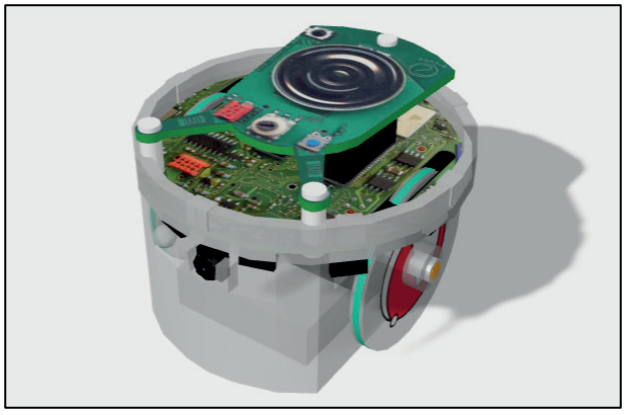


Robot e-puck dentro de Webots

A continuación, se incorpora un robot móvil al entorno de simulación. Este robot, conocido como el e-puck, es de tipo diferencial y cuenta con 10 leds, así como varios sensores, incluyendo 8 sensores de distancia y una cámara. En la **Figura 73** se muestra la apariencia de este robot.

Figura 73

Renderización del robot E-puck



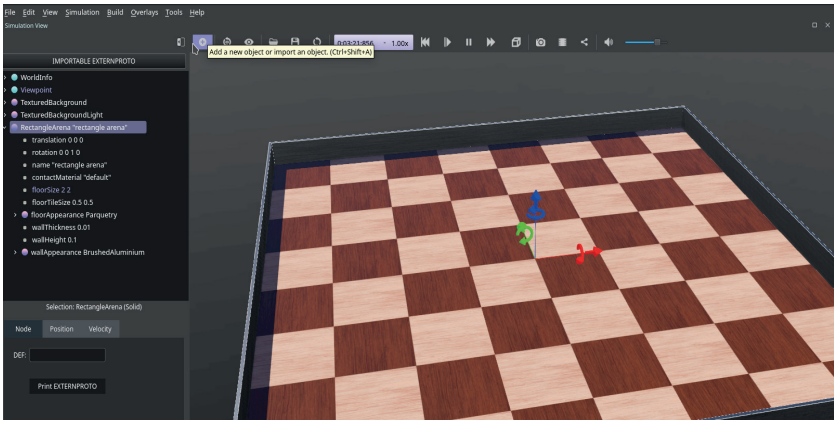
Para incorporar el robot e-puck al entorno de simulación, es necesario verificar que la simulación esté pausada y que el tiempo virtual se encuentre en 0. Al modificar un entorno de simulación con la intención de guardar los cambios, es crucial pausar la simulación primero y restablecerla a su estado inicial. Si no se realiza de esta manera, pueden acumularse errores. Por lo tanto, cualquier modificación dentro del entorno de simulación debe seguir el siguiente orden: (1) pausar, (2) restablecer, (3) modificar y, finalmente, (4) guardar los cambios realizados.

A continuación, se presentan los pasos necesarios para agregar el robot mencionado:

- Hacer doble clic en 'RectangleArena' y luego presionar el botón 'Agregar', ubicado en la parte superior del árbol de escena. En la **Figura 74** se muestra la ejecución de este paso.

Figura 74

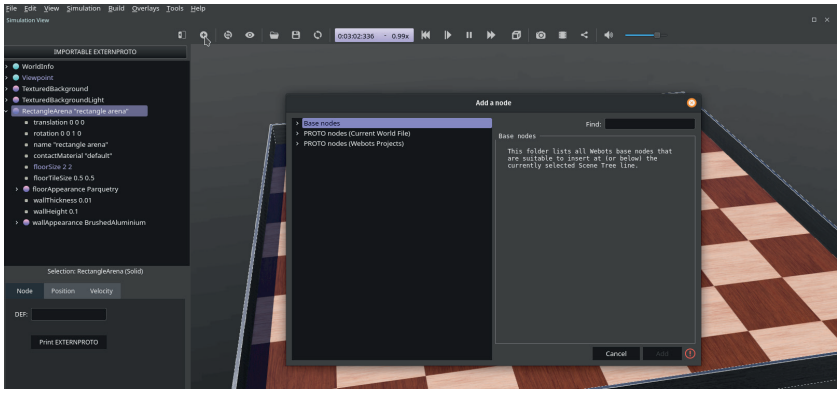
Nuevos elementos agregados al entorno Webots



- Al hacer clic, se desplegará una nueva ventana en la que el usuario tendrá acceso a todos los objetos y robots dentro del simulador Webots. En la **Figura 75** se puede visualizar lo descrito.

Figura 75

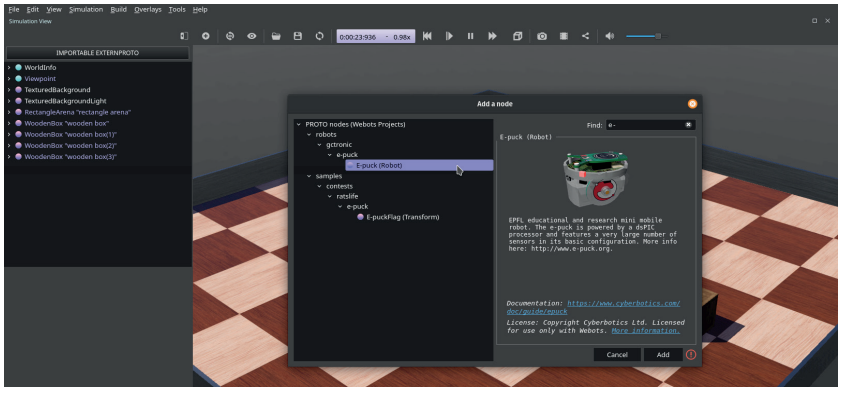
Ventana auxiliar con los elementos disponibles dentro de Webots



- Dentro del pequeño buscador, se debe introducir 'e-puck' y el sistema encontrará automáticamente el objeto que se desea agregar al ambiente de simulación. En la **Figura 76** consta más información sobre este paso.

Figura 76

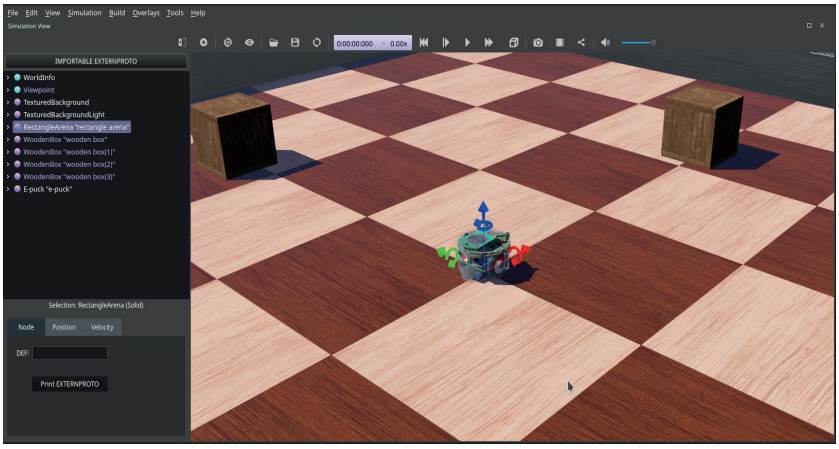
Selección del robot e-puck dentro de la ventana auxiliar



- El robot aparecerá en el medio del área de trabajo el usuario podrá moverlo y girarlo de igual forma como se hizo con el objeto 'WoodenBox'. En la **Figura 77** se muestra la representación respectiva.

Figura 77

E-puck robot dentro del ambiente 3D de Webots



El usuario podrá notar que el robot se mueve de manera autónoma dentro del espacio de trabajo. Además, cada uno de los leds parpadea y el robot es capaz de evitar obstáculos presentes en el ambiente, como cajas y paredes. Cabe recordar que es posible aplicar fuerzas y torques al robot mediante los comandos por teclado; este procedimiento fue mencionado con anterioridad.

Crear el Controlador

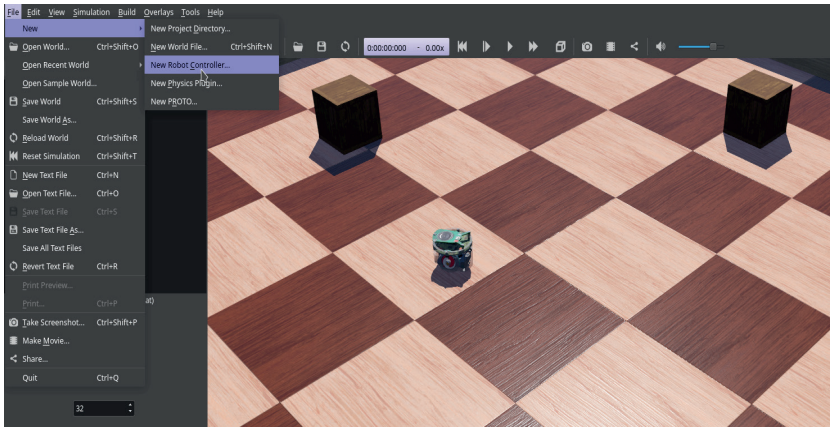
Una vez incorporado el sistema robótico en el sistema de simulación, el usuario deberá crear el controlador para dicho robot. Como se mencionó previamente, los controladores pueden programarse en diversos lenguajes, como C, C + +, Matlab, Python y Java. Por razones prácticas, se va a utilizar Python para el desarrollo de los controladores.

A continuación, se presentan los pasos para crear un nuevo controlador utilizando el lenguaje de programación mencionado:

- Hacer clic en la siguiente dirección o seleccionar **File / New / New Robot Controller**. En la **Figura 78** se muestra su representac**ión.**

Figura 78

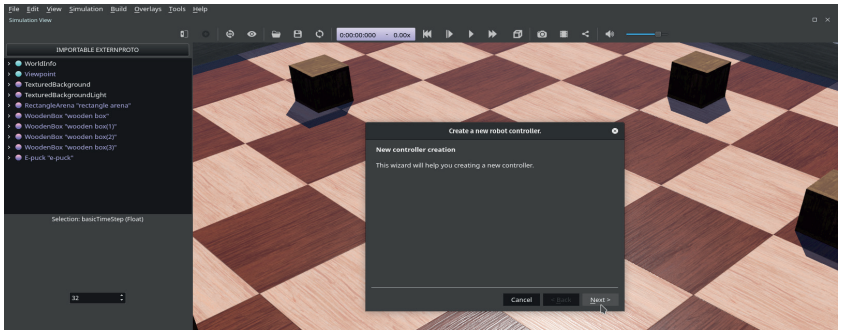
Creación de un nuevo controlador



- Luego, se muestra una ventana que asistirá al usuario en la creación de un nuevo controlador para el sistema robótico. En la **Figura 79** consta su representación.

Figura 79

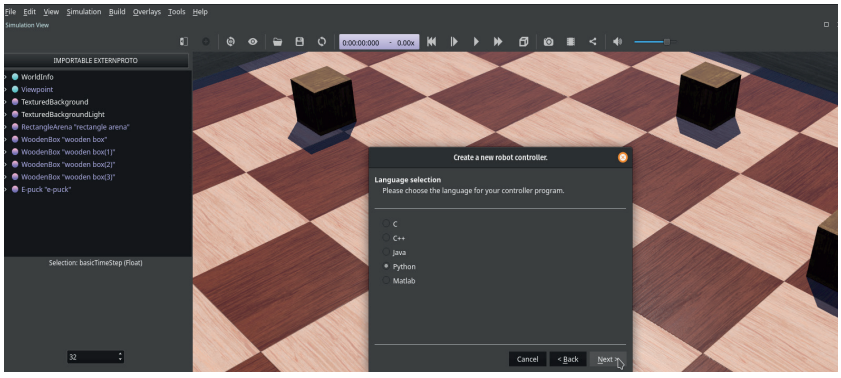
Ventana de asistencia para la creación del controlador



- El usuario deberá seleccionar el lenguaje de programación que desea utilizar, por ejemplo, Python. En la **Figura 80** se indica este paso.

Figura 80

Selección del lenguaje de programación para el controlador

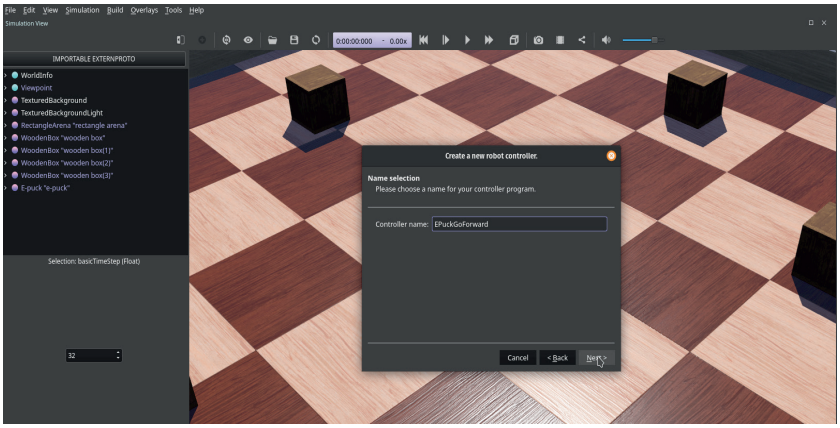


- A continuación, se asigna un nombre al controlador. En este ejemplo práctico, el controlador se llamará

'EPuckGoForward'. En la **Figura 81** se muestra este paso.

Figura 81

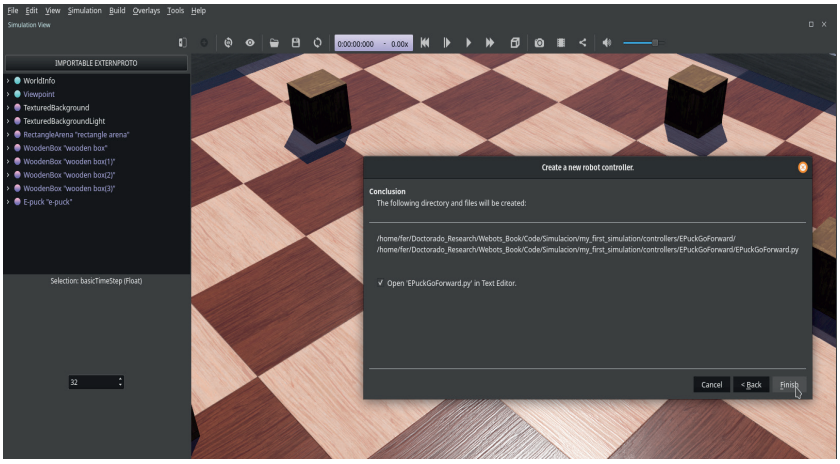
Asignación del nombre para el controlador



- Finalmente, se muestran los archivos creados automáticamente por el asistente. Al dar clic en 'Finish', el controlador se creará automáticamente. En la **Figura 82** se muestra lo descrito.

Figura 82

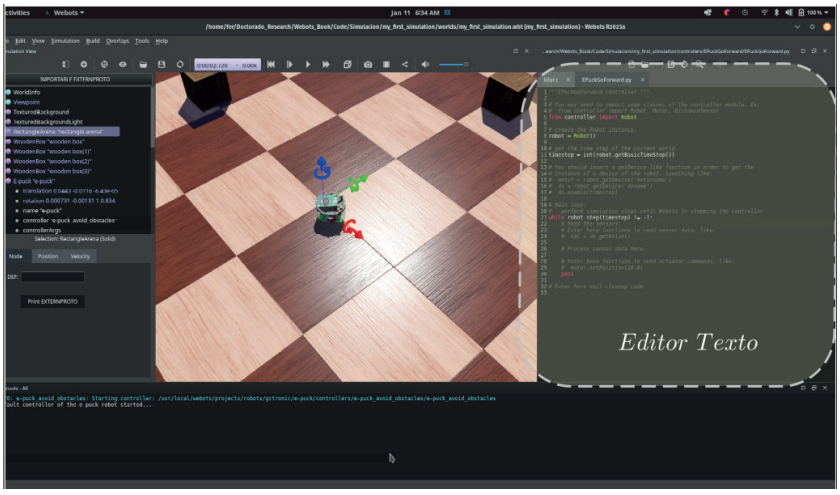
Visualización de la creación de los elementos necesarios para el controlador



- A continuación, se presenta la ubicación del archivo creado en la interfaz de usuario. Este archivo cuenta con una plantilla inicial que el usuario puede utilizar para programar distintos robots. Asimismo, el usuario tiene la opción de modificar este archivo para llevar a cabo tareas específicas. En la **Figura 83** se visualiza dicho archivo de texto dentro de la interfaz.

Figura 83

Archivo controlador

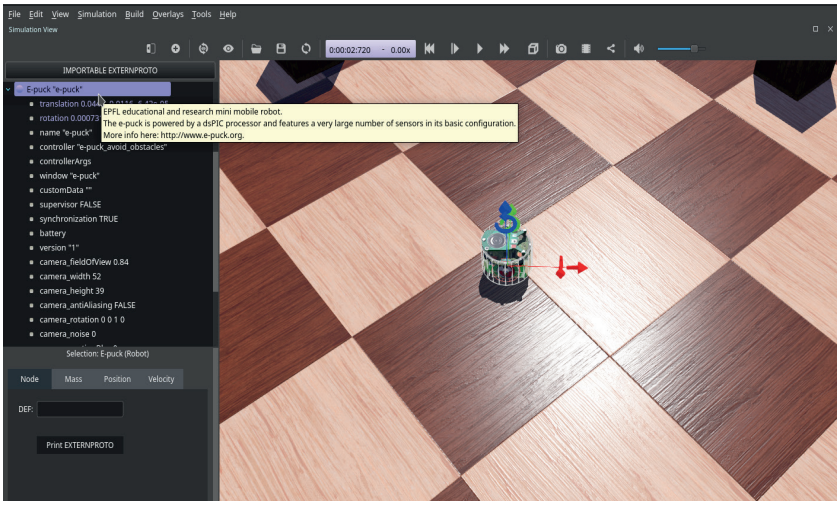


Una vez creado, el controlador debe asociase al robot e-puck. Para lograrlo, se seguirán los siguientes pasos, los cuales permitirán asignar el nuevo controlador.

- Hacer doble clic sobre 'e-puck' en el árbol de escena, lo cual mostrará todos los campos presentes en este nodo. En la **Figura 84** se indica este paso.

Figura 84

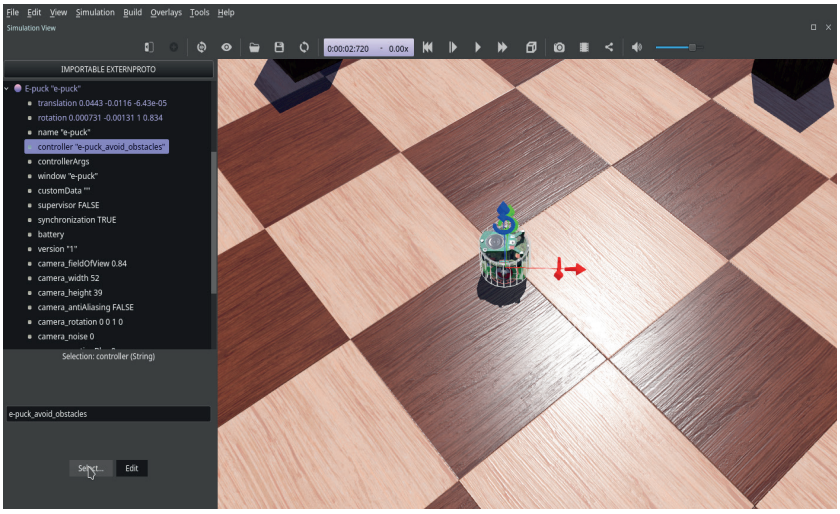
Propiedades del robot e-puck dentro del árbol de escena



- Dentro de los campos mostrados, se encuentra el campo 'controller' que por defecto tiene asignado el controlador 'e-puck_avoid_obstacles'. El usuario debe seleccionar la opción 'select' para asignar otro controlador al robot. En la **Figura 85** consta más información sobre lo descrito.

Figura 85

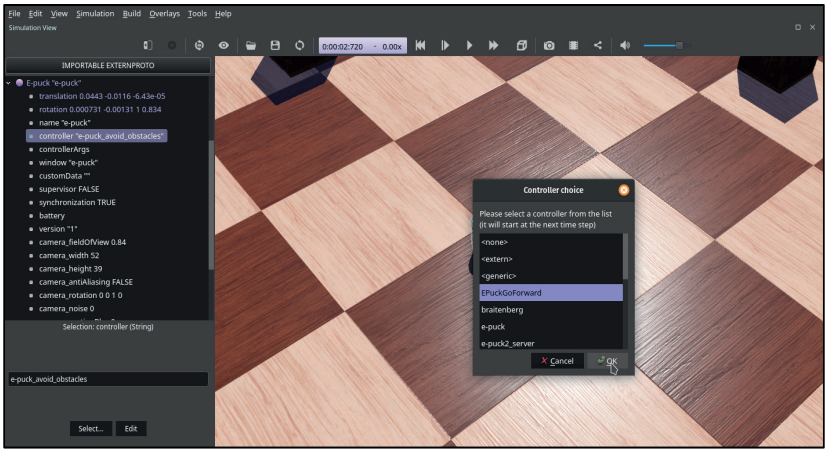
Propiedades del controlador asociadas al robot e-puck



- Una vez seleccionada la opción mencionada, se abre una nueva ventana que indica todos los controladores disponibles para este robot, además del controlador creado anteriormente. El usuario debe elegir el controlador 'EPuckGoForward' y hacer clic en 'Ok'. En la **Figura 86** se ilustra este procedimiento.

Figura 86

Asignación del nuevo controlador al robot e-puck



- Para que los cambios se reflejen, es pertinente guardar las modificaciones y luego reiniciar el entorno de simulación. Con esto, el robot no deberá realizar ningún movimiento, demostrando así que se ha cambiado su controlador.

Una vez completados todos los pasos, se pueden realizar modificaciones dentro del controlador creado. A continuación, se describirán brevemente las funciones que serán introducidas en el nuevo controlador.

- **get_motor:** Retorna el objeto motor asociado al nombre proporcionado.
- **set_motor:** Configura el motor para permitir el control de velocidad.
- **set_motors_velocity:** Establece la velocidad angular de los motores del robot con base en un vector de velocidades.

- **main:** Es la función principal, donde se ejecuta la simulación del robot. Define el paso de tiempo y el tiempo total de simulación. Además, controla los motores izquierdo y derecho para que el robot se mueva durante un tiempo especificado.

En general, el código genera señales de control 'u', que representan las velocidades angulares para cada motor. Estas velocidades se ejecutan durante un tiempo específico antes de completar la simulación.

Para obtener más información sobre el código desarrollado, visita el enlace https://bit.ly/45PSRJb, donde se presenta con mayor detalle la implementación de dicho controlador. Finalmente, la simulación completa está disponible en https://bit.ly/3zwpFLh. Accede a la carpeta 'Simulacion', luego a 'my_first_simulation'. Recuerda que puedes descargar todo el repositorio y acceder fácilmente desde el directorio de Webots al abrir un nuevo proyecto.

3.2 Modificación del Ambiente de Simulación en Webots

El objetivo de este tutorial es enseñar al usuario cómo crear diferentes objetos en el ambiente de simulación. En este contexto, se introduce un balón que tiene la capacidad de interactuar con la simulación gracias a la configuración de las físicas de dicho objeto.

Así, se utiliza el ambiente de simulación previamente desarrollado; sin embargo, este será guardado con otro nombre designado, "obstacles.wbt". Este procedimiento se puede llevar a cabo utilizando los menús de usuario antes definidos.

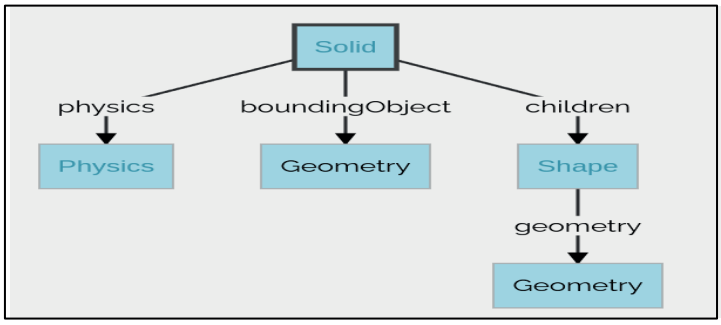
Previo a iniciar el desarrollo de este tutorial, es importante introducir uno de los nodos más significativos dentro de Webots. El nodo Sólido o "Solid Node" se utiliza para representar cuerpos rígidos donde las deformaciones no son consideradas, por ejemplo, una tabla o una rueda de un robot móvil. Dichos elementos deben conservar sus propiedades a pesar de estar sujetos a fuerzas que actúen sobre ellos.

En general, el motor de físicas de Webots está diseñado para utilizar solo cuerpos rígidos. Por lo tanto, para representar un cuerpo rígido en Webots, el usuario debe usar el nodo sólido "Solid Node". Dentro de este, se encuentran diferentes subnodos que corresponden a las características del cuerpo rígido. Es importante señalar que la representación gráfica de un "Solid Node" está definida por "Shape" y sus subelementos. Las mallas de colisión se pueden definir mediante el campo "boundingObject". Finalmente, el campo de físicas o "physics" es donde se definen las características del objeto, como masa e inercias.

Sin embargo, todos estos elementos pueden estar o no definidos. Para definir correctamente las físicas, se debe incluir el campo "boundingObject". La **Figura 87** presenta los elementos que conforman un nodo "Solid Node", junto con cada uno de sus componentes.

Figura 87

Mapa jerárquico de los elementos sujetos a físicas tipo "Solid"



Es importante mencionar que la geometría del "Solid Node" puede ser de cualquier forma, aunque con el fin de acelerar la simulación, se recomienda utilizar esferas, cajas y cilindros. Una vez que se han establecido estos conceptos, es posible agregar objetos a la simulación.

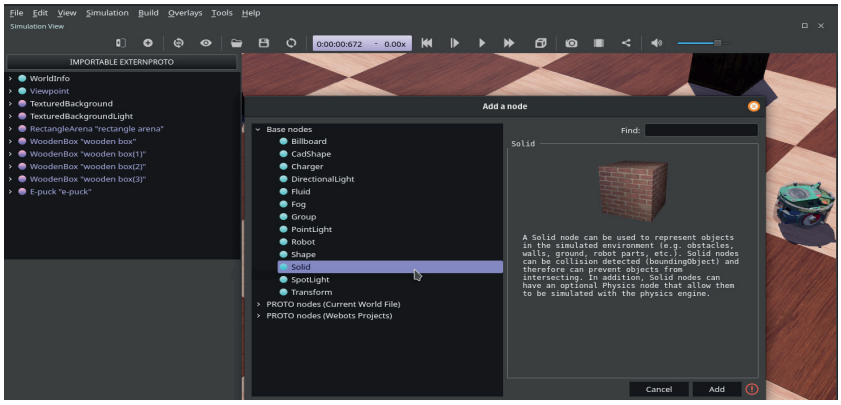
Crear y agregar un balón

En esta sección, se va a agregar un balón al ambiente de simulación. Este elemento se modela como un cuerpo rígido utilizando una esfera para definir su geometría. A continuación, se presentan los pasos a seguir:

- Dentro del árbol de escena, hacer clic en el botón "Add" para abrir una nueva ventana con varios elementos disponibles. El usuario debe seleccionar "Base nodes", dentro del cual encontrará la opción "Solid". En la **Figura 88** se muestra la representación correcta de este paso.

Figura 88

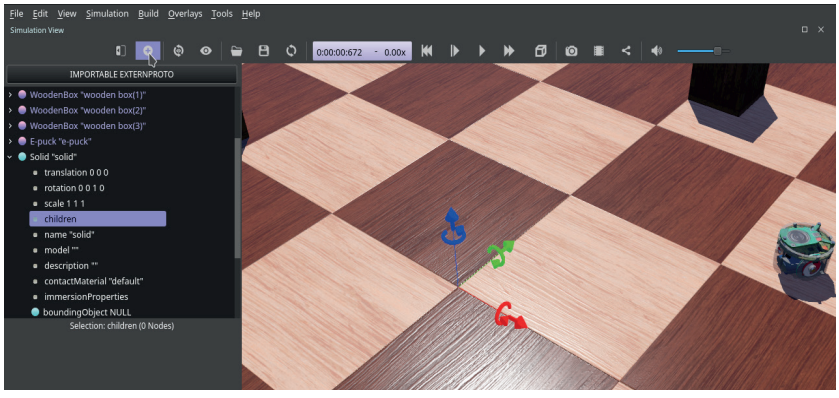
Ventana auxiliar para agregar un elemento de tipo "Solid"



- Luego, no se muestra ningún objeto nuevo en la escena 3D. No obstante, dentro del árbol de escena, se expande el nodo "Solid" y se debe seleccionar el campo "children", donde se agrega el campo "Shape" usando el botón "Add". En la **Figura 89** se representa este paso.

Figura 89

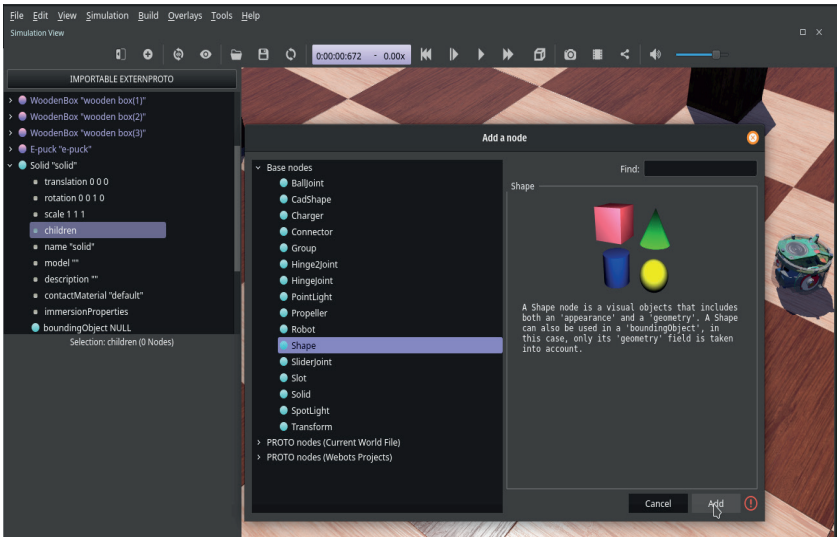
Formas geométricas agregadas al elemento "Solid"



- A continuación, se muestra la ventana generada, donde el usuario debe expandir los archivos de “Base Nodes”, mostrando el atributo “Shape” y, finalmente, hacer clic en agregar. En la **Figura 90** se muestra este procedimiento.

Figura 90

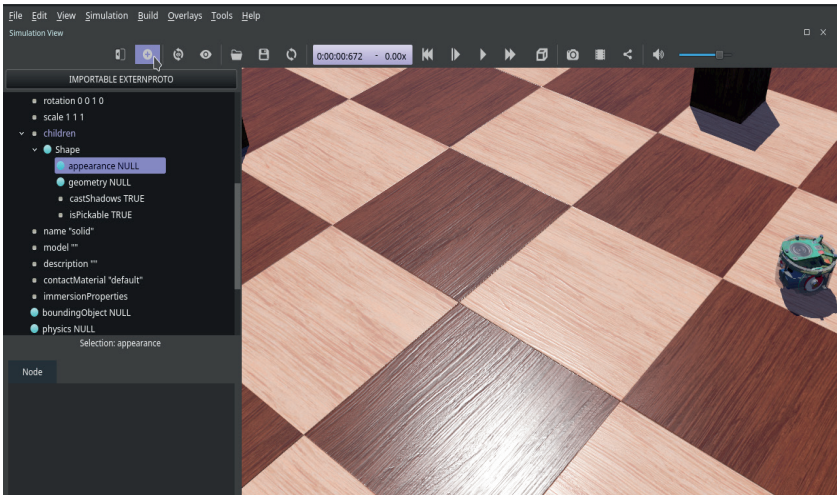
Ilustración de las diferentes formas geométricas disponibles para el elemento “Solid”



- Una vez agregada esta característica, el campo “children” presenta el campo “Shape”. El usuario debe posicionarse en el campo “appearance” y presionar el botón “Add”. En la **Figura 91** se indica este procedimiento.

Figura 91

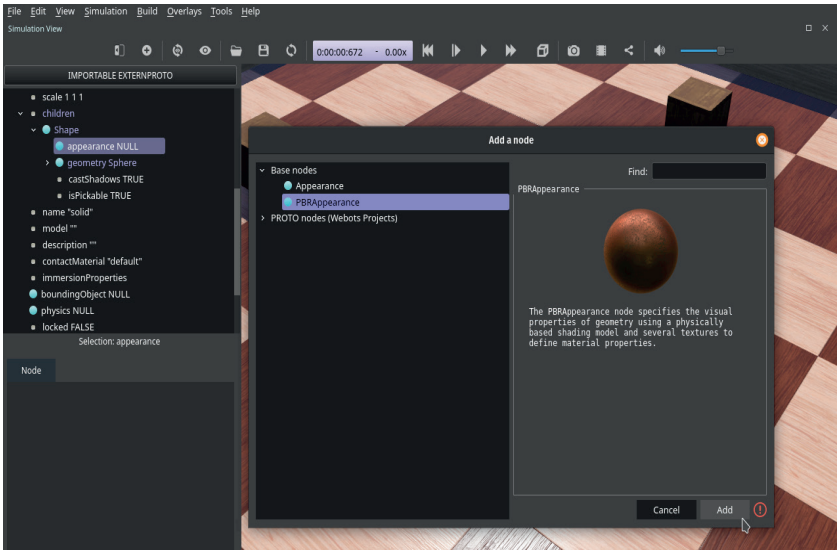
Asignación de apariencia a las propiedades de forma



- Automáticamente, aparece una nueva ventana, donde el usuario debe expandir la opción “Base node” para así agregar el campo “PBRAppearance”. En la **Figura 92** se muestra este paso.

Figura 92

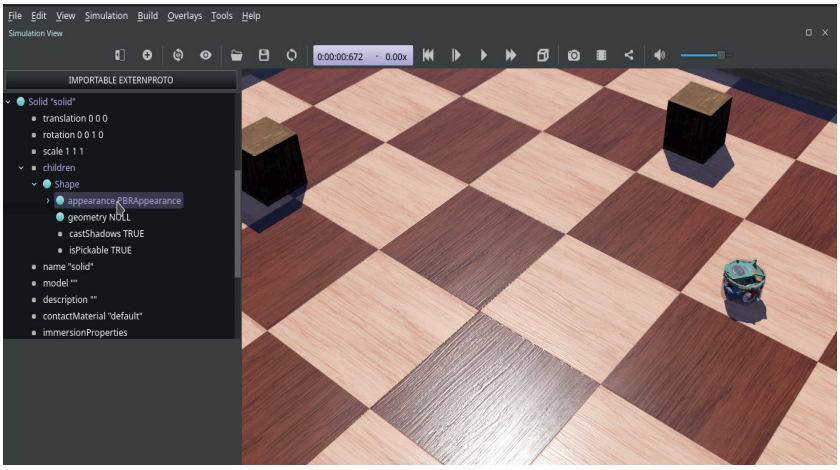
Ventana auxiliar con las diferentes propiedades de apariencia



- Tras este paso, el árbol de escena debe visualizarse de la siguiente forma, con los campos “material”, “texture” y “geometry” dentro del shape del objeto. En la **Figura 93** se muestra lo previamente descrito.

Figura 93

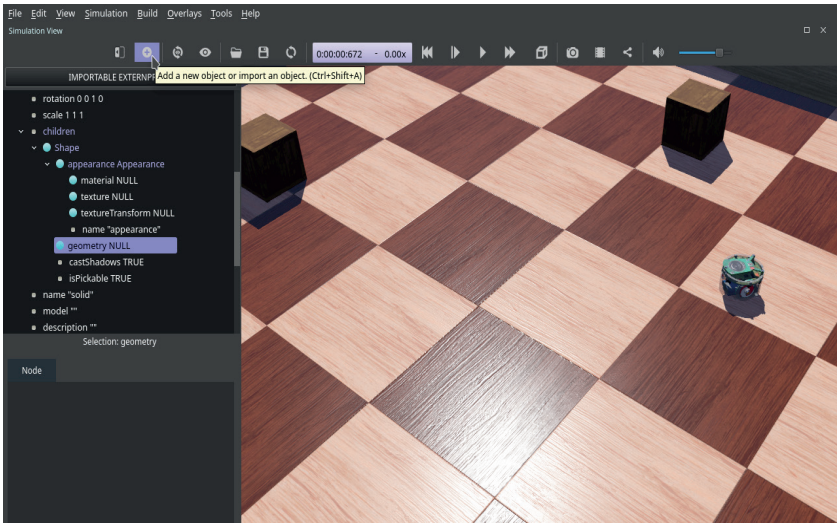
Ilustración de campos "texture" y "geometry" dentro del elemento



- Una vez completados estos pasos, el usuario debe agregar la geometría respectiva al objeto. Para esto, es preciso recordar que se desea agragar un balón o cilindro al ambiente de simulación. Por lo tanto, el usuario debe seleccionar la opción "geometry" y presionar el botón "Add". En la **Figura 94** se muestra este procedimiento.

Figura 94

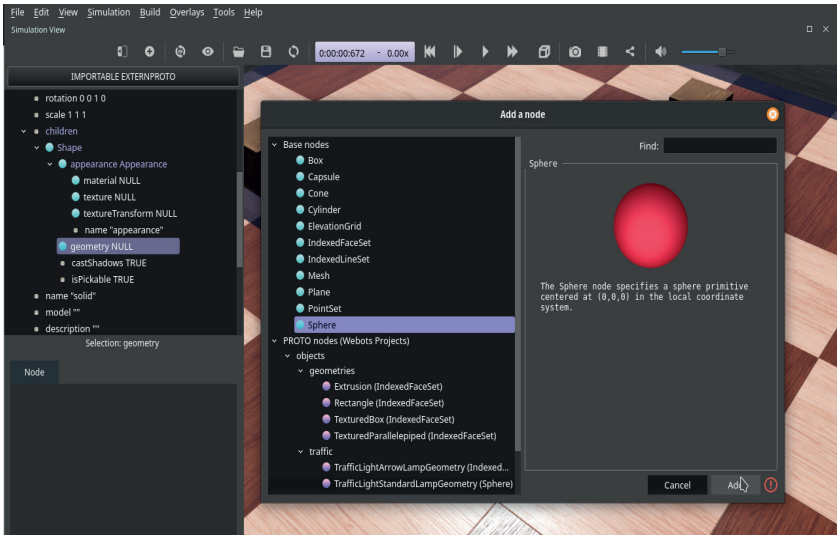
Una nueva geometría se agrega al elemento



- Después, aparece una nueva ventana `para que el usuario pueda elegir entre distintas geometrías básicas. Entonces, se debe seleccionar la geometría “Sphere”. En la **Figura 95** se muestra este paso.

Figura 95

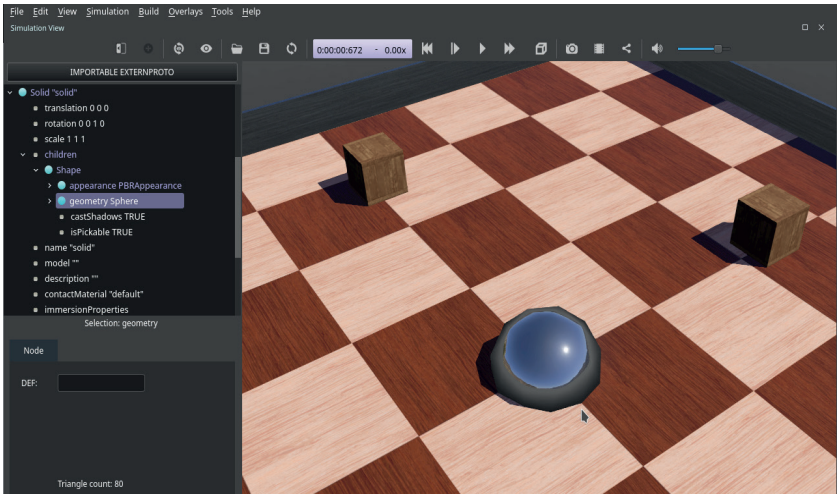
Ilustración de la selección de una esfera como geometría para el elemento tipo "Solid"



- Una vez que se haya completado el paso previo, se visualiza una esfera dentro del ambiente de simulación; esta se ubica en el cero absoluto del entorno. En la **Figura 96** se representa lo anteriormente descrito.

Figura 96

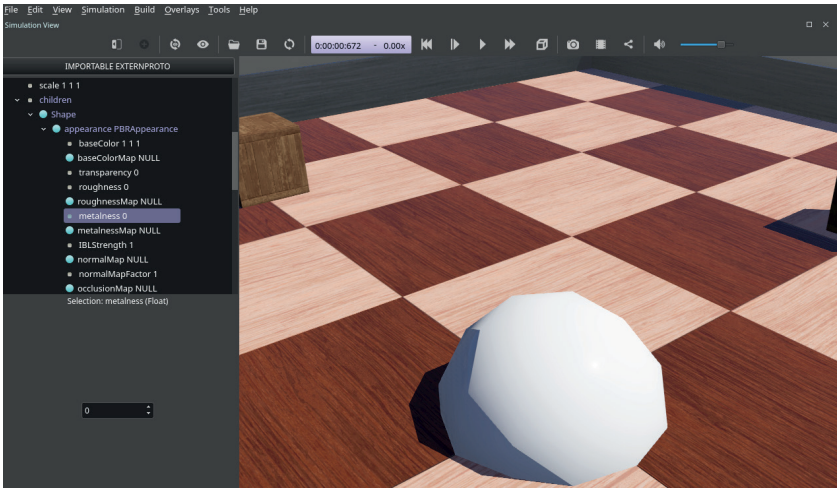
Elemento "Solid" con su respectiva geometría dentro del ambiente de simulación



- Para modificar la apariencia de la bola o esfera, el usuario debe dirigirse a los campos de "PBRAppearance" y ajustar el valor del campo "metalness" a cero. En la **Figura 97** se muestra el resultado de este procedimiento.

Figura 97

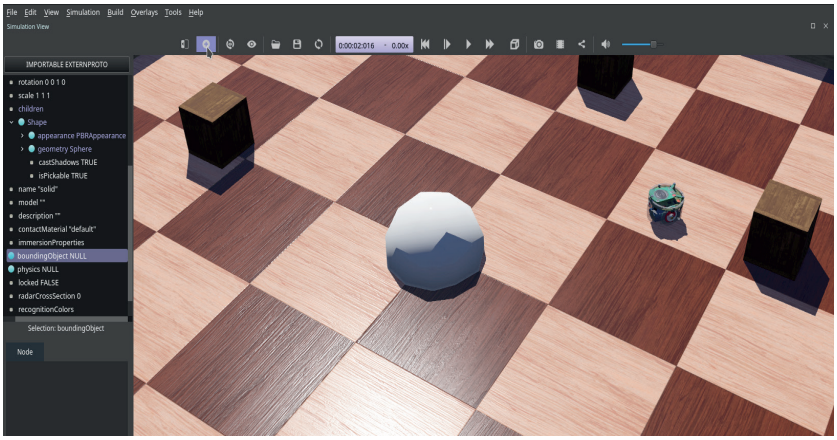
Asignación de nuevos valores al parámetro 'metalness'



- Finalmente, para dotar al objeto de propiedades como masa o inercias, es necesario adjuntar una geometría al "boundingObject". Este parámetro forma parte de los campos de "Solid". Cuando te encuentres en este campo, presiona el botón "Add". En la **Figura 98** se presenta una representación gráfica que indica dónde encontrar este elemento.

Figura 98

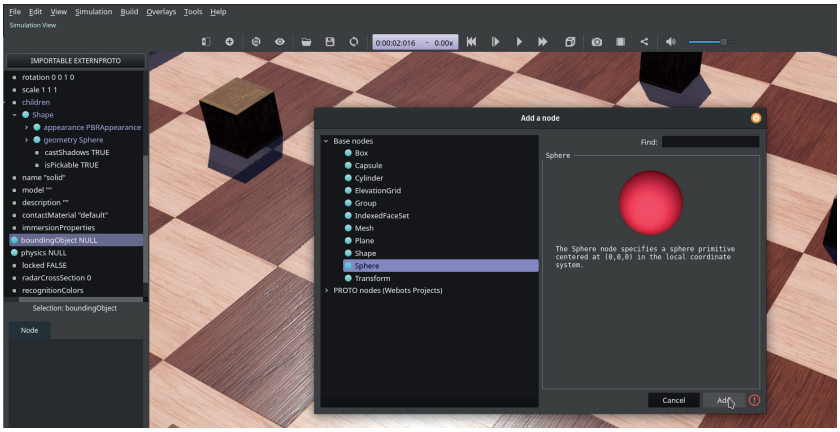
Asignación de propiedades de colisión al elemento "Solid"



- Luego, se presenta nuevamente un menú de asistencia para el usuario. Debe seleccionar la misma geometría que se eligió en el campo "Shape". Es importante mencionar que el "boundingObject" puede tener geometrías primitivas para mejorar el cálculo de colisiones. En la **Figura 99** se ilustra este paso.

Figura 99

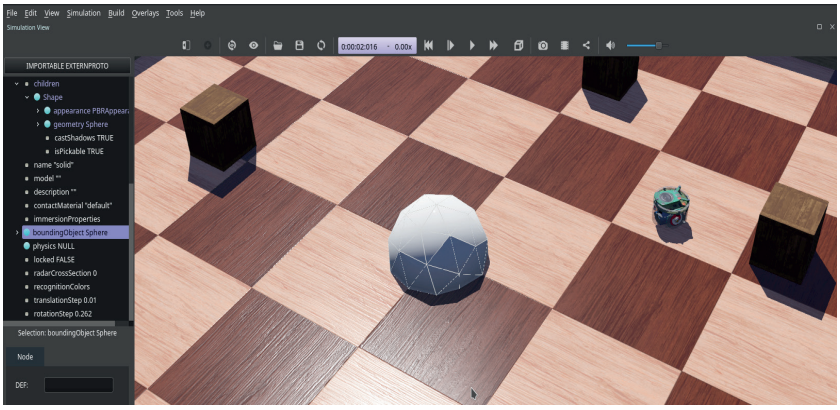
Selección de la geometría de colisión para el elemento "Solid"



- Aquí se puede apreciar que se ha creado una malla de colisiones para el objeto. Esto le será de ayuda al simulador para calcular colisiones y variables físicas. En la **Figura 100** consta la representación de lo descrito.

Figura 100

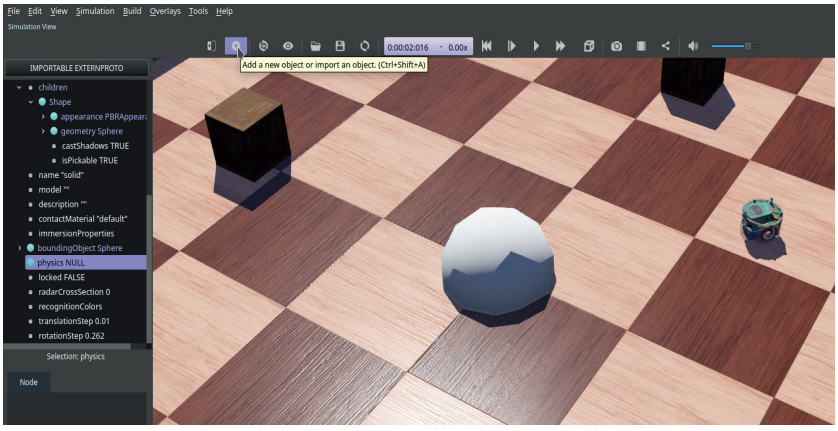
Representación de las propiedades de colisión del elemento "Solid"



- Como uno de los últimos pasos, se debe agregar el campo "physics" en "Solid" y luego presionar "Add". En la **Figura 101** se indica este procedimiento.

Figura 101

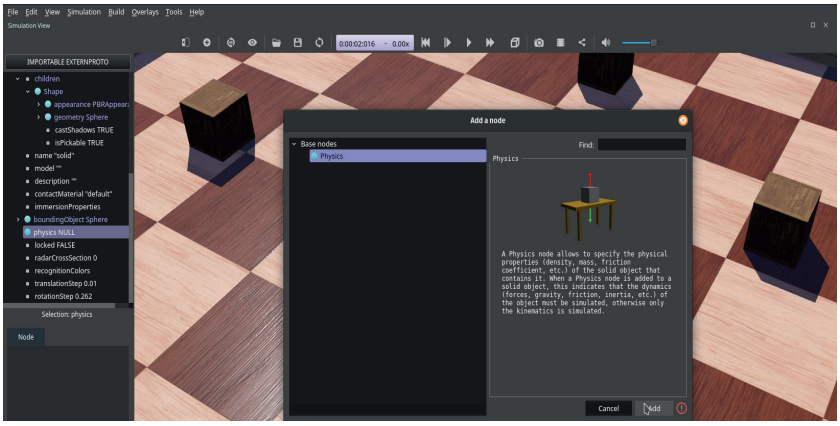
Asignación de físicas al elemento "Solid"



- Nuevamente, se muestra una ventana de asistencia. El usuario deberá incluir el parámetro “Physics”. En la **Figura 102** se muestra este paso.

Figura 102

Ventana auxiliar para asignación de físicas



- Finalmente, se deben guardar los cambios realizados y reiniciar la simulación.

Para mayor información, se tiene disponible el ambiente de simulación en el enlace https://bit.ly/3zwpFLh. El usuario, simplemente, debe abrir el ambiente de simulación desde Webots.

Con la información proporcionada en esta sección, el usuario o lector tendrá los conceptos fundamentales para trabajar dentro de Webots. En las siguientes secciones se abordarán temas como el diseño de un robot tipo diferencial y un manipulador desde cero en Webots, así como la inclusión de

los sensores necesarios para realizar algoritmos de control. En último lugar, se desarrollarán los controladores que tengan acceso a toda la información proveniente de los sensores.

4

Simulación de Robots Móviles

En esta sección del libro se abordará la creación de un robot móvil tipo diferencial. Para esto, se incluirán cada uno de los pasos que el usuario debe considerar al momento de crear el robot dentro del simulador Webots. Adicionalmente, se explicará cómo agregar sensores al robot previamente creado, para así otorgarle un mayor número de funcionalidades que, posteriormente, serán utilizadas en el diseño de algoritmos de control.

4.1 Crear un Robot Móvil en Webots

Para la creación del robot móvil, el usuario debe tener una leve conceptualización de modelado 3D. No se considera el uso directo de herramientas de diseño 3D debido a que en su mayoría son de pago, mientras que Webots es un *software* libre. Los pasos para una correcta creación del cuerpo del robot se presentan a continuación.

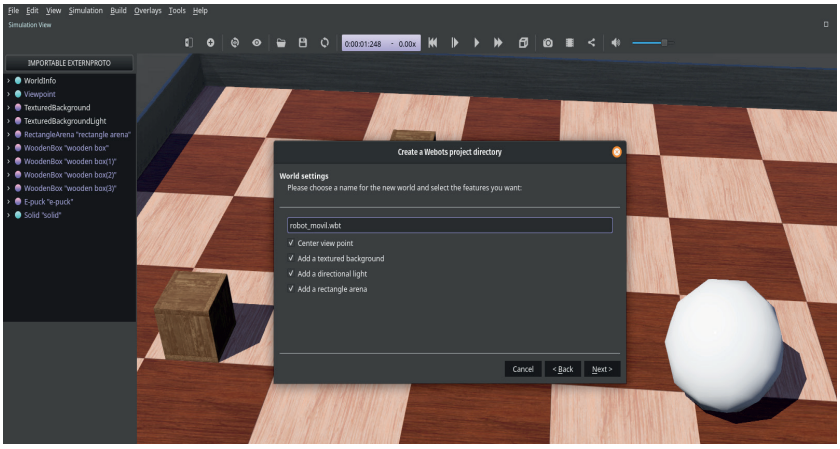
Cuerpo del robot móvil

Crear un nuevo directorio de trabajo, como ejemplo práctico, se considera 'Robot_Movil'. Se debe incluir 'rectangle arena' y el nombre del archivo del mundo es 'robot_movil'. En la **Figura 103** se muestra más información sobre este paso. Este directo-

rio se utiliza para organizar y almacenar archivos relacionados con el proyecto del robot móvil en Webots; esto facilita la gestión y la estructuración del proyecto.

Figura 103

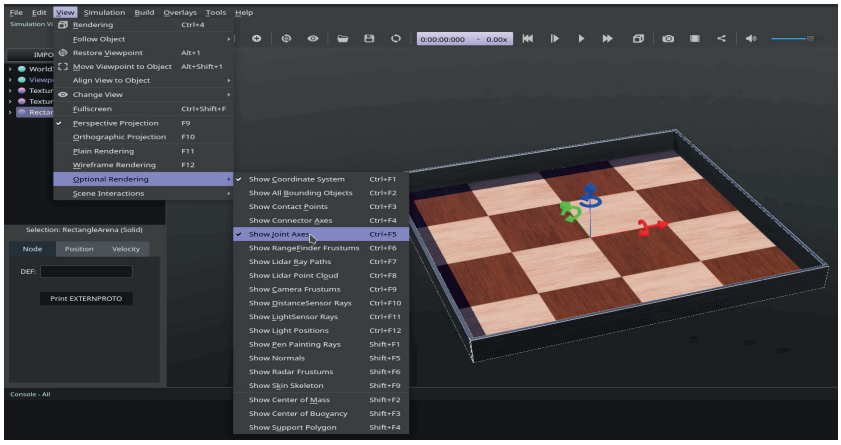
Creación de un nuevo directorio de trabajo llamado 'Robot_movil'



- Dentro del menú 'View', el usuario debe seleccionar el submenú 'Optional Rendering', en el que se presentan varias opciones. Sin embargo, solo debe escoger 'Show coordinate system' y 'Show Joint Axes', con lo cual el usuario es capaz de visualizar los ejes de giro del sistema, así como el sistema de referencia inercial. En la **Figura 104** se presenta más información sobre lo descrito.

Figura 104

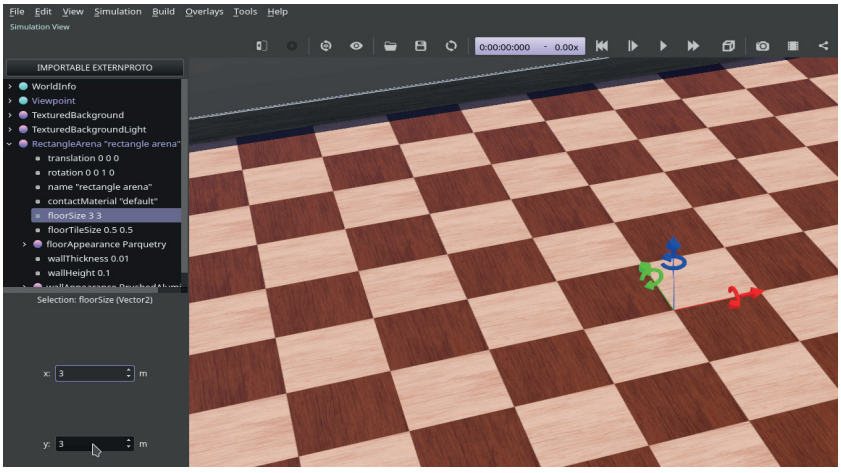
Activación de sistemas de coordenadas para visualizar el sistema inercial de la simulación



- Las dimensiones de 'Rectanglearena' deben ser seleccionadas por el usuario a su conveniencia; sin embargo, se recomienda asignar al menos 3 metros para los ejes 'x' y 'y' dentro del parámetro 'floorsize'. En la **Figura 105** se muestra una representación de dicho paso.

Figura 105

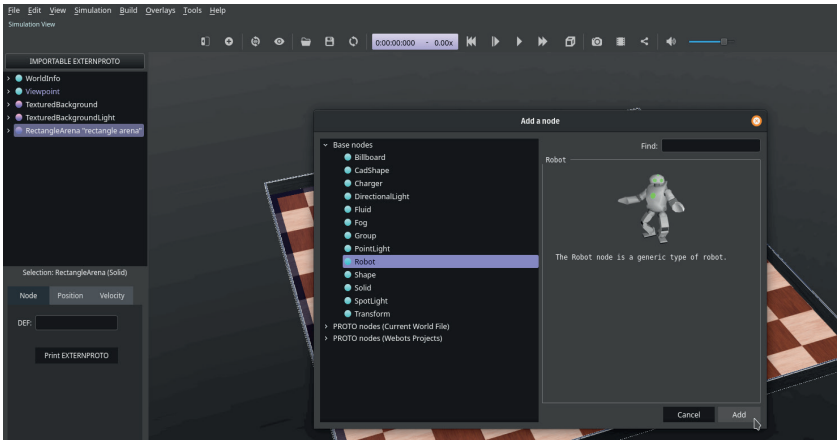
Acción que asegura que las dimensiones del 'RectangleArena' se asignen adecuadamente



- Después, se debe añadir un nuevo objeto dentro del ambiente de simulación, específicamente, se incluye un objeto tipo robot. En la **Figura 106** se muestra a detalle este paso.

Figura 106

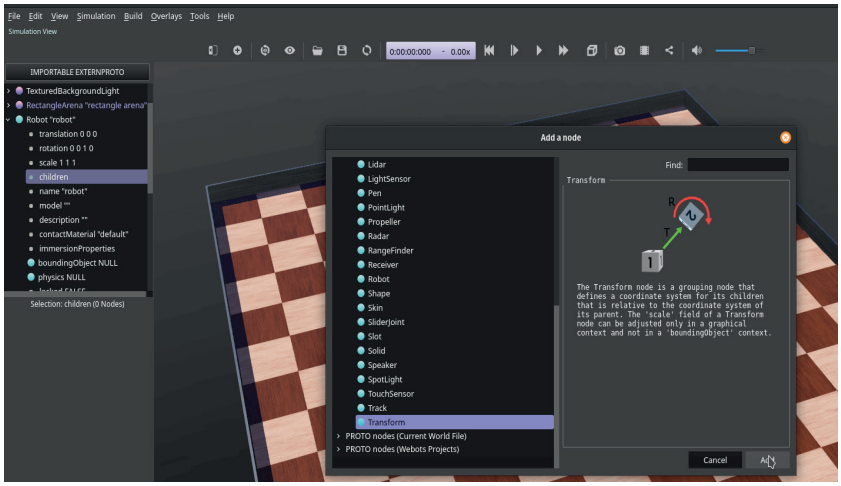
Representación de cómo incluir un elemento tipo robot en Webots



- Una vez creado el objeto robot, puede expandir la información de este objeto en el árbol de escena. El usuario se dirige a la pestaña 'children', donde se agrega el objeto 'transform'. En la **Figura 107** se muestra más información sobre este paso.

Figura 107

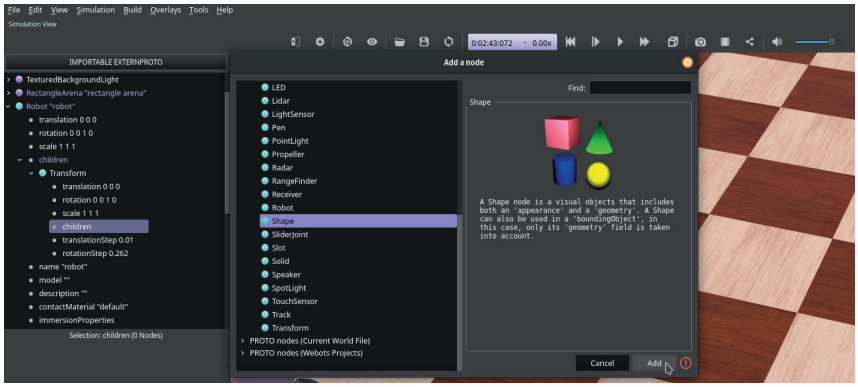
Propiedades de tipo "transform" se incluyen dentro del elemento robot



- Luego, se expande la información presente en 'transform', donde se agrega un elemento 'shape' dentro de la pestaña 'children'. En la **Figura 108** se muestra este procedimiento.

Figura 108

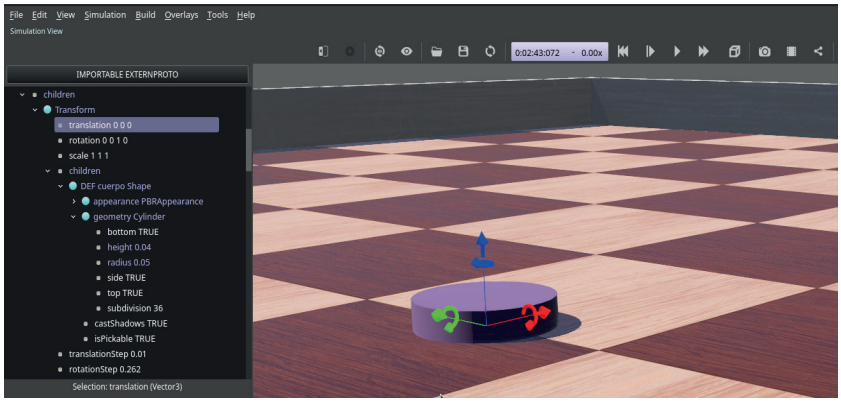
Incorporando las propiedades de tipo 'shape' al elemento del robot



- Dentro de la información de 'shape', se agrega una figura de tipo cilíndrica en 'geometry', donde los parámetros a introducir son 'height = 0.04' y 'radius = 0.05'. Además, se seleccionan los colores del objeto en el parámetro 'appearance'. También se debe definir el nombre del elemento 'shape' como 'cuerpo'. En la **Figura 109** se muestra este procedimiento.

Figura 109

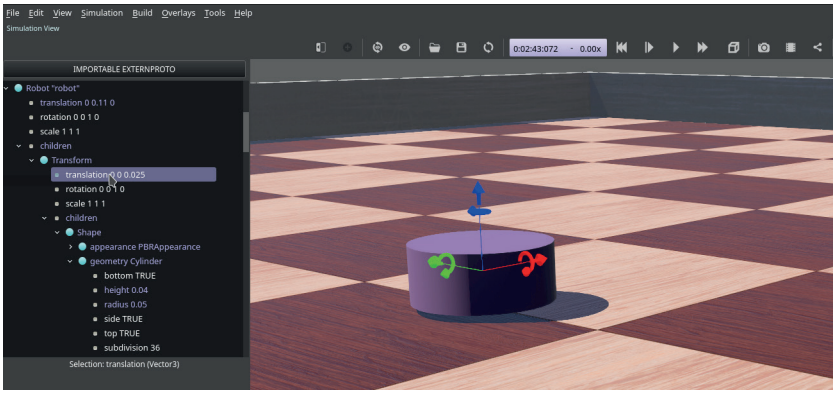
Presentación del cuerpo del sistema robótico móvil en Webots



- Esta sección del robot es el cuerpo. Para agregar las ruedas correctamente, se debe generar un pequeño desfase entre el cuerpo y el suelo. Esta modificación se puede realizar al cambiar los parámetros presentes en el campo 'transform', respectivamente el valor de 0.025 en el eje "z". En la **Figura 110** se muestra el resultado de este procedimiento.

Figura 110

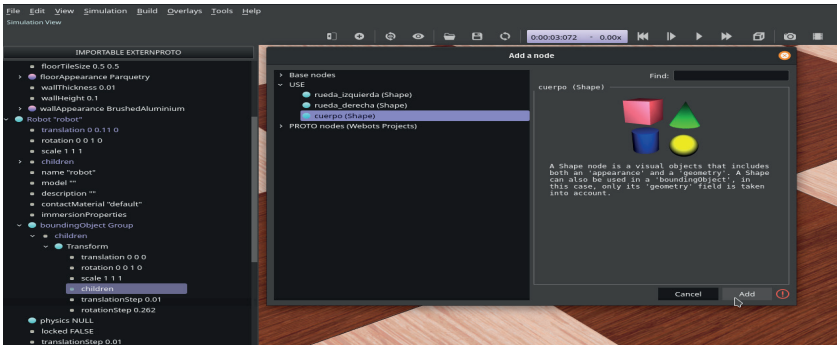
Ilustración del correcto desplazamiento del cuerpo del sistema robótico móvil



- Finalmente, se agragan las propiedades de masa e inercia al elemento generado. Para ello, dentro de las propiedades del 'robot' se encuentra 'boundingObject', en el que se debe incluir el elemento 'group' que contiene el parámetro 'Transform'; dentro de este, se agrega la geometría ya creada. Este elemento se enumera como 'shape' con el nombre 'cuerpo'. En la **Figura 111** se indica más información sobre este procedimiento.

Figura 111

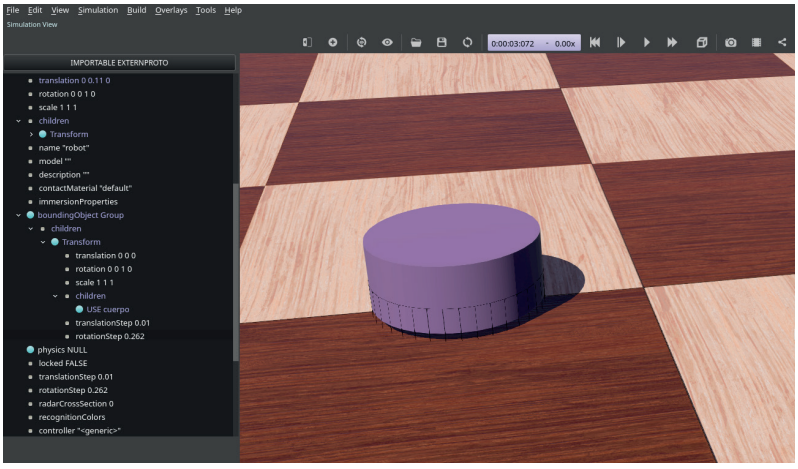
Propiedades de colisión agregadas al sistema robótico



- El resultado previo no será el correcto debido a un desfase presente. En la **Figura 112** se visualiza este problema.

Figura 112

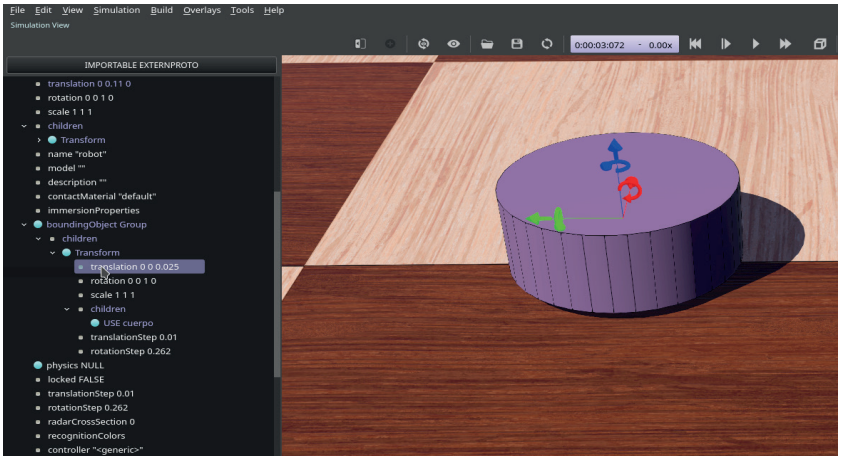
Propiedades de colisión relacionadas con el desplazamiento del cuerpo



- Para corregir el error que se presenta como un desfase, se debe utilizar el elemento 'transform'. En la **Figura 113** se muestra el resultado de lo descrito.

Figura 113

Propiedades de colisión perfectamente posicionadas



- De esta manera, se completa el desarrollo del cuerpo del robot. En la siguiente sección, se demostrará cómo generar una de las ruedas.

Ruedas del robot móvil

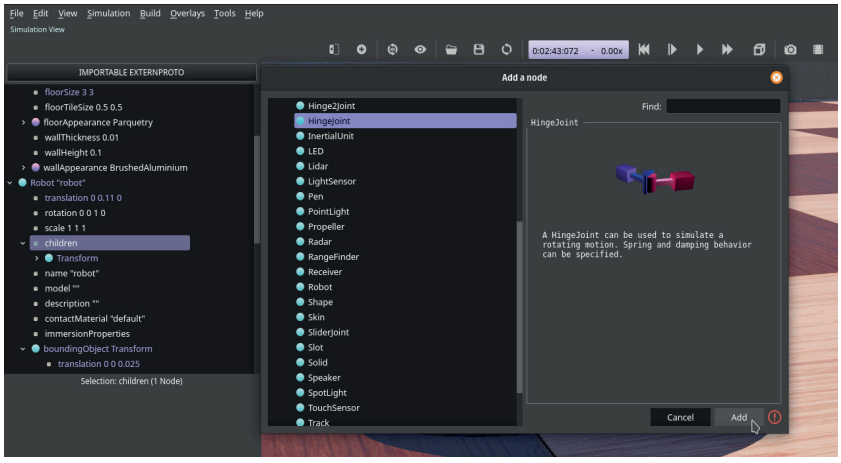
A continuación, se muestran los pasos necesarios para crear ruedas actuadas por motores dentro del simulador robótico Webots.

Dentro del elemento 'Robot', seleccionar su parámetro 'children', en el que se incluye el elemento 'HingeJoint', que per-

mite crear elementos rotativos sobre un determinado eje. En la **Figura 114** se presenta este procedimiento.

Figura 114

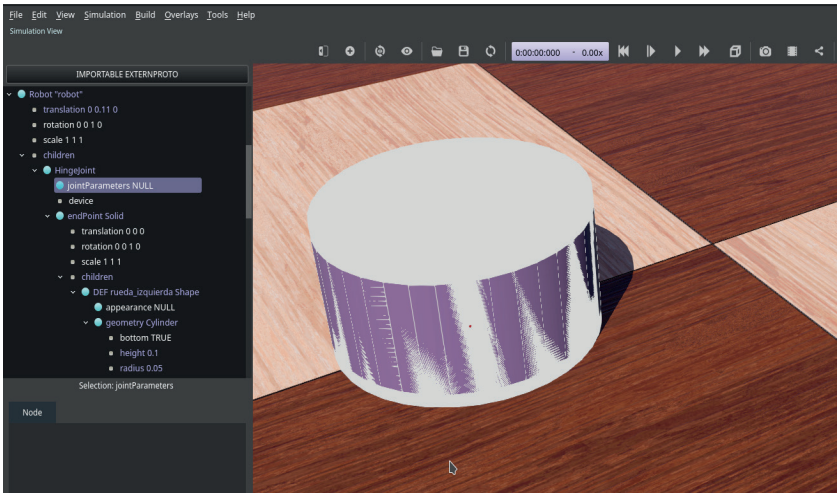
Representación de cómo incluir propiedades tipo "HingeJoint" dentro del sistema



- Dentro de los elementos de 'HingeJoint', el usuario debe seleccionar el denominado 'endPoint', en el que se debe añadir un elemento tipo 'Solid' para así, dentro de 'Solid', incluir un componente 'Shape' con una respectiva forma cilíndrica. El elemento 'Shape' puede ser nombrado 'rueda_izquierda' para su posterior uso dentro del sistema de colisiones. En la **Figura 115** se muestra una representación de lo explicado.

Figura 115

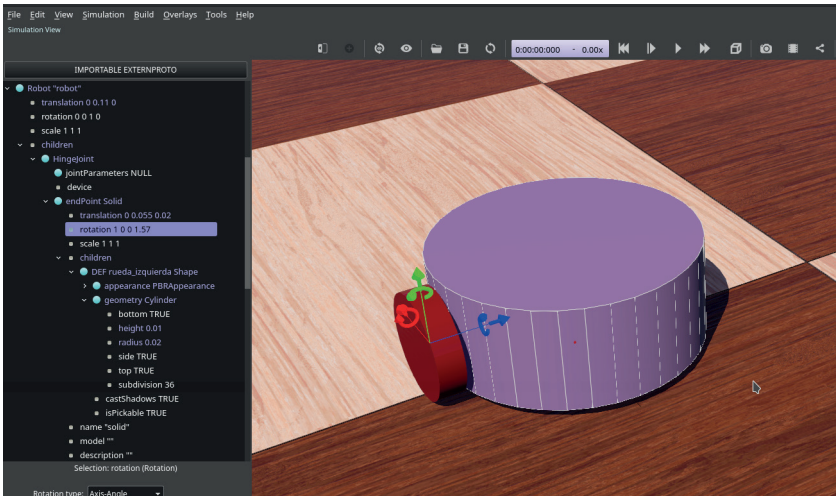
Configuración inadecuada de las propiedades de tipo "Hinge-Joint"



- Como se observa, por defecto no se obtiene el resultado deseado. Por lo tanto, el usuario debe modificar las dimensiones del cilindro, es decir, 'height = 0.01' y 'radius = 0.02'. Finalmente, utilizando las propiedades del elemento 'Solid', es posible trasladar y orientar la rueda. En la **Figura 116** se muestra el procedimiento descrito.

Figura 116

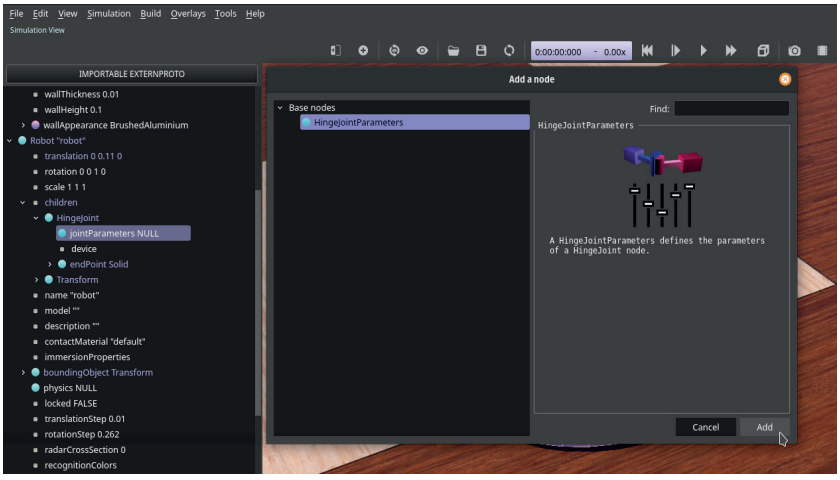
Configuración correcta del elemento "HingeJoint" con su respectiva geometría



- El siguiente paso es determinar el eje de giro de la rueda. Para lograr esto, el usuario debe seleccionar la opción 'jointParameters' y agregar el parámetro 'HingeJointParameters'. En la **Figura 117** se indica detalladamente este proceso.

Figura 117

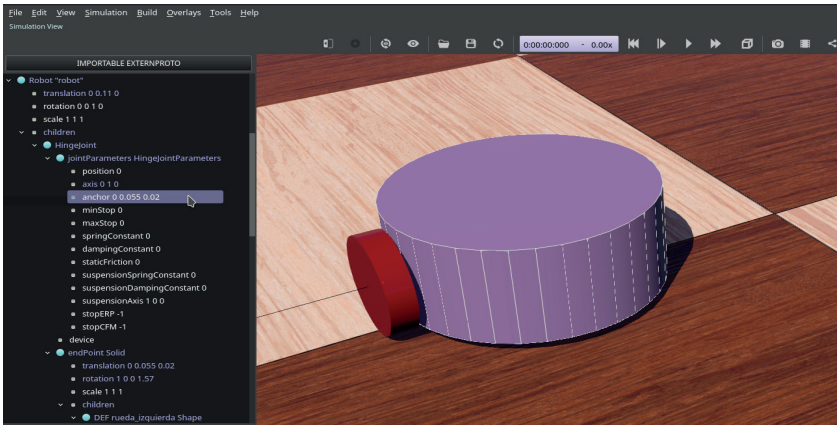
Configuración del eje de giro rueda izquierda



- Una vez creado dicho parámetro, se modifica el 'axis' de rotación, el cual estará definido únicamente para el eje 'y'. Además, el parámetro 'anchor' debe coincidir con la posición del 'Solid' creado previamente. En la **Figura 118** se presenta la asignación de estos valores.

Figura 118

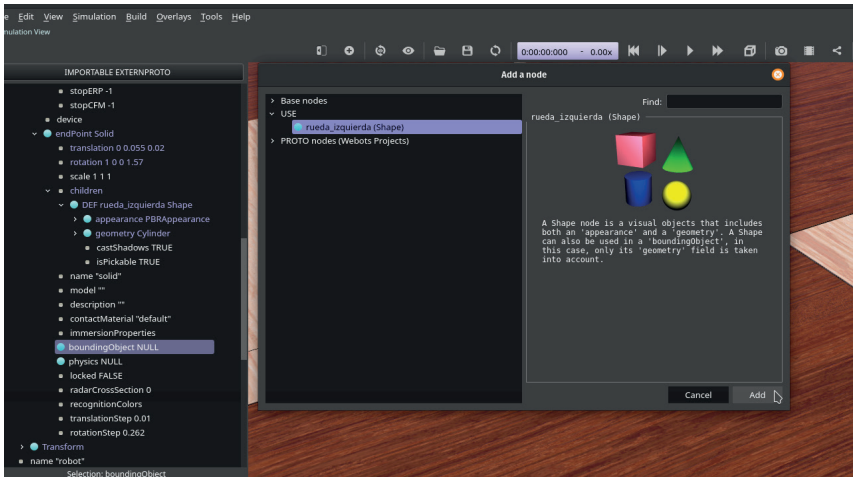
Selección del eje de rotación apropiado para la rueda izquierda



- A continuación, se añaden los elementos de colisiones respectivos de dicha rueda. Para esto, dentro de los parámetros del elemento 'Solid', se encuentra el elemento 'boundingObject', donde se incluye el nombre del 'Shape' respectivo que, en este caso, es 'rueda_izquierda'. En la **Figura 119** se muestra el respectivo procedimiento.

Figura 119

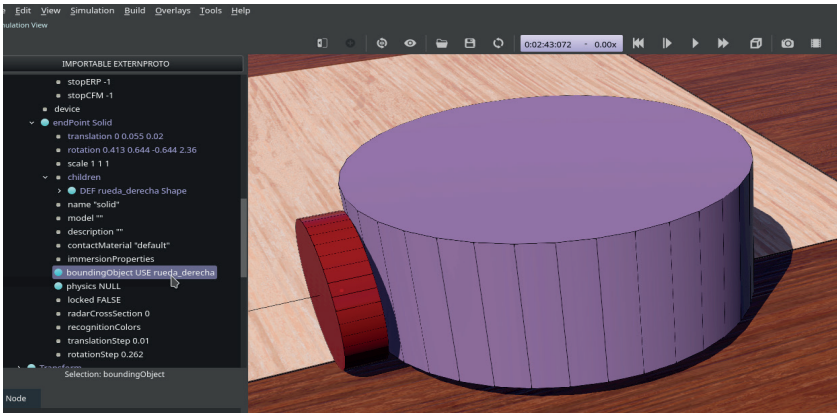
Propiedades de colisión para la rueda la izquierda



- En la **Figura 120** se puede visualizar que la rueda presenta las mallas necesarias para detectar colisiones.

Figura 120

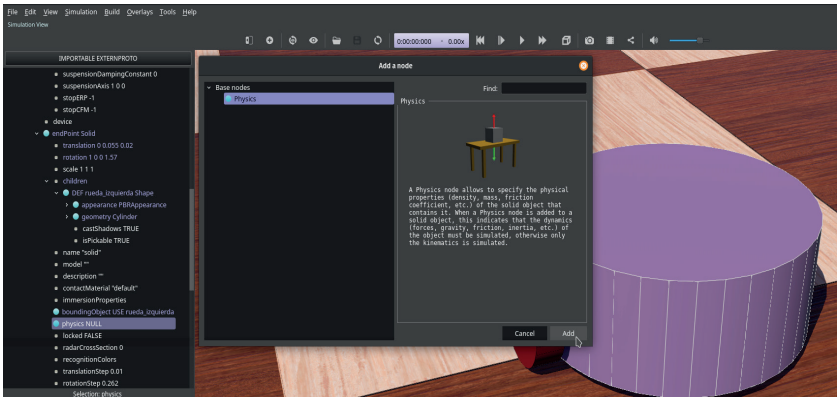
Resultado de geometrías de colisión correctamente posicionadas



- Por otro lado, para que la rueda disponga de masa e inercias, se incluyen las físicas necesarias. Para ello, se agrega el elemento 'Physics' dentro del parámetro 'physics'. En la **Figura 121** se presenta este procedimiento.

Figura 121

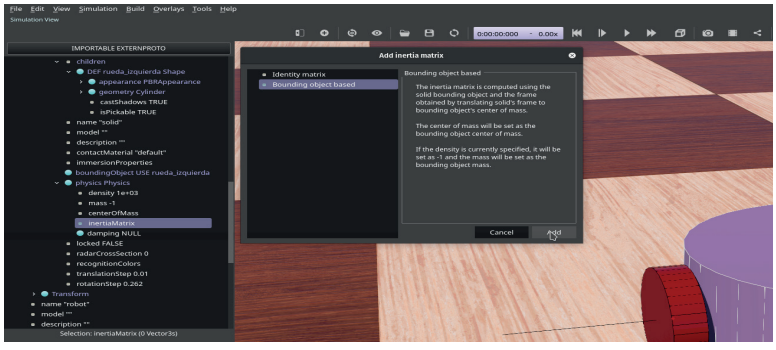
Ventana auxiliar para agregar propiedades físicas a las ruedas



- Finalmente, dentro de la lista de objetos generados se tiene el parámetro 'inertiaMatrix', en el que se debe incluir la opción 'Bounding object based', con la cual se calcula automáticamente la masa e inercia de la rueda. En la **Figura 122** se visualiza este procedimiento.

Figura 122

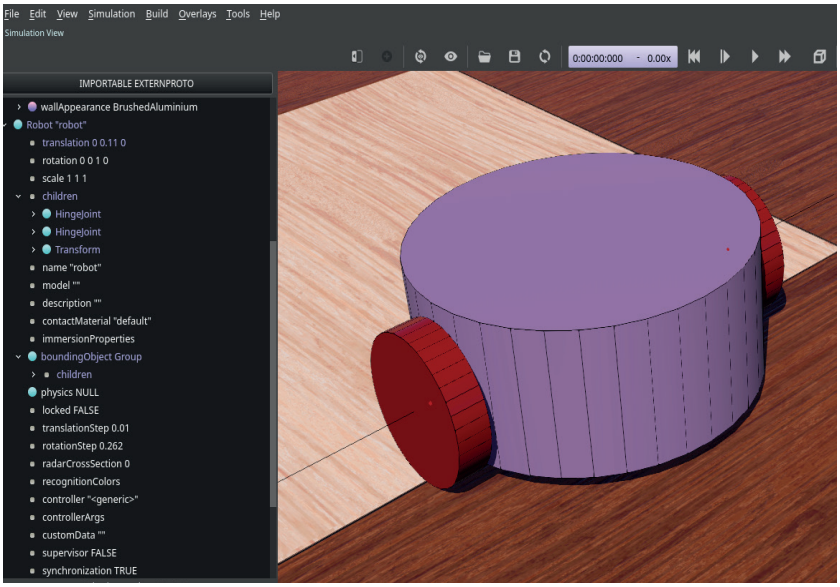
Ventana que proporciona la opción de asignar la matriz de masas e inercias a la rueda izquierda del vehículo



- Para la rueda derecha, se repite este procedimiento. Sin embargo, el desplazamiento para el eje “y” tiene el valor de -0.055; este también debe incluirse dentro de los parámetros del “anchor”. Finalmente, el nombre asignado a la variable ‘Shape’ debe ser “rueda_derecha”. En la **Figura 123** se muestra una representación final del sistema.

Figura 123

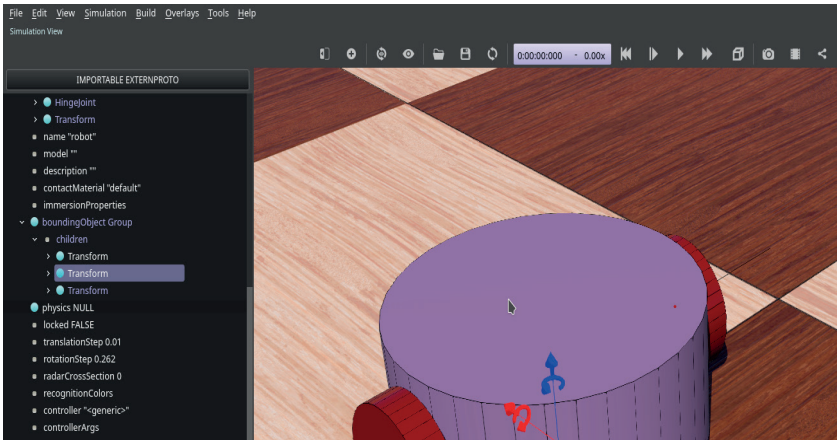
Representación de la rueda derecha del sistema robótico móvil



- Después. se agregan los elementos de colisión de las dos ruedas creadas. Para esto, el usuario debe incluir dos objetos tipo “transform” dentro del parámetro ‘boundingObject’ del ‘robot’. En la **Figura 124** se muestra este procedimiento.

Figura 124

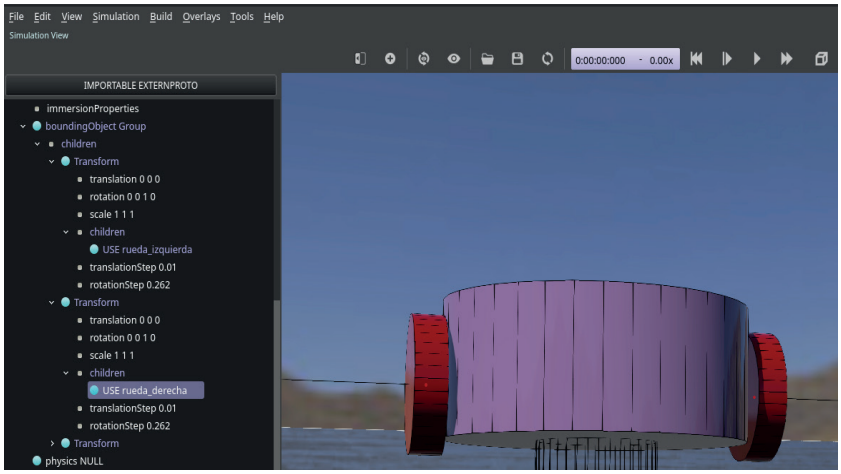
Propiedades de colisión para las ruedas derecha e izquierda



- A continuación, se incluye cada una de las geometrías de las ruedas, las cuales fueron guardadas con los nombres de 'rueda_derecha' y 'rueda_izquierda'. Sin embargo, al momento de incluir estos elementos, no están ubicados en la posición correcta. En la **Figura 125** se muestra lo anteriormente descrito.

Figura 125

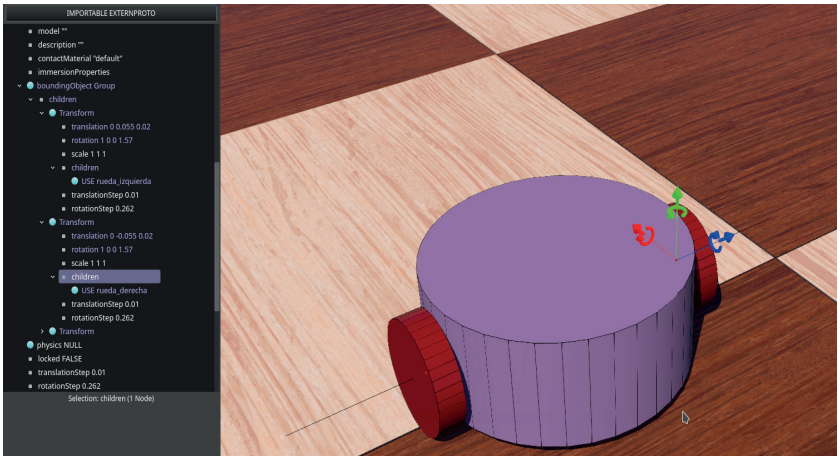
Geométricas de colisión para las ruedas derecha e izquierda



- Con el elemento 'transform' es posible hacer las correcciones respectivas, que consisten en trasladar y rotar los elementos hasta ubicarlos en la posición deseada. En la **Figura 126** se presenta mayor información sobre dichos valores, los cuales fueron asignados tanto a la traslación como a la rotación de cada rueda.

Figura 126

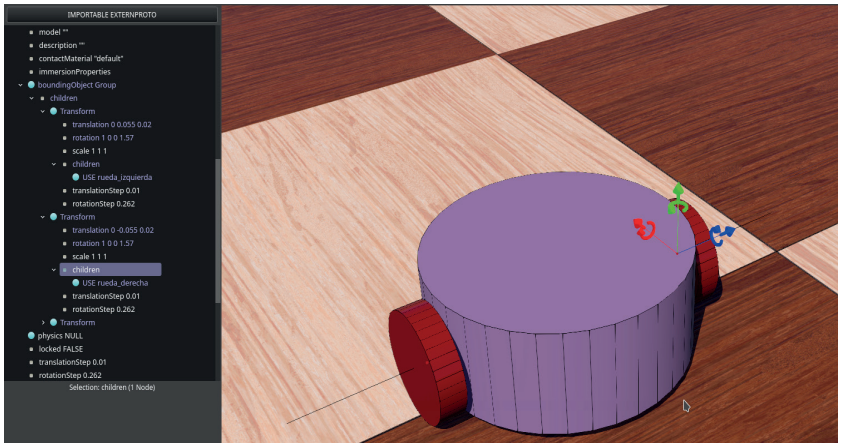
Posicionamiento correcto de las geometrías de colisión



- Finalmente, dentro de los parámetros del 'robot' se integran las físicas; para lo cual, se selecciona dentro del elemento 'inertiaMatrix' la opción 'Bounding object', que calcula automáticamente la masa del robot y su matriz de inercia. Con esto, se culmina el proceso de creación de un robot tipo móvil. En la **Figura 127** se muestra la representación final del sistema.

Figura 127

Representación de un robot móvil dentro del ambiente de simulación de Webots



Para mayor información, está disponible el ambiente de simulación en el enlace https://bit.ly/3VQ6bZq, bajo el nombre de "Robot_Movil". El usuario simplemente debe abrir el ambiente de simulación desde Webots.

4.2 Sensores y Actuadores del Robot Móvil

A continuación, se agregarán los actuadores y sensores al robot móvil tipo diferencial. El robot debe disponer de sensores de posición y velocidad angular para cada rueda, un GPS, que permite conocer la traslación del robot respecto al sistema de referencia y una unidad de medición inercial IMU, con la cual es posible obtener los ángulos de orientación del robot móvil. Además, los actuadores necesarios están relacionados con el movimiento angular de cada una de las ruedas, los cuales pue-

den ser catalogados como motores de corriente continua, ya que imitan su comportamiento. Con este tipo de actuador, el robot móvil se puede desplazar y rotar.

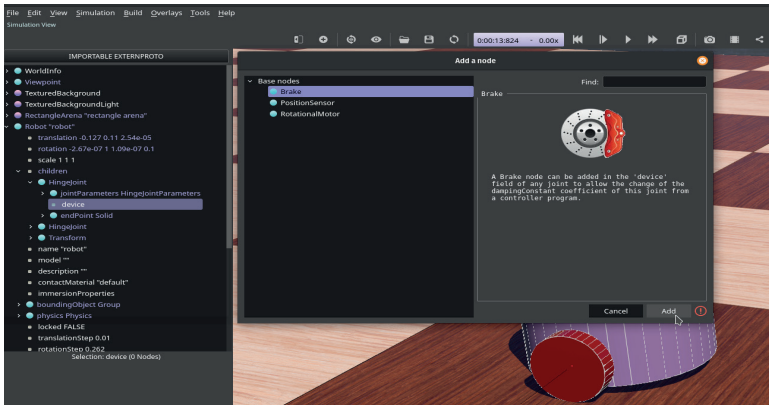
Sensor de posición y velocidad angular (ruedas)

Este tipo de sensor nos permite conocer el desplazamiento angular y, además, se puede determinar la velocidad angular de cada rueda. Para agregar estos componentes al robot móvil previamente diseñado, se deben seguir las siguientes instrucciones:

Dentro del árbol de escena, específicamente en el parámetro 'HingeJoint' de la rueda izquierda o derecha, se encuentra el elemento 'device', donde el usuario puede agregar el sensor de posición angular. En la **Figura 128** se muestra una representación de este procedimiento, que debe llevarse a cabo tanto en la rueda derecha como en la izquierda.

Figura 128

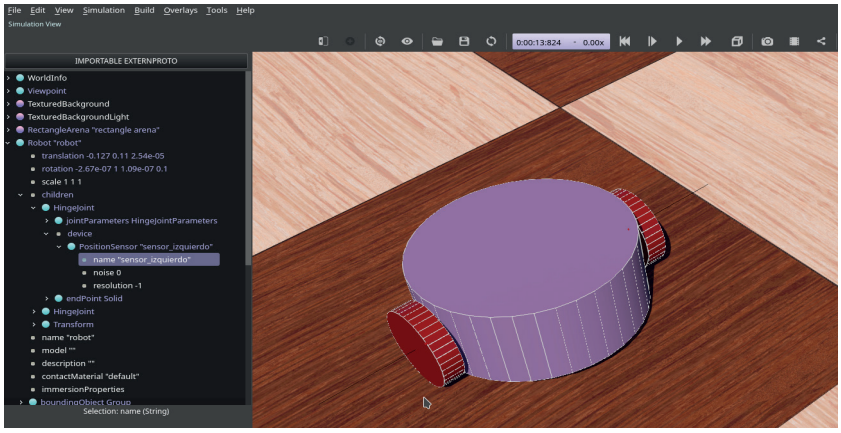
Ventana auxiliar para incluir el sensor de desplazamiento angular



- Estos sensores deben tener un nombre adecuado; por ejemplo, 'sensor_izquierdo'. Es preciso recordar que este procedimiento debe ser realizado para ambas ruedas. En la **Figura 129** se muestra su correcta implementación.

Figura 129

Información adicional del nombre de cada sensor de desplazamiento angular



- Las instrucciones presentadas son adecuadas para que el usuario tenga acceso a la información de la posición angular de cada rueda, recordando el nombre asignado para cada elemento.

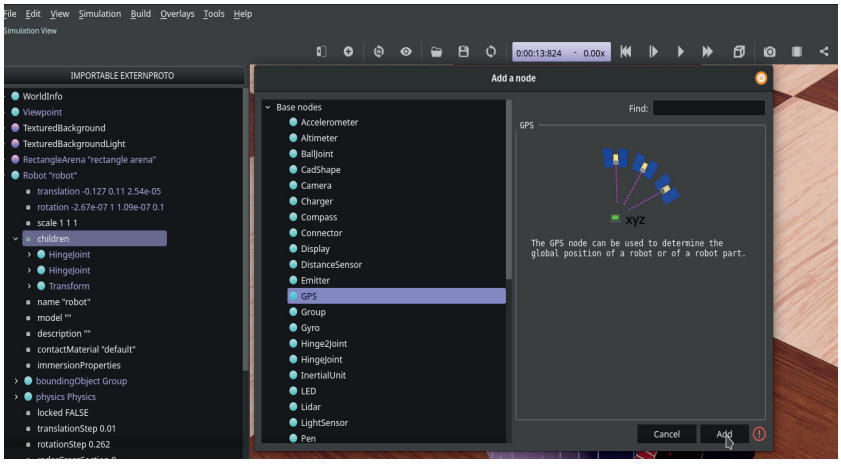
Sensor de GPS

Este tipo de sensor permite al usuario acceder a la localización del robot móvil. En los siguientes pasos se presentan las instrucciones necesarias para integrar este elemento a la simulación.

Dentro de los parámetros del 'robot', dirigirse al elemento 'children', donde se selecciona el sensor GPS dentro de la gama de sensores que dispone el simulador robótico. En la **Figura 130** se presenta una forma más detallada de esta instrucción.

Figura 130

Ventana auxiliar para agregar un sensor tipo GPS



- Una vez integrado el sensor en el robot móvil, su nombre por defecto es 'gps'. A criterio del usuario, este se puede modificar. Sin embargo, es importante recordar que bajo dicho nombre se puede acceder a la información presente en el sensor. En la siguiente imagen se presenta el nombre asignado por defecto y diferentes parámetros que pueden ser modificados por el usuario.

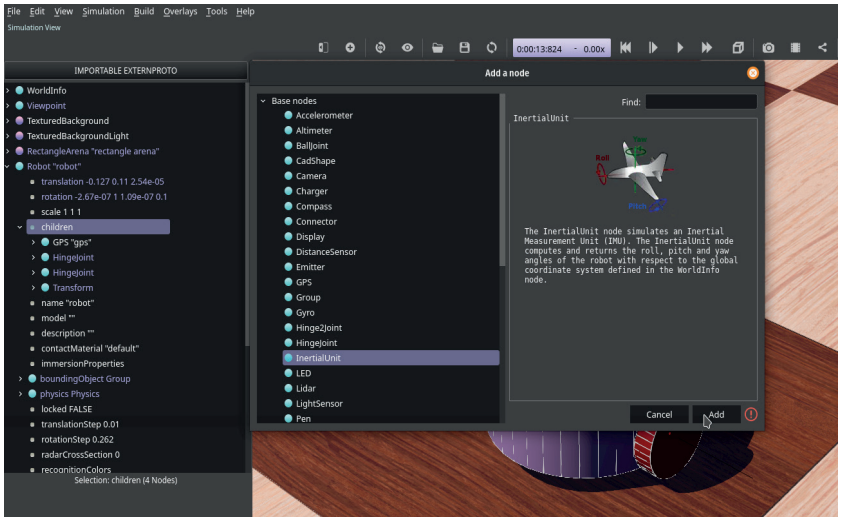
Este sensor permite medir la velocidad y orientación del robot; esto es de mucha utilidad al momento de realizar algo-

ritmos de control u otras aplicaciones. La correcta integración de dicho elemento se muestra en las siguientes instrucciones:

Como en las instrucciones anteriores, para agregar este tipo de sensor, el usuario debe posicionarse en el elemento 'children' del 'robot' y agregar un nuevo elemento. Inmediatamente, se presenta una nueva ventana donde el usuario encuentra este tipo de sensor con el nombre de "InertialUnit". En la **Figura 131** se muestra este procedimiento.

Figura 131

Ventana auxiliar para agregar un sensor de tipo IMU



Una vez completado este procedimiento, se dispone de los sensores necesarios en el robot móvil.

Actuadores rotacionales para las ruedas

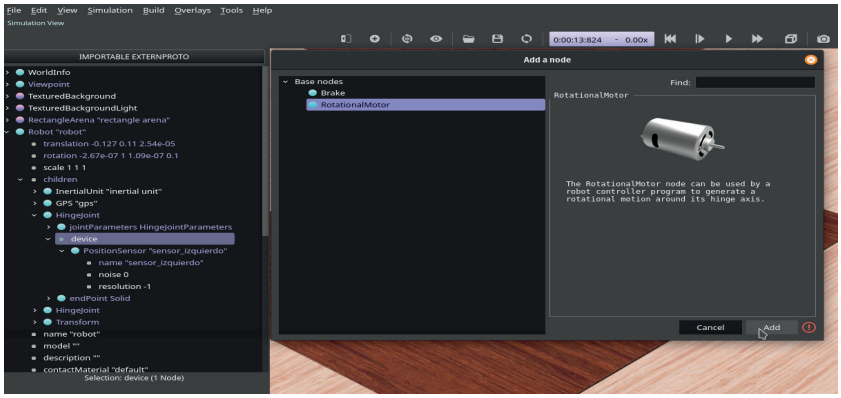
Este tipo de actuador permite accionar las ruedas del robot tipo diferencial, imitando el comportamiento de un motor de co-

rriente continua acoplado al eje de las ruedas. Para una correcta incorporación de dicho elemento, se recomienda seguir las siguientes instrucciones:

- Para incorporar este elemento, el usuario debe posicionarse en los parámetros 'HingeJoint' de cada rueda del sistema y seleccionar la opción 'device'; luego, introducir un nuevo elemento. Con esto, se despliega una nueva ventana que muestra los actuadores disponibles, en este caso, son del tipo 'RotationalMotor'. En la **Figura 132** se muestra una representación detallada de este paso.

Figura 132

***Figura 1**: Ventana auxiliar para agregar actuador rotativo tipo motor*

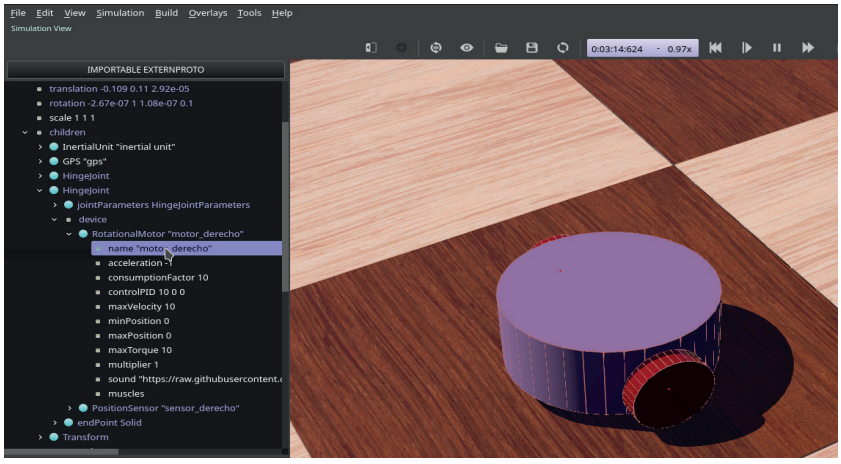


- Es preciso recordar que el paso previamente mencionado debe ser realizado tanto para la rueda izquierda como para la derecha. Una vez creados los elementos

'RotationalMotor', a estos son asignados nombres genéricos. Se recomienda modificar estas variables por 'motor_derecho' y 'motor_izquierdo', respectivamente, para cada una de las ruedas. En la **Figura 133** se muestra más información sobre este paso.

Figura 133

Propiedades del árbol de escena que muestran los nombres de los actuadores rotativos

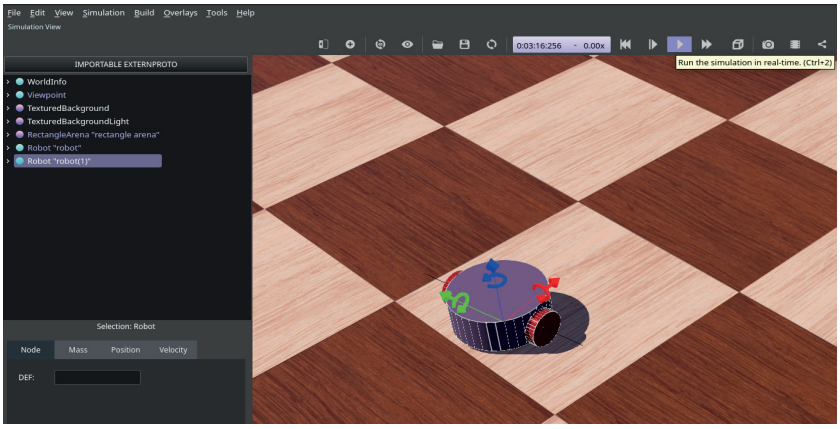


También es preciso tener en cuenta que el simulador Webots dispone de muchos más sensores; sin embargo, en este documento solo se utilizan los mencionados previamente. Una vez integrados estos elementos, el robot móvil es capaz de moverse al ser accionado y controlado por el usuario. Si el usuario desea visualizar la información de dichos sensores y acceder a los actuadores sin usar el lenguaje de programación, es posible hacerlo al seguir estos pasos:

- Ejecutar la simulación accionando el botón superior en la interfaz de usuario. En la **Figura 134** se muestra este procedimiento.

Figura 134

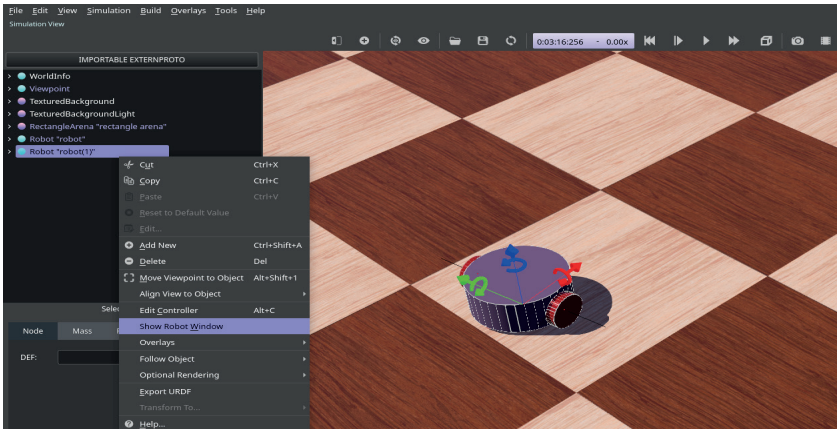
Ejecución del entorno del simulación en Webots



- Luego, acceder a la información de los sensores y actuadores a través de la función 'Show Robot Window'. Para acceder a este parámetro, el usuario debe dirigirse al árbol de escena y hacer clic en la pestaña 'robot' del robot móvil. Esto despliega una ventana donde se encuentra la opción 'Show Robot Window'. En la **Figura 135** se muestra todo lo descrito.

Figura 135

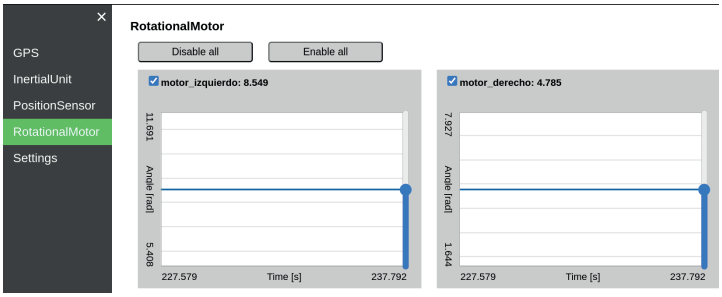
Ilustración que representa el acceso a la información de "Show Robot Window"



- Después, se despliega la información del robot en uno de los navegadores que se encuentra instalado en el computador. En la **Figura 136** se presenta cómo aparece lo explicado en este paso.

Figura 136

Interfaz de usuario donde se visualiza la información de los sensores y actuadores del sistema robótico



- En esta figura se observa cada uno de los sensores y actuadores agregados al sistema robótico móvil. Por ejemplo, se muestra el acceso a los motores 'motor_izquierdo' y 'motor_derecho'; esto permite al usuario modificar los valores y observar cómo se desplaza el sistema robótico.

Sin embargo, aún no se ha especificado cómo se accede a la información de los sensores dentro del lenguaje de programación Python; este contenido se aborda en la siguiente sección.

Para mayor información, está disponible el ambiente de simulación en el enlace https://bit.ly/3VQ6bZq bajo el nombre de "Robot_Movil". El usuario simplemente debe abrir el ambiente de simulación desde Webots.

4.3 Accionamiento del Robot y Acceso a la Información Sensorial

En la presente sección, se explican las funciones necesarias para acceder a la información de los sensores desde el lenguaje de programación Python. De igual forma, para accionar los motores acoplados a las ruedas, es necesario conocer y usar adecuadamente ciertas funciones dentro del entorno de Python.

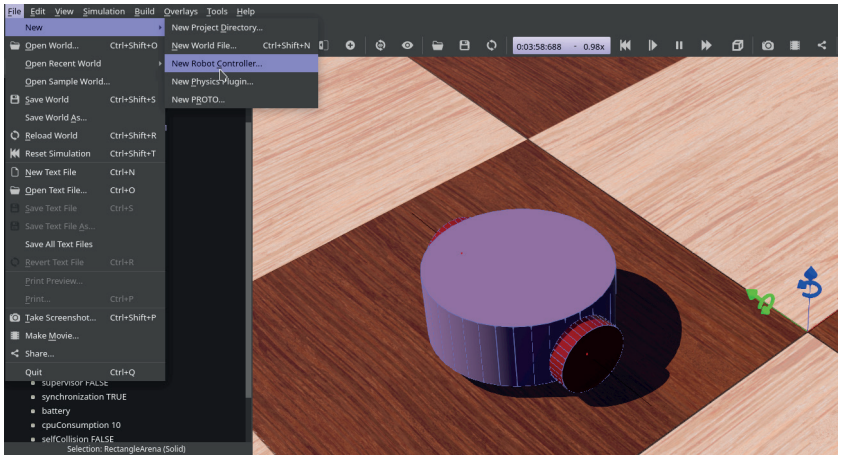
Es importante mencionar que el usuario debe crear un controlador y acoplarlo al sistema robótico; para esto, se siguen las siguientes instrucciones:

- Una vez creado el robot móvil y luego de incorporar en él los sensores y actuadores, se procede a crear un controlador. Para ello, el usuario debe dirigirse

al menú ‘File’ y después seleccionar la pestaña ‘New’. Con esto, se despliegan varias opciones, entre ellas se encuentra ‘New Robot Controller’. En la **Figura 137** se muestra este procedimiento.

Figura 137

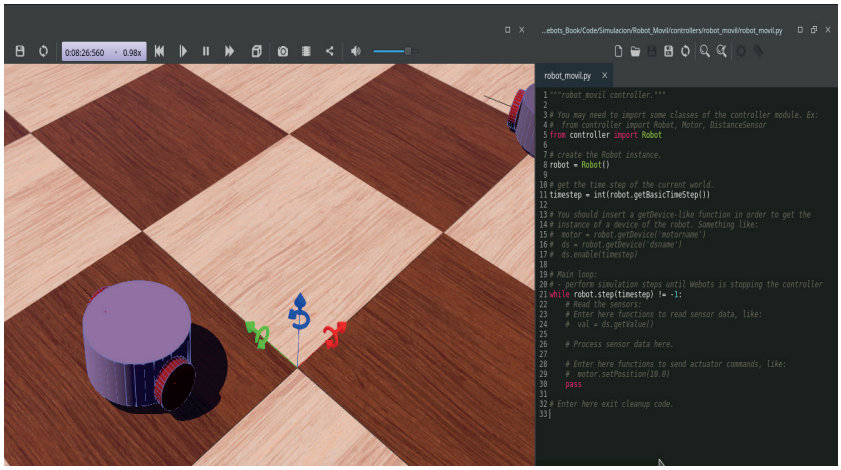
Creación de un nuevo controlador para el robot móvil



- Inmediatamente, aparece una nueva ventana donde se debe selecciona Python como el lenguaje de programación. Además, se tiene que asignar un nombre adecuado para el archivo, por ejemplo, ‘robot_movil’. Una vez completado este procedimiento, en el lado derecho de la interfaz de usuario se muestra el archivo creado. En la **Figura 138** se muestra de manera detallada este paso.

Figura 138

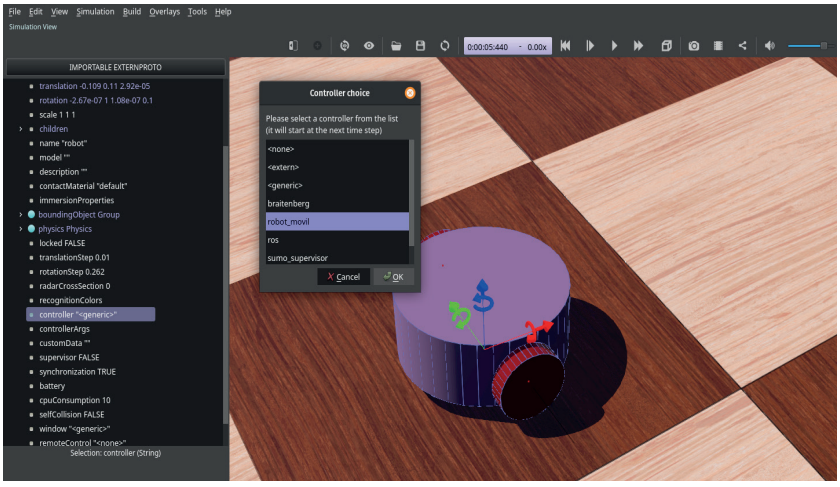
Ventana auxiliar donde se presenta la información creada por defecto en el controlador



- Cuando ya se ha creado el controlador respectivo, se lo asigna al robot móvil. Para ello, dentro de los parámetros de "robot", se debe seleccionar la pestaña "controller" para que el usuario pueda especificar el controlador asignado al robot. En la **Figura 139** se muestra más información sobre lo previamente descrito.

Figura 139

Asignación del controlador al sistema robótico móvil



- Al concluir este procedimiento, se pueden utilizar las funcionalidades de Python para acceder a la información de los sensores o controlar el robot móvil.

Accionando Motores

Para accionar los motores presentes en el robot móvil, como 'motor_izquierdo' y 'motor_derecho', se utilizan las clases y funciones de Webots compatibles con Python. A continuación, se mencionan las funciones que permiten tener acceso a los motores para accionarlos.

- **get_motor:** Permite al usuario especificar el nombre de los dispositivos que desea accionar. En este caso, se pueden incluir tanto el 'motor_izquierdo' como el 'motor_derecho.

- **set_motor:** Realiza la configuración necesaria para controlar la velocidad de cada uno de los motores.
- **set_motors_velocity:** Permite al usuario seleccionar la velocidad angular de rotación deseada para cada uno de los motores.
- **main:** Utiliza un bucle para enviar señales de control a los motores del robot a lo largo del tiempo. Se obtienen y establecen los motores izquierdo y derecho; luego, se espera un tiempo de muestreo antes de cada iteración del bucle.

Para obtener más información sobre el código desarrollado, se puede ingresar en el enlace https://bit.ly/3RZnAh9, donde se presenta con mayor detalle la implementación de dicho controlador. Finalmente, la simulación completa está disponible en https://bit.ly/3L9KmPJ, con el nombre de "robot_movil_sensores_actuadores.wbt".

Sensor GPS e IMU

Para acceder a la información del sensor GPS y la unidad de medición inercial IMU, es necesario desarrollar ciertas funciones. A continuación, se presenta más información estas:

- **set_gps:** Permite al usuario especificar el nombre del dispositivo GPS, en este caso, ha sido designado como 'gps'.
- **set_imu:** Permite al usuario especificar el nombre de la unidad de medición inercial que, en este caso, ha sido designado como 'inertial unit'.

- **get_states:** Permite al usuario acceder a la información de los sensores 'gps' e 'inertial unit'. Sin embargo, debido a que el robot móvil solo experimenta desplazamientos en el plano "xy" y rotación alrededor del eje "z", esta función solo presenta dicha información, donde las unidades son metros y radianes.

Para más información sobre la correcta implementación de estas funciones en el lenguaje de programación Python, se puede ingresar en el enlace https://bit.ly/3RZnAh9. Esta simulación está disponible en https://bit.ly/3L9KmPJ, con el nombre de "robot_movil_sensores_actuadores.wbt".

Sensor de desplazamiento angular

Al igual que en los sensores previos, para acceder a la información del sensor de desplazamiento angular, es necesario desarrollar ciertas funciones que brinden acceso a esta información. A continuación, se presenta más información sobre estas funciones:

- **get_sensor_pos:** Permite al usuario especificar el nombre del sensor de desplazamiento angular. En este ejemplo en particular, están definidos como 'sensor_izquierdo' o 'sensor_derecho'.
- **get_angular_pos:** Permite al usuario obtener información del sensor de posición angular. Estos datos se encuentran en radianes.
- **get_angular_pos:** Permite al usuario obtener información relacionada con la velocidad angular de cada una de las ruedas.

Para más información sobre la correcta implementación de estas funciones en el lenguaje de programación Python, se puede ingresar en el enlace https://bit.ly/3RZnAh9. Esta simulación está disponible en https://bit.ly/3L9KmPJ, con el nombre de "robot_movil_sensores_actuadores.wbt".

Con todo lo explicado en esta sección, es posible generar algoritmos de control, ya que se dispone de la información necesaria de los sensores, como la posición y la orientación del robot móvil. Además, es posible activar cada uno de los motores que están acoplados a las ruedas del sistema robótico.

5

Simulación de Robots Manipuladores

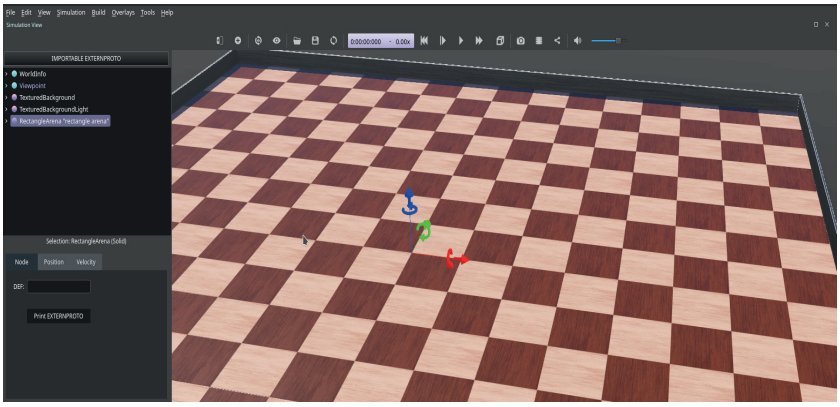
Para la simulación del robot manipulador, no se desarrolla paso a paso el proceso de armado. Sin embargo, se utiliza uno de los ejemplos ya definidos dentro del simulador, y se dan indicaciones sobre cómo acceder y controlar los sensores y actuadores que están incluidos por defecto en el sistema.

El robot manipulador con el que se realizan las explicaciones se llama 'Kuka youbot', el cual está montado sobre una plataforma móvil de tipo omnidireccional, en la que se acopla un manipulador de 5 grados de libertad. Tanto la plataforma como el manipulador pueden ser utilizados en conjunto o de forma independiente. Sin embargo, esta sección únicamente considera al manipulador como tema de interés.

Por lo tanto, para tener un mejor ambiente de simulación, se recomienda crear un nuevo directorio de trabajo bajo el nombre de 'Robot_Manipulador', dentro del cual se debe incluir el área de trabajo, también conocida como 'rectangle arena'; además, el ambiente de simulación debe tener un nombre, por ejemplo, de 'robot_manipulador.wbt'. En la **Figura 140** se muestra el resultado de la implementación de todos estos pasos.

Figura 140

Espacio de simulación vació donde se incluirá al robot manipulador "Kuka Youbot".

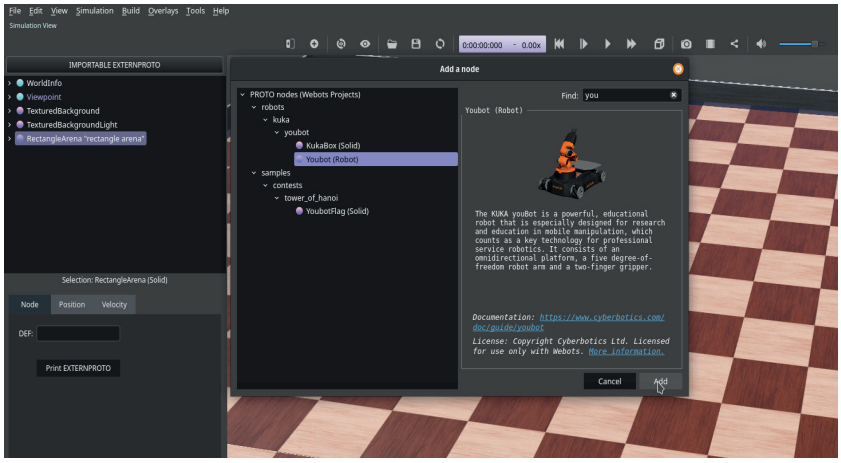


También, se debe recordar que es necesario dimensionar el área de trabajo 'rectangle arena', asignando valores de 5 metros tanto para el eje 'x' como para el 'y'.

Para agregar el robot 'Kuka Youbot', el usuario debe ubicarse en el árbol de escena; luego, se tiene que añadir un nuevo elemento donde se busca el robot previamente mencionado. En la **Figura 141** se muestra de manera detallada lo descrito.

Figura 141

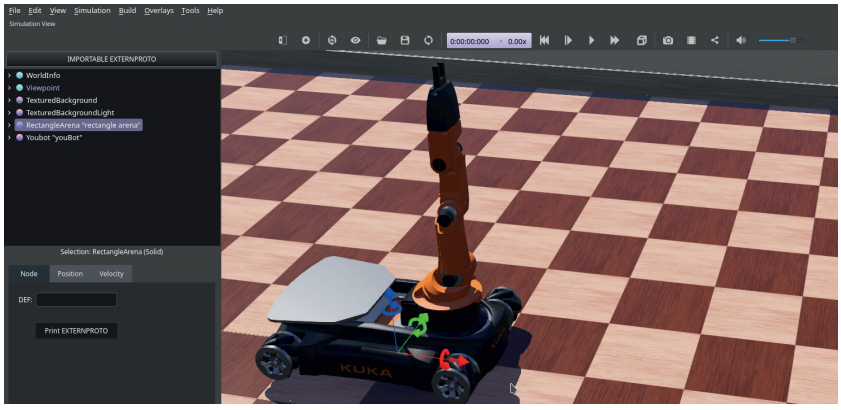
Ventana auxiliar para incluir al robot "Kuka Youbot" dentro del ambiente de simulación



Una vez completado este procedimiento, el robot se incluye dentro del ambiente de simulación. En la **Figura 142** se visualiza el robot 'Kuka Youbot'.

Figura 142

Robot "Kuka Youbot" dentro del entorno de simulación



Por defecto, el controlador asignado al robot 'Kuka Youbot' es el 'youbot'; esto permite al usuario maniobrar la plataforma desde el teclado. Sin embargo, para obtener un correcto funcionamiento de la plataforma dentro del ambiente de simulación, se deben hacer ciertas modificaciones en las propiedades de contacto. El usuario puede descargar y acceder directamente al ambiente de simulación correctamente configurado desde el enlace https://bit.ly/4cNUedB, con el nombre de "robot_manipulador.wbt".

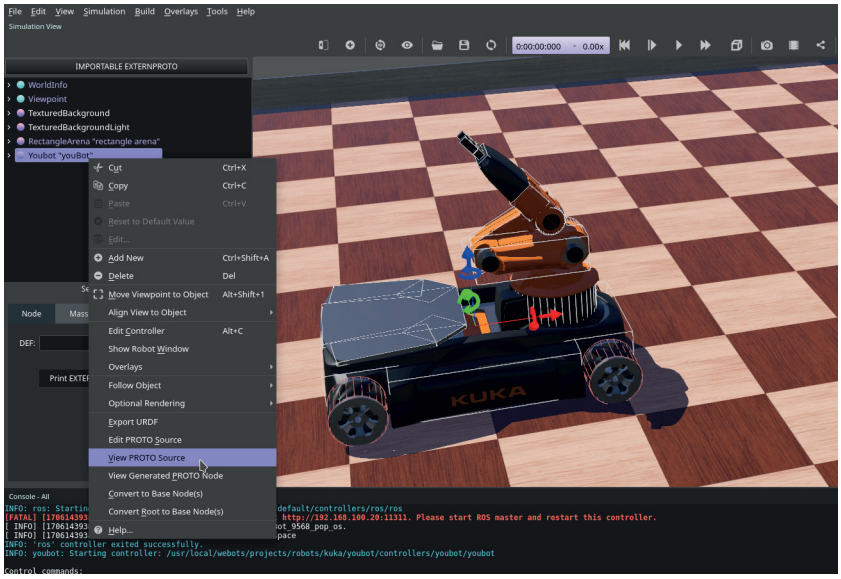
5.1 Sensores y Actuadores del Robot Manipulador

Para acceder a los sensores y actuadores del robot 'Kuka Youbot', es necesario conocer sus nombres. Sin embargo, por defecto, el usuario no tiene a disposición dicha información. Por lo tanto, para conocer dichos parámetros, se deben realizar las siguientes instrucciones:

El usuario debe colocarse sobre el robot dentro del ambiente 3D o, a su vez, dentro del árbol de escena y dar clic derecho. Así, se despliega un submenú en el que se encuentra la opción 'View PROTO Source'. En la **Figura 143** se muestra más detalladamente este procedimiento.

Figura 143

Ventana de acceso al "View PROTO Source" del robot "Kuka Youbot



- Una vez completado este procedimiento, en el lado derecho de la interfaz del usuario aparece un archivo de texto plano, conocido como 'Youbot.proto'. Este archivo contiene la información del sistema mecánico y sus respectivos sensores y actuadores. El usuario puede revisar este archivo y verificar cómo se definen las geometrías, sensores y actuadores. En la **Figura 144** se muestra una representación de este archivo.

Figura 144

Información previa dentro del archivo "Youbot.proto"

Este archivo de texto plano contiene la información de los sensores y en la anterior figura se puede apreciar cómo se llegan a definir los actuadores de tipo motor rotacional como "arm1" y su respectivo sensor como "arm1sensor".

Basada en la información de 'Youbot.proto', es posible definir los diferentes actuadores de rotación presentes en el manipulador como 'arm1', 'arm2', 'arm3', 'arm4' y, finalmente, 'arm5'. Adicionalmente, los sensores angulares se presentan como 'arm1sensor', 'arm2sensor', 'arm3sensor', 'arm4sensor' y 'arm5sensor'. Esta información se proporciona al usuario para luego acceder y controlar los sensores y motores, respectivamente, utilizando el lenguaje de programación Python. No se utilizan sensores adicionales, ya que esta sección se centra en el manipulador y no en la plataforma.

5.2 Acceder y Controlar los Sensores y Actuadores

Con base en la información de 'Youbot.proto', es posible definir los diversos sensores y actuadores presentes en el sistema para su uso en el lenguaje de programación Python. De esta manera, se procede a crear un controlador usando el lenguaje de programación Python, que se sugiere nombrar 'youbot_sensores_actuadores' y asignarlo adecuadamente al robot manipulador previamente definido. Una vez creado el controlador, es necesario definir algunas funciones que permitan al usuario especificar estos elementos. A continuación, se presenta una explicación detallada de cada función:

- **get_motor:** Permite al usuario definir los motores rotacionales que se utilizan dentro del manipulador robótico. Estos motores son 'arm1', 'arm2', 'arm3', 'arm4' y, finalmente, 'arm5'.
- **set_motor_pos:** Ayuda a especificar el desplazamiento angular deseado para cada motor del sistema robótico.
- **set_motor_velocity:** Asigna la velocidad angular para cada articulación presente en el manipulador robótico.
- **get_angular_position:** Permite al usuario obtener el desplazamiento angular en el que se encuentra cada articulación del sistema robótico.
- **init_system:** Mueve el robot manipulador a una posición inicial, apropiada para iniciar la fase de control del sistema.

Para obtener más información sobre el desarrollo del programa, se puede acceder al enlace https://bit.ly/4cNUedB,

donde se presenta cada una de las funciones mencionadas anteriormente. Una vez completada esta sección, el manipulador robótico 'Kuka Youbot' está listo para llevar a cabo estructuras de control avanzadas, que serán presentadas en la siguiente sección.

6

Control de Sistemas Robóticos

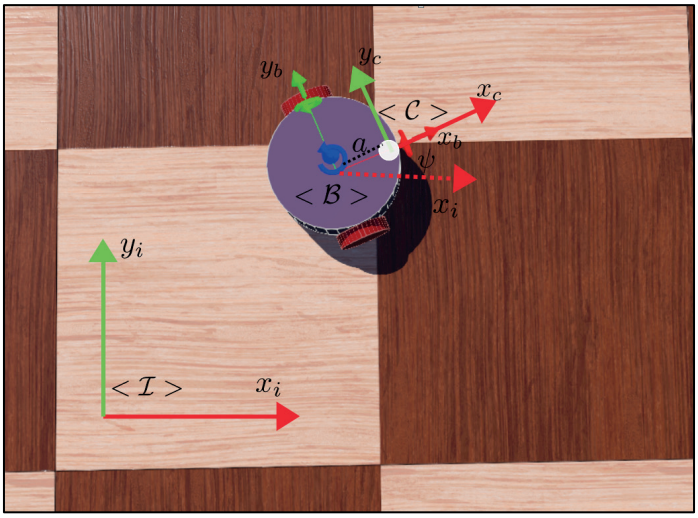
Para el diseño de estructuras de control, es necesario conocer cómo se comporta el sistema a controlar. A fin de entender su comportamiento, es posible utilizar tanto la cinemática diferencial como la dinámica del mismo. En esta sección, solo se va a usar la cinemática diferencial para el robot móvil tipo diferencial y para el manipulador robótico 'Kuka Youbot'. Una vez definido el modelo del sistema a emplear, se aplican los conceptos de control no lineal basados en la teoría de estabilidad de Lyapunov para formular una ley de control adecuada para cada sistema.

6.1 Formulación de las Leyes de Control para Robot Móvil

Con el propósito de generar una estructura de control adecuada, primero se debe definir qué modelo matemático del sistema se va a utilizar. Por lo tanto, mediante el uso de matrices de transformación homogénea, es posible describir la orientación y traslación de un robot móvil tipo diferencial (Recalde, Guevara, Cuzco, & Andaluz, 2020). Para ello, se establecen una serie de sistemas de referencia que nos servirán para desarrollar esta formulación. En la **Figura 145** se evidencia lo descrito.

Figura 145

Representación de un robot móvil tipo diferencial



Es posible formular la ubicación y orientación del sistema de referencia $\{\mathcal{C}\}$ en función de matrices de transformación homogénea, como se muestra a continuación:

$$ {}^{I}\mathbf{H}_{B} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & x \\ \sin(\psi) & \cos(\psi) & y \\ 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{B}\mathbf{H}_{C} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{I}\mathbf{H}_{C} = {}^{I}\mathbf{H}_{B}\ {}^{B}\mathbf{H}_{C} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & a\cos(\psi) + x \\ \sin(\psi) & \cos(\psi) & a\sin(\psi) + y \\ 0 & 0 & 1 \end{bmatrix} $$

De las formulaciones presentadas, se extrae la posición del sistema $\{\mathcal{C}\}$ respecto a $\{\mathcal{I}\}$, de la siguiente forma:

$$\begin{bmatrix} h_x \\ h_y \end{bmatrix} = \begin{bmatrix} a\cos(\psi) + x \\ a\sin(\psi) + y \end{bmatrix}$$

donde $\mathbf{h} = [h_x \quad h_y] \in \mathcal{R}^2$, representa la posición del punto de interés del robot móvil.

Bajo estas consideraciones, la cinemática diferencial puede ser escrita de la siguiente forma:

$$\begin{bmatrix} \dot{h}_x \\ \dot{h}_y \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -a\sin(\psi) \\ \sin(\psi) & a\cos(\psi) \end{bmatrix} \begin{bmatrix} \mu \\ \omega \end{bmatrix}$$

$$\dot{\mathbf{h}} = \mathbf{J}(\psi)\dot{\mathbf{q}}$$

donde $\dot{\mathbf{h}} = [\dot{h}_x \quad \dot{h}_y] \in \mathcal{R}^2$, es la velocidad del robot móvil respecto al sistema de referencia $\{\mathcal{I}\}$y $\mathbf{J}(\psi) \in \mathcal{R}^{2\times 2}$, y es la matriz Jacobiana, que permite realizar un mapeo lineal desde el espacio de control $\dot{\mathbf{q}} = [\mu \quad \omega] \in \mathcal{R}^2$ hacia (Martins, 2008).

En el diseño del controlador se emplean conceptos de control no lineal basados en la teoría de Lyapunov, donde se define una función candidata de tal manera que $V(\cdot) : \mathcal{R}^2 \to \mathcal{R}_{\geq 0}$. Esto garantiza la estabilidad del sistema bajo la influencia de la ley de control (Andaluz, V.,2014).

Con base en lo descrito, se propone una función candidata de Lyapunov asociada a los errores de control, la cual se define de la siguiente forma:

$$V(\mathbf{h}) = \tfrac{1}{2}\tilde{\mathbf{h}}^T\tilde{\mathbf{h}}$$

donde $\tilde{\mathbf{h}} = [\mathbf{h}_d - \mathbf{h}]^T$, es el vector de errores de control considerando que $\mathbf{h}_d = [h_{xd} \quad h_{yd}] \in \mathcal{R}^2$, es el vector de posiciones

deseadas a las que se desea ubicar el robot móvil.

Para definir una ley de control adecuada, se aplica la derivada temporal a la función candidata de Lyapunov previamente definida, lo que resulta en:

$$\dot{V}(\mathbf{h}) = \tilde{\mathbf{h}}^T \dot{\tilde{\mathbf{h}}}$$

de donde se tiene que $\dot{\tilde{\mathbf{h}}} = \dot{\mathbf{h}}_d - \mathbf{J}(\psi)\dot{\mathbf{q}}$ por lo que la derivada de la función candidata de Lyapunov se define como:

$$\dot{V}(\mathbf{h}) = \tilde{\mathbf{h}}^T(\dot{\mathbf{h}}_d - \mathbf{J}(\psi)\dot{\mathbf{q}})$$

al utilizar los conceptos de control no lineal, es posible generar una ley de control que permita definir la derivada de la función candidata de Lyapunov como negativa. Por lo tanto, se propone la siguiente ley de control:

$$\dot{\mathbf{q}}_c = \mathbf{J}^{-1}(\psi)(\dot{\mathbf{h}}_d + \mathbf{K}_2 \tanh(\mathbf{K}_2^{-1}\mathbf{K}_1\tilde{\mathbf{h}}))$$

donde $(\mathbf{K}_1, \mathbf{K}_2)$ son matrices definidas positivas. Finalmente, para verificar la estabilidad en lazo cerrado del sistema, se considera que existe un perfecto seguimiento de las acciones generadas por el controlador en el sistema robótico, tal que $\dot{\mathbf{q}} \equiv \dot{\mathbf{q}}_c$. Por consiguiente, la derivada temporal de la función de Lyapunov se define negativamente de la siguiente forma:

$$\dot{V}(\mathbf{h}) = -\tilde{\mathbf{h}}^T\mathbf{K}_2 \tanh(\mathbf{K}_2^{-1}\mathbf{K}_1\tilde{\mathbf{h}})$$

así, se define la estabilidad del sistema en lazo cerrado.

Además, es posible transformar las velocidades de control generadas en el marco de referencia $\{\mathcal{C}\}$ hacia las velocidades angulares de cada rueda. Esto se realiza mediante el siguiente mapeo lineal:

$$\begin{bmatrix} \mu \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{L} & -\frac{r}{L} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix}$$

donde es el radio que tienen las ruedas, es la distancia entre cada una de las ruedas y finalmente (ω_r, ω_l) son las velocidades angulares de la rueda derecha e izquierda, respectivamente.

6.2 Validación de las Leyes de Control de un Robot Móvil

Para verificar la efectividad de la estrategia de control, se utiliza el robot móvil desarrollado previamente. Sin embargo, se crea un nuevo entorno de simulación llamado 'robot_movil_control.wbt' y se aplica un controlador al robot móvil, con el nombre de 'controlador.py'.

Adicionalmente, es importante mencionar que los parámetros del robot móvil dentro del simulador robótico son los siguientes: el radio de las ruedas, $r = 0.02\ [m]$; la distancia entre cada rueda, $L = 0.11\ [m]$; y, finalmente, la distancia al punto de interés, $a = 0.055\ [m]$. Así también, las ganancias del controlador se definen por $\mathbf{K}_1 = \text{diag}(1, 1)$ y $\mathbf{K}_2 = \text{diag}(0.5, 0.5)$.

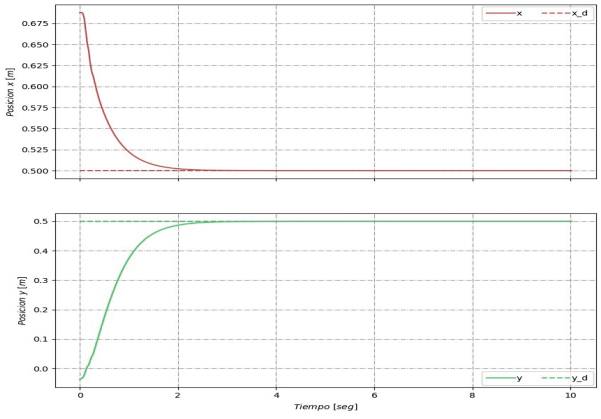
Después de establecer estas consideraciones previas, se definen las funciones dentro del archivo 'controlador.py' de la siguiente forma:

- **law_control:** Utiliza la información proporcionada por los sensores GPS e IMU, junto con la posición o trayectoria deseada, para calcular la acción de control correspondiente basada en la estructura de control propuesta.

- **transformation:** Permite transformar las acciones de control generadas por la ley de control en las velocidades angulares de cada rueda, para luego ser enviadas al robot móvil.
- **plot_results:** Permite al usuario visualizar el progreso del sistema durante el tiempo de simulación y verificar cómo la ley de control funciona correctamente.
- **plot_results_control:** Permite mostrar las acciones de control generadas por la estructura de control propuesta, proporcionando así un mejor entendimiento del sistema.
- Los resultados de la propuesta de control se presentan en la **Figura 146**, donde se puede notar cómo la ley de control garantiza que el robot móvil se dirija a los puntos deseados definidos por $\mathbf{h}_d = \begin{bmatrix} 1.0 & 0.0 \end{bmatrix}$.

Figura 146

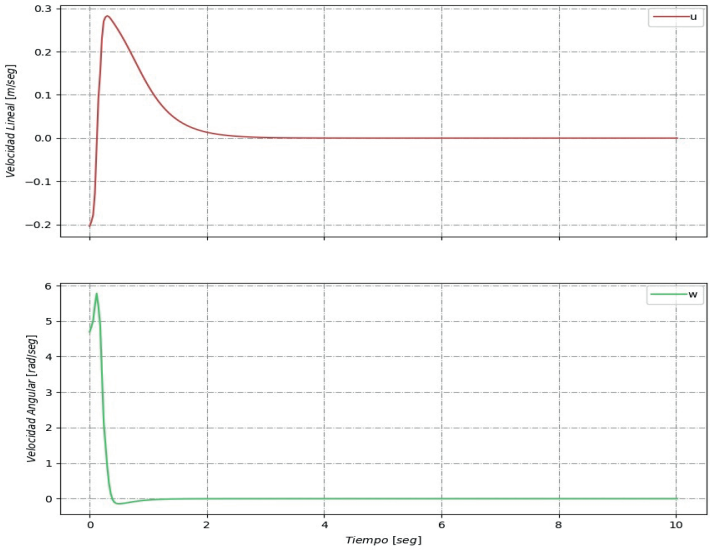
Posiciones deseadas y reales del sistema robótico móvil



Finalmente, las acciones de control generadas por la estructura de control propuesta. En la **Figura 147** se muestra lo descrito.

Figura 147

Acciones de control del sistema robótico móvil



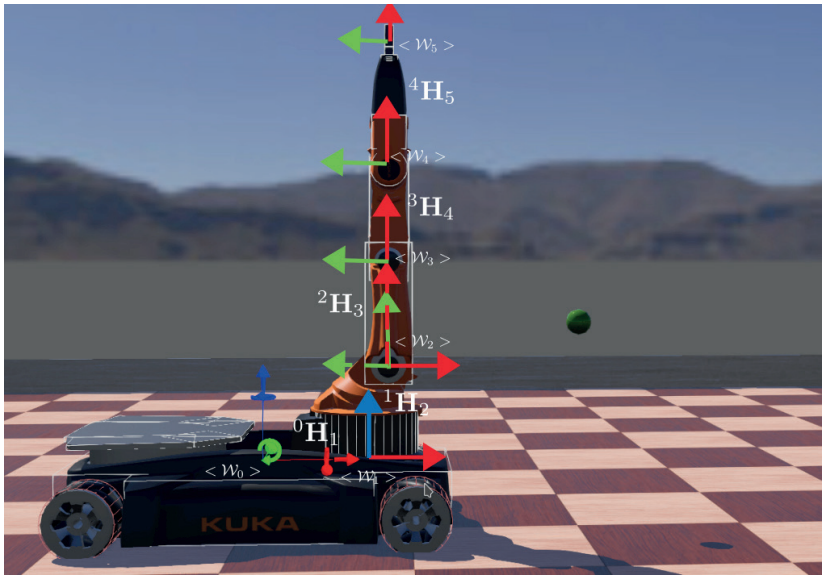
Para más información sobre la correcta implementación de estas funciones en el lenguaje de programación Python, se puede ingresar en el enlace https://bit.ly/3xyU7Uw. Esta simulación está disponible en https://bit.ly/3L9KmPJ, con el nombre de "robot_movil_control.wbt".

6.3 Formulación de las Leyes de Control para un Robot Manipulador

La formulación de la estructura de control para el manipulador robótico 'Kuka Youbot' está estrechamente relacionada con la cinemática directa de este sistema. Por lo tanto, en la **Figura 148** se presenta los elementos respectivos a la cinemática directa.

Figura 148

Representación del robot manipulador "Kuka Youbot" con sus respectivos sistemas de coordenadas



donde las matrices de transformación se encuentran definidas por:

$$
{}^{0}\mathbf{H}_{1} = \begin{bmatrix} 1 & 0 & 0 & a_0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{1}\mathbf{H}_{2} = \begin{bmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) & a_1\cos(\theta_1) \\ \sin(\theta_1) & 0 & -\cos(\theta_1) & a_1\sin(\theta_1) \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{2}\mathbf{H}_{3} = \begin{bmatrix} -\sin(\theta_2) & -\cos(\theta_2) & 0 & -a_2\sin(\theta_2) \\ \cos(\theta_2) & -\sin(\theta_2) & 0 & a_2\cos(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{3}\mathbf{H}_{4} = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & a_3\cos(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3) & 0 & a_3\sin(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{5}\mathbf{H}_{5} = \begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & a_4\cos(\theta_4) \\ \sin(\theta_4) & \cos(\theta_4) & 0 & a_4\sin(\theta_4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

con las matrices homogéneas previamente definidas, se puede encontrar la posición del efector final ubicado en el marco $\{\mathcal{W}_5\}$, representado por:

$$
\begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix} = \begin{bmatrix} a_0 + a_1\cos(\theta_1) - a_2\sin(\theta_2)\cos(\theta_1) - a_3\sin(\theta_2+\theta_3)\cos(\theta_1) - a_4\sin(\theta_2+\theta_3+\theta_4)\cos(\theta_1) \\ (a_1 - a_2\sin(\theta_2) - a_3\sin(\theta_2+\theta_3) - a_4\sin(\theta_2+\theta_3+\theta_4))\sin(\theta_1) \\ a_2\cos(\theta_2) + a_3\cos(\theta_2+\theta_3) + a_4\cos(\theta_2+\theta_3+\theta_4) + d_1 \end{bmatrix}
$$

donde $\mathbf{h} = [h_x \quad h_y \quad h_z] \in \mathcal{R}^3$s el vector de posición del efector final, $\mathbf{q} = [\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4] \in \mathcal{R}^4$ es el vector de estados internos del robot y, finalmente, $(a_0, a_1, d_1, a_2, a_3, a_4)$ son constantes geométricas del sistema.

Una vez establecida la cinemática directa, es posible encontrar la cinemática diferencial, que está definida por:

$$\dot{\mathbf{h}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$$

donde $\dot{\mathbf{h}} = [\dot{h}_x \quad \dot{h}_y \quad \dot{h}_z] \in \mathcal{R}^3$ es la velocidad del efector final $\{\mathcal{W}_5\}$ respecto al sistema $\{\mathcal{W}_0\}$. Por otro lado, $\mathbf{J}(\mathbf{q}) \in \mathcal{R}^{3\times 4}$ es una matriz Jacobiana que permite un mapeo lineal desde es espacio de control $\dot{\mathbf{q}} = [\dot{\theta}_1 \quad \dot{\theta}_2 \quad \dot{\theta}_3 \quad \dot{\theta}_4] \in \mathcal{R}^4$ hacia el espacio de operación $\dot{\mathbf{h}}$.

Al tener formulado el modelo del sistema, se puede diseñar el controlador mediante el uso de las técnicas de control no lineal y la teoría de estabilidad de Lyapunov, donde se define una función candidata $V(\cdot) : \mathcal{R}^3 \to \mathcal{R}_{\geq 0}$, la cual se encuentra representada de la siguiente forma:

$$V(\mathbf{h}) = \tfrac{1}{2}\tilde{\mathbf{h}}^T\tilde{\mathbf{h}}$$

donde $\tilde{\mathbf{h}} = \left[\mathbf{h}_d - \mathbf{h}\right]^T$ presenta el error de control, de esta manera, $\mathbf{h}_d = [h_{xd} \quad h_{yd} \quad h_{zd}] \in \mathcal{R}^3$ es el vector de posiciones deseadas para el efector final del manipulador robótico.

Después, al aplicar la derivada temporal a la función candidata de Lyapunov, se obtiene:

$$\dot{V}(\mathbf{h}) = \tilde{\mathbf{h}}^T\dot{\tilde{\mathbf{h}}}$$

al considerar que $\dot{\tilde{\mathbf{h}}} = \dot{\mathbf{h}}_d - \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$, la derivada temporal de la función candidata de Lyapunov se define de la siguiente forma:

$$\dot{V}(\mathbf{h}) = \tilde{\mathbf{h}}^T(\dot{\mathbf{h}}_d - \mathbf{J}(\mathbf{q})\dot{\mathbf{q}})$$

como se hizo con el robot móvil, se pueden aplicar los conceptos de control no lineal para generar una ley de control que permita definir la derivada de la función candidata de Lyapunov como negativa. Por lo tanto, se propone la siguiente ley de control:

$$\dot{\mathbf{q}}_c = \mathbf{J}^{-1}(\mathbf{q})(\dot{\mathbf{h}}_d + \mathbf{K}_2 \tanh(\mathbf{K}_2^{-1}\mathbf{K}_1\tilde{\mathbf{h}}))$$

$(\mathbf{K}_1, \mathbf{K}_2)$ son matrices positivas. Se considera que existe un perfecto seguimiento de las acciones generadas por el controlador en el sistema robótico, entonces, $\dot{\mathbf{q}} \equiv \dot{\mathbf{q}}_c$. Por lo tanto, la derivada temporal de la función de Lyapunov se define de manera negativa así:

$$\dot{V}(\mathbf{h}) = -\tilde{\mathbf{h}}^T \mathbf{K}_2 \tanh(\mathbf{K}_2^{-1}\mathbf{K}_1\tilde{\mathbf{h}})$$

con esto, es posible demostrar que el sistema en lazo cerrado bajo la influencia del controlador diseñado es estable.

Para más información de las leyes de control propuesta, se pueden revisar los estudios de (Varela, 2018) titulado *Modelling and control of a mobile manipulator for trajectory tracking*,(Andaluz, Molina, Erazo, & Ortiz, 2017) titutlado *Numerical Methods for Cooperative Control of Double Mobile Manipulators*;y y la conferencia de (Andaluz V. , 2011) "Coordinated cooperative control of mobile manipulators".

6.4 Validación de las Leyes de Control de un Robot Manipulador

Para verificar la efectividad de la estrategia de control, se utiliza el robot manipulador "Kuka Youbot" desarrollado en una sección anterior. Sin embargo, se crea un nuevo entorno de simulación llamado 'robot_manipulador_controlador.wbt' y se aplica un controlador al robot móvil, con el nombre de 'youbot_controller.py'.

Además, los parámetros utilizados dentro del controlador se definen de la siguiente forma:

$$a_0 = 0.156[m],\ a_1 = 0.033[m],\ d_1 = 0.147[m],\ a_2 = 0.155[m],\ a_3 = 0.135[m]$$

y $a_4 = 0.218[m]$. Además, las ganancias del controlador pueden ser definidas como $\mathbf{K}_1 = \text{diag}(1, 1)$ y $\mathbf{K}_2 = \text{diag}(0.5, 0.5)$.

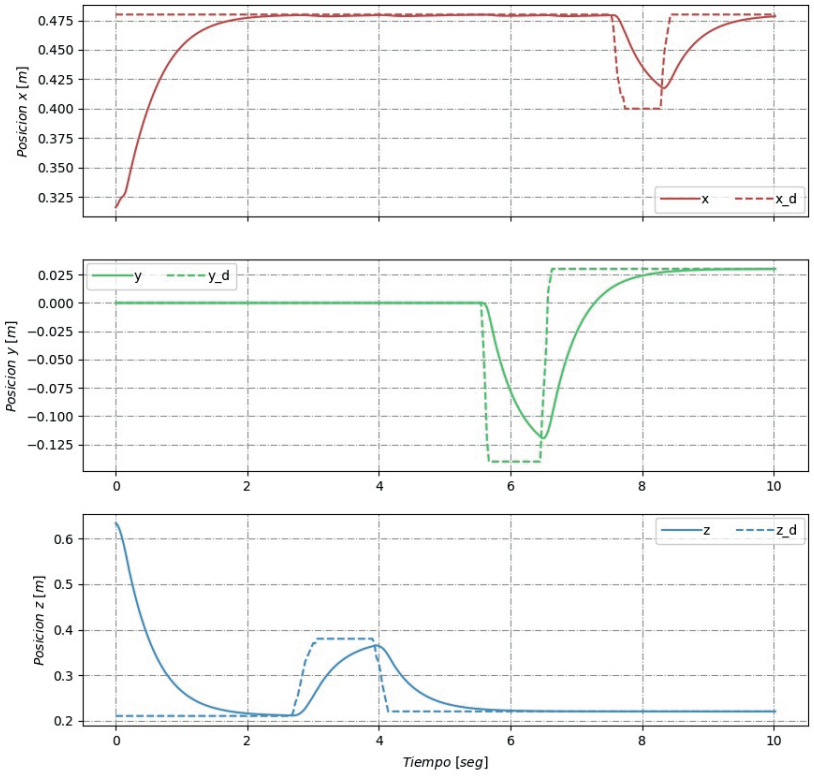
Según estas consideraciones, pueden definirse las funciones que permiten al manipulador robótico posicionar el efector final en la posición deseada.

- **forward_kinematics**: Utiliza la información proporcionada por los sensores de desplazamiento angular de cada uno de los motores del manipulador robótico, para así calcular la posición del efector final.
- **law_control**: Permite al usuario definir el punto deseado y, mediante la cinemática directa y la estructura de control formulada, es capaz de calcular las velocidades angulares necesarias para cada articulación y así cumplir con la tarea deseada.

Finalmente, en la **Figura 149** se muestran los resultados de la estructura de control propuesta.

Figura 149

Resultados del controlador (se visualiza cómo el efector final es capaz de situarse en la posición deseada)

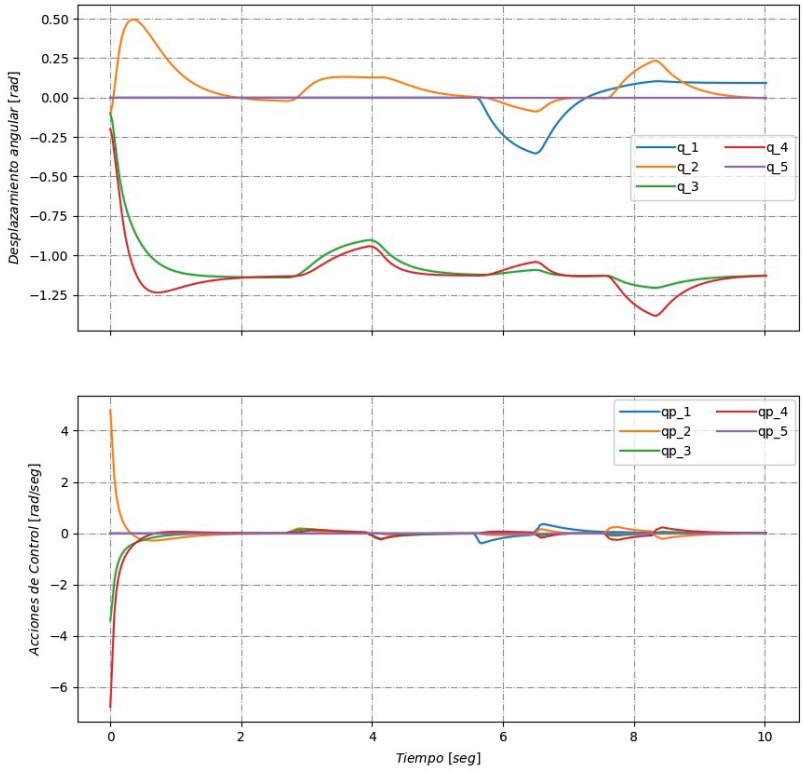


donde $\mathbf{h}_d = [h_{xd} \quad h_{yd} \quad h_{zd}]^T$ es el vector de posiciones deseadas que el usuario puede variar a lo largo del tiempo. En la figura anterior, se observa que la estructura de control es capaz de hacer un perfecto seguimiento de las posiciones deseadas por el usuario, demostrando la efectividad de la propuesta de control.

En la **Figura 150** se presentan las acciones de control generadas.

Figura 150

Acciones de control generadas para el robot "Kuka Youbot"



Para más información sobre la correcta implementación de estas funciones en el lenguaje de programación Python, se puede ingresar en el enlace https://bit.ly/4cqOQgP. Esta simulación está disponible en https://bit.ly/4cNUedB, con el nombre de "robot_manipulador_controlador.wbt".

Referencias

Andaluz, V. (2011). Coordinated cooperative control of mobile manipulators. Conferencia Internacional IEEE 2011 sobre Tecnología Industrial.

Andaluz, V. (2012). Adaptive unified motion control of mobile manipulators. *Práctica de ingeniería de control*, 1337-1352.

Andaluz, V. (2014). Robust Control with Dynamic Compensation for Human-Wheelchair System. *Springer*. doi:https://doi.org/10.1007/978-3-319-13966-1_37

Andaluz, V. (2016). Unity3D-MatLab Simulator in Real Time for Robotics Applications. *Springer*. doi:https://doi.org/10.1007/978-3-319-40621-3_19

Andaluz, V., Molina, M., Erazo, Y., & Ortiz, J. (2017). Numerical Methods for Cooperative Control of Double Mobile Manipulators. *Springer*. doi:https://doi.org/10.1007/978-3-319-65292-4_77

Corke. (2017). Robotics, Vision and Control. *Springer*. doi:https://doi.org/10.1007/978-3-319-54413-7

Grafarend, E., & Kühnel, W. (2011). A minimal atlas for the rotation group SO(3). *Springer*, 113-122. doi:https://doi.org/10.1007/s13137-011-0018-x

Griffiths, D. F., & Higham, D. J. (2010). Euler's Method. *Springer*. doi:https://doi.org/10.1007/978-0-85729-148-6_2

Ixaru, L. G., & Vanden Berghe, G. (2004). Runge-Kutta Solvers for Ordinary Differential Equations. *Exponential fitting*, 223-304. doi:https://doi.org/10.1007/978-1-4020-2100-8_6

Koenig, N., & Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. 2149–2154. doi:https://ieeexplore.ieee.org/document/1389727

Lang, S. (1987). Vector Spaces. In: Linear Algebra. Undergraduate Texts in Mathematics. *Springer*, 24-43. doi:https://doi.org/10.1007/978-1-4757-1949-9_1

Martins, F. (2008). An adaptive dynamic controller for autonomous mobile robot trajectory tracking. *Práctica de ingeniería de control*, 1354-1363. doi:https://doi.org/10.1016/j.conengprac.2008.03.004

Michel, O. (2004). Professional mobile robot simulation. 39–42. doi:https://journals.sagepub.com/doi/10.5772/5618

Recalde, L., Guevara, B., Cuzco, G., & Andaluz, V. (2020). Optimal Control Problem of a Differential Drive Robot. *Springer*. doi: https://doi.org/10.1007/978-3-030-55789-8_7

Román-Ibáñez, V. (2018). A Low-Cost Immersive Virtual Reality System for Teaching Robotic Manipulators Programming. doi:https://doi.org/10.3390/SU10041102

Siciliano. (2009). Springer handbook of robotics. *Springer*.

Siciliano, B. (2016). Robotics. *Springer*. doi:https://ieeexplore.ieee.org/document/6386109

Varela, J. (2018). Modelling and control of a mobile manipulator for trajectory tracking. *Conferencia INCISCOS*.

Waldron, K., & Schmiedeler, J. (2008). Kinematics. *Springer*. doi:https://doi.org/10.1007/978-3-540-30301-5_2

ANEXOS

Robot_Movil

February 14, 2024

1 Anexo 1

1.1 Importar Librerias

[1]:
```
import numpy as np
from sympy import*
init_printing()
```

1.1.1 Declarar variables simbolicas

[2]:
```
## Variables Simbolicas
psi = symbols("psi",  real=True)
x = symbols("x",  real=True)
y = symbols("y",  real=True)
a = symbols("a",  real=True)
```

1.1.2 Matriz rotacion de B respecto a I

[3]:
```
iRb = Matrix([[cos(psi), -sin(psi)], [sin(psi), cos(psi)]])
```

[4]:
```
iRb
```

[4]:
$$\begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix}$$

1.1.3 Desplazamiento de B respecto a I

[5]:
```
desplazamiento_ib  = Matrix([[x], [y]])
```

[6]:
```
desplazamiento_ib
```

[6]:
$$\begin{bmatrix} x \\ y \end{bmatrix}$$

1.1.4 Matriz homogenea de B respecto a V

```
iHb = BlockMatrix([[iRb, desplazamiento_ib], [zeros(1,2), ones(1,1)]]).
↪as_explicit()
```

```
iHb
```

$$\begin{bmatrix} \cos(\psi) & -\sin(\psi) & x \\ \sin(\psi) & \cos(\psi) & y \\ 0 & 0 & 1 \end{bmatrix}$$

1.1.5 Matriz rotacion de C respecto a B

```
bRc = eye(2, 2)
```

```
bRc
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

1.1.6 Desplazamiento de C respecto a B

```
desplazamiento_bc  = Matrix([[a], [0]])
```

```
desplazamiento_bc
```

$$\begin{bmatrix} a \\ 0 \end{bmatrix}$$

1.1.7 Matriz homogenea de B respecto a V

```
bHc = BlockMatrix([[bRc, desplazamiento_bc], [zeros(1,2), ones(1,1)]]).
↪as_explicit()
```

```
bHc
```

$$\begin{bmatrix} 1 & 0 & a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

1.1.8 Posicion y Orientacion del Frame C

```
iHc = simplify(iHb*bHc)
```

```
iHc
```

$$\begin{bmatrix} \cos(\psi) & -\sin(\psi) & a\cos(\psi) + x \\ \sin(\psi) & \cos(\psi) & a\sin(\psi) + y \\ 0 & 0 & 1 \end{bmatrix}$$

Orientacion_3D

February 14, 2024

1 Anexo 2

1.1 Importar Librerias

[1]:
```
import numpy as np
from sympy import*
init_printing()
```

1.1.1 Declarar variables simbolicas

[2]:
```
## Variables Simbolicas
theta = symbols("theta",  real=True)
theta
```

[2]: θ

1.1.2 Rotacion eje x

[3]:
```
R_x = Matrix([[1, 0, 0], [0, cos(theta), -sin(theta)], [0, sin(theta),␣
↪cos(theta)]])
```

[4]:
```
R_x
```

[4]: $$\begin{bmatrix}1 & 0 & 0\\0 & \cos{\left(\theta \right)} & - \sin{\left(\theta \right)}\\0 & \sin{\left(\theta \right)} & \cos{\left(\theta \right)}\end{bmatrix}$$

1.1.3 Rotacion eje y

[5]:
```
R_y = Matrix([[cos(theta), 0, sin(theta)], [0, 1, 0], [-sin(theta), 0,␣
↪cos(theta)]])
R_y
```

[5]: $$\begin{bmatrix}\cos{\left(\theta \right)} & 0 & \sin{\left(\theta \right)}\\0 & 1 & 0\\- \sin{\left(\theta \right)} & 0 & \cos{\left(\theta \right)}\end{bmatrix}$$

1.1.4 Rotacion eje z

[6]:
```
R_z = Matrix([[cos(theta), -sin(theta), 0], [sin(theta), cos(theta), 0], [0, 0,␣
↪1]])
R_z
```

[6]: $$\begin{bmatrix}\cos{\left(\theta \right)} & - \sin{\left(\theta \right)} & 0\\\sin{\left(\theta \right)} & \cos{\left(\theta \right)} & 0\\0 & 0 & 1\end{bmatrix}$$

[ ]:

Matrices_Homogeneas

February 14, 2024

1 Anexo 3

1.1 Importar Librerias

[1]:
```
import numpy as np
from sympy import*
init_printing()
```

1.1.1 Declarar variables simbolicas

[2]:
```
## Variables Simbolicas
theta = symbols("theta",  real=True)
phi = symbols("phi",  real=True)
psi = symbols("psi",  real=True)
```

1.1.2 Rotacion eje x

[3]:
```
H_x = Matrix([[1, 0, 0, 0], [0, cos(phi), -sin(phi), 0], [0, sin(phi),␣
↪cos(phi), 0], [0, 0, 0, 1]])
```

[4]:
```
H_x
```

[4]:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[ ]:
```

```

1.1.3 Rotacion eje y

[5]:
```
H_y = Matrix([[cos(theta), 0, sin(theta), 0], [0, 1, 0, 0], [-sin(theta), 0,␣
↪cos(theta), 0], [0, 0, 0, 1]])
H_y
```

[5]:

$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.1.4 Rotacion eje z

[6]:
```
H_z = Matrix([[cos(psi), -sin(psi), 0, 0], [sin(psi), cos(psi), 0, 0], [0, 0,␣
↪1, 0], [0, 0, 0, 1]])
H_z
```

[6]:
$$\begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 & 0 \\ \sin(\psi) & \cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.1.5 Composicion de Matrices Homogeneas

[7]:
```
H = simplify((H_z@H_y@H_x))
H
```

[7]:
$$\begin{bmatrix} \cos(\psi)\cos(\theta) & \sin(\phi)\sin(\theta)\cos(\psi) - \sin(\psi)\cos(\phi) & \sin(\phi)\sin(\psi) + \sin(\theta)\cos(\phi)\cos(\psi) & 0 \\ \sin(\psi)\cos(\theta) & \sin(\phi)\sin(\psi)\sin(\theta) + \cos(\phi)\cos(\psi) & -\sin(\phi)\cos(\psi) + \sin(\psi)\sin(\theta)\cos(\phi) & 0 \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[ ]:

[8]:
```
H = simplify((H_x@H_y@H_z))
H
```

[8]:
$$\begin{bmatrix} \cos(\psi)\cos(\theta) & -\sin(\psi)\cos(\theta) & \sin(\theta) & 0 \\ \sin(\phi)\sin(\theta)\cos(\psi) + \sin(\psi)\cos(\phi) & -\sin(\phi)\sin(\psi)\sin(\theta) + \cos(\phi)\cos(\psi) & -\sin(\phi)\cos(\theta) & 0 \\ \sin(\phi)\sin(\psi) - \sin(\theta)\cos(\phi)\cos(\psi) & \sin(\phi)\cos(\psi) + \sin(\psi)\sin(\theta)\cos(\phi) & \cos(\phi)\cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[ ]:

[ ]:

Planar_robot

February 14, 2024

1 Anexo 4

1.1 Importar librerias

[1]:
```
import numpy as np
from sympy import*
init_printing()
```

[2]:
```
## Variables Simbolicas
theta_1 = symbols("theta_1",  real=True)
theta_2 = symbols("theta_2",  real=True)
l_1 = symbols("l_1",  real=True)
l_2 = symbols("l_2",  real=True)
```

[3]:
```
_0H_1 = Matrix([[cos(theta_1), -sin(theta_1), 0, 0], [sin(theta_1),␣
↪cos(theta_1), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[4]:
```
_0H_1
```

[4]:
$$\begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[5]:
```
_1H2 = Matrix([[1, 0, 0, l_1], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[6]:
```
_1H2
```

[6]:
$$\begin{bmatrix} 1 & 0 & 0 & l_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[7]:
```
_2H_3 = Matrix([[cos(theta_2), -sin(theta_2), 0, 0], [sin(theta_2),␣
↪cos(theta_2), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[8]:
```
_2H_3
```

[8]:

$$\begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[9]:
```
_3H4 = Matrix([[1, 0, 0, l_2], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[10]:
```
_3H4
```

[10]:
$$\begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[11]:
```
H = simplify(_OH_1@_1H2@_2H_3@_3H4)
```

[12]:
```
H
```

[12]:
$$\begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & l_1\cos(\theta_1)+l_2\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & l_1\sin(\theta_1)+l_2\sin(\theta_1+\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[ ]:

[13]:
```
P = H[0:2, 3]
```

[14]:
```
P
```

[14]:
$$\begin{bmatrix} l_1\cos(\theta_1)+l_2\cos(\theta_1+\theta_2) \\ l_1\sin(\theta_1)+l_2\sin(\theta_1+\theta_2) \end{bmatrix}$$

[ ]:

[ ]:

Robot_3D

February 14, 2024

1 Anexo 5

[ ]:

1.1 Importar Librerias

[1]:
```
import numpy as np
from sympy import*
init_printing()
```

[2]:
```
## Variables Simbolicas
theta_1 = symbols("theta_1",  real=True)
theta_2 = symbols("theta_2",  real=True)
theta_3 = symbols("theta_3",  real=True)
alpha = symbols("alpha",  real=True)

l_1 = symbols("l_1",  real=True)
l_2 = symbols("l_2",  real=True)
l_3 = symbols("l_3",  real=True)
```

[3]:
```
_0H_1 = Matrix([[cos(theta_1), -sin(theta_1), 0, 0], [sin(theta_1),␣
↪cos(theta_1), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[4]:
```
_0H_1
```

[4]:
$$\begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[5]:
```
_1H2 = Matrix([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, l_1], [0, 0, 0, 1]])
```

[6]:
```
_1H2
```

[6]:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[7]:
```
_2H3 = Matrix([[1, 0, 0, 0], [0, cos(alpha), -sin(alpha), 0], [0, sin(alpha),␣
↪cos(alpha), 0], [0, 0, 0, 1]])
```

[8]:
```
_2H3
```

[8]:
$$\begin{bmatrix}1 & 0 & 0 & 0\\0 & \cos(\alpha) & -\sin(\alpha) & 0\\0 & \sin(\alpha) & \cos(\alpha) & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[9]:
```
_2H3 = _2H3.subs(alpha, pi/2)
```

[10]:
```
_2H3
```

[10]:
$$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 0 & -1 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[11]:
```
_3H4 = Matrix([[cos(theta_2), -sin(theta_2), 0, 0], [sin(theta_2),␣
↪cos(theta_2), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[12]:
```
_3H4
```

[12]:
$$\begin{bmatrix}\cos(\theta_2) & -\sin(\theta_2) & 0 & 0\\\sin(\theta_2) & \cos(\theta_2) & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[13]:
```
_4H5 = Matrix([[1, 0, 0, l_2], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[14]:
```
_4H5
```

[14]:
$$\begin{bmatrix}1 & 0 & 0 & l_2\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[15]:
```
_5H6 = Matrix([[cos(theta_3), -sin(theta_3), 0, 0], [sin(theta_3),␣
↪cos(theta_3), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[16]:
```
_5H6
```

[16]:
$$\begin{bmatrix}\cos(\theta_3) & -\sin(\theta_3) & 0 & 0\\\sin(\theta_3) & \cos(\theta_3) & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[17]:
```
_6H7 = Matrix([[1, 0, 0, l_3], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[18]:
```
H = simplify(_0H_1@_1H2@_2H3@_3H4@_4H5@_5H6@_6H7)
```

[19]:
```
H
```

[19]:
$$\begin{bmatrix} \cos(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3)\cos(\theta_1) & \sin(\theta_1) & (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\cos(\theta_1) \\ \sin(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_1)\sin(\theta_2+\theta_3) & -\cos(\theta_1) & (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\sin(\theta_1) \\ \sin(\theta_2+\theta_3) & \cos(\theta_2+\theta_3) & 0 & l_1+l_2\sin(\theta_2)+l_3\sin(\theta_2+\theta_3) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[ ]:
```

```

[20]:
```
P = H[0:3, 3]
```

[21]:
```
P
```

[21]:
$$\begin{bmatrix} (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\cos(\theta_1) \\ (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\sin(\theta_1) \\ l_1+l_2\sin(\theta_2)+l_3\sin(\theta_2+\theta_3) \end{bmatrix}$$

[ ]:
```

```

[22]:
```
R = H[0:3, 0:3]
```

[23]:
```
R
```

[23]:
$$\begin{bmatrix} \cos(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3)\cos(\theta_1) & \sin(\theta_1) \\ \sin(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_1)\sin(\theta_2+\theta_3) & -\cos(\theta_1) \\ \sin(\theta_2+\theta_3) & \cos(\theta_2+\theta_3) & 0 \end{bmatrix}$$

[ ]:
```

```

[ ]:
```

```

February 14, 2024

1 Anexo 6

1.1 Importar Librerias

[1]:
```
import numpy as np
from sympy import*
init_printing()
```

1.1.1 Declarar variables simbolicas

[2]:
```
## Variables Simbolicas
psi = symbols("psi",  real=True)
x = symbols("x",  real=True)
y = symbols("y",  real=True)
a = symbols("a",  real=True)
mu = symbols("mu", real = True)
omega = symbols("omega", real = True)
```

1.1.2 Matriz rotacion de B respecto a I

[3]:
```
iRb = Matrix([[cos(psi), -sin(psi)], [sin(psi), cos(psi)]])
```

[4]:
```
iRb
```

[4]:
$$\begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix}$$

1.1.3 Desplazamiento de B respecto a I

[5]:
```
desplazamiento_ib  = Matrix([[x], [y]])
```

[6]:
```
desplazamiento_ib
```

[6]:
$$\begin{bmatrix} x \\ y \end{bmatrix}$$

1.1.4 Matriz homogenea de B respecto a V

[7]:
```
iHb = BlockMatrix([[iRb, desplazamiento_ib], [zeros(1,2), ones(1,1)]]).
↪as_explicit()
```

[8]:
```
iHb
```

[8]:
$$\begin{bmatrix} \cos(\psi) & -\sin(\psi) & x \\ \sin(\psi) & \cos(\psi) & y \\ 0 & 0 & 1 \end{bmatrix}$$

1.1.5 Matriz rotacion de C respecto a B

[9]:
```
bRc = eye(2, 2)
```

[10]:
```
bRc
```

[10]:
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

1.1.6 Desplazamiento de C respecto a B

[11]:
```
desplazamiento_bc  = Matrix([[a], [0]])
```

[12]:
```
desplazamiento_bc
```

[12]:
$$\begin{bmatrix} a \\ 0 \end{bmatrix}$$

1.1.7 Matriz homogenea de B respecto a V

[13]:
```
bHc = BlockMatrix([[bRc, desplazamiento_bc], [zeros(1,2), ones(1,1)]]).
↪as_explicit()
```

[14]:
```
bHc
```

[14]:
$$\begin{bmatrix} 1 & 0 & a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

1.1.8 Posicion y Orientacion del Frame C

[15]:
```
iHc = simplify(iHb*bHc)
```

[16]:
```
iHc
```

[16]:
$$\begin{bmatrix} \cos(\psi) & -\sin(\psi) & a\cos(\psi) + x \\ \sin(\psi) & \cos(\psi) & a\sin(\psi) + y \\ 0 & 0 & 1 \end{bmatrix}$$

[17]:
```
P = iHc[0:2, 2]
```

[18]:
```
P
```

[18]:
$$\begin{bmatrix} a\cos(\psi) + x \\ a\sin(\psi) + y \end{bmatrix}$$

[19]:
```
q = Matrix([[x], [y], [psi]])
```

[20]:
```
q
```

[20]:
$$\begin{bmatrix} x \\ y \\ \psi \end{bmatrix}$$

[21]:
```
dp = P.jacobian([x, y, psi])
```

[22]:
```
dp
```

[22]:
$$\begin{bmatrix} 1 & 0 & -a\sin(\psi) \\ 0 & 1 & a\cos(\psi) \end{bmatrix}$$

[23]:
```
dp.shape
```

[23]:
$$(2,\ 3)$$

[24]:
```
q_dot = Matrix([[mu*cos(psi)], [mu*sin(psi)], [omega]])
```

[25]:
```
q_dot
```

[25]:
$$\begin{bmatrix} \mu\cos(\psi) \\ \mu\sin(\psi) \\ \omega \end{bmatrix}$$

February 14, 2024

1 Anexo 6

1.1 Importar Librerias

[1]:
```
import numpy as np
from sympy import*
init_printing()
```

[2]:
```
## Variables Simbolicas
theta_1 = symbols("theta_1",  real=True)
theta_2 = symbols("theta_2",  real=True)
theta_3 = symbols("theta_3",  real=True)
alpha = symbols("alpha",  real=True)

l_1 = symbols("l_1",  real=True)
l_2 = symbols("l_2",  real=True)
l_3 = symbols("l_3",  real=True)
```

[3]:
```
_0H_1 = Matrix([[cos(theta_1), -sin(theta_1), 0, 0], [sin(theta_1),␣
↪cos(theta_1), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[4]:
```
_0H_1
```

[4]:
$$\begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[5]:
```
_1H2 = Matrix([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, l_1], [0, 0, 0, 1]])
```

[6]:
```
_1H2
```

[6]:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[7]:
```
_2H3 = Matrix([[1, 0, 0, 0], [0, cos(alpha), -sin(alpha), 0], [0, sin(alpha),␣
↪cos(alpha), 0], [0, 0, 0, 1]])
```

[8]:
```
_2H3
```

[8]:
$$\begin{bmatrix}1 & 0 & 0 & 0\\0 & \cos{(\alpha)} & -\sin{(\alpha)} & 0\\0 & \sin{(\alpha)} & \cos{(\alpha)} & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[9]:
```
_2H3 = _2H3.subs(alpha, pi/2)
```

[10]:
```
_2H3
```

[10]:
$$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 0 & -1 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[11]:
```
_3H4 = Matrix([[cos(theta_2), -sin(theta_2), 0, 0], [sin(theta_2),␣
↪cos(theta_2), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[12]:
```
_3H4
```

[12]:
$$\begin{bmatrix}\cos{(\theta_2)} & -\sin{(\theta_2)} & 0 & 0\\\sin{(\theta_2)} & \cos{(\theta_2)} & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[13]:
```
_4H5 = Matrix([[1, 0, 0, l_2], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[14]:
```
_4H5
```

[14]:
$$\begin{bmatrix}1 & 0 & 0 & l_{2}\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[15]:
```
_5H6 = Matrix([[cos(theta_3), -sin(theta_3), 0, 0], [sin(theta_3),␣
↪cos(theta_3), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[16]:
```
_5H6
```

[16]:
$$\begin{bmatrix}\cos{(\theta_3)} & -\sin{(\theta_3)} & 0 & 0\\\sin{(\theta_3)} & \cos{(\theta_3)} & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[17]:
```
_6H7 = Matrix([[1, 0, 0, l_3], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
```

[18]:
```
H = simplify(_0H_1@_1H2@_2H3@_3H4@_4H5@_5H6@_6H7)
```

[19]:
```
H
```

[19]:
$$\begin{bmatrix} \cos(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3)\cos(\theta_1) & \sin(\theta_1) & (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\cos(\theta_1) \\ \sin(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_1)\sin(\theta_2+\theta_3) & -\cos(\theta_1) & (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\sin(\theta_1) \\ \sin(\theta_2+\theta_3) & \cos(\theta_2+\theta_3) & 0 & l_1+l_2\sin(\theta_2)+l_3\sin(\theta_2+\theta_3) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[ ]:
```

```

[20]:
```
P = H[0:3, 3]
```

[21]:
```
P
```

[21]:
$$\begin{bmatrix} (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\cos(\theta_1) \\ (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\sin(\theta_1) \\ l_1+l_2\sin(\theta_2)+l_3\sin(\theta_2+\theta_3) \end{bmatrix}$$

[ ]:
```

```

[22]:
```
R = H[0:3, 0:3]
```

[23]:
```
R
```

[23]:
$$\begin{bmatrix} \cos(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3)\cos(\theta_1) & \sin(\theta_1) \\ \sin(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_1)\sin(\theta_2+\theta_3) & -\cos(\theta_1) \\ \sin(\theta_2+\theta_3) & \cos(\theta_2+\theta_3) & 0 \end{bmatrix}$$

[ ]:
```

```

[24]:
```
dP = P.jacobian([theta_1, theta_1, theta_3])
```

[25]:
```
dP
```

[25]:
$$\begin{bmatrix} -(l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\sin(\theta_1) & -(l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\sin(\theta_1) & -l_3\sin(\theta_2+\theta_3)\,c \\ (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\cos(\theta_1) & (l_2\cos(\theta_2)+l_3\cos(\theta_2+\theta_3))\cos(\theta_1) & -l_3\sin(\theta_1)\sin(\theta_2 \\ 0 & 0 & l_3\cos(\theta_2+\theta_ \end{bmatrix}$$

[26]:
```
print_latex(dP)
```

```
\left[\begin{matrix}- \left(l_{2} \cos{\left(\theta_{2} \right)} + l_{3}
\cos{\left(\theta_{2} + \theta_{3} \right)}\right) \sin{\left(\theta_{1}
\right)} & - \left(l_{2} \cos{\left(\theta_{2} \right)} + l_{3}
\cos{\left(\theta_{2} + \theta_{3} \right)}\right) \sin{\left(\theta_{1}
\right)} & - l_{3} \sin{\left(\theta_{2} + \theta_{3} \right)}
\cos{\left(\theta_{1} \right)}\\\left(l_{2} \cos{\left(\theta_{2} \right)} +
l_{3} \cos{\left(\theta_{2} + \theta_{3} \right)}\right) \cos{\left(\theta_{1}
\right)} & \left(l_{2} \cos{\left(\theta_{2} \right)} + l_{3}

\cos{\left(\theta_{2} + \theta_{3} \right)}\right) \cos{\left(\theta_{1}
\right)} & - l_{3} \sin{\left(\theta_{1} \right)} \sin{\left(\theta_{2} +
\theta_{3} \right)}\\0 & 0 & l_{3} \cos{\left(\theta_{2} + \theta_{3}
\right)}\end{matrix}\right]
```

[ ]:
```

```

Pendulo_simulacion

February 14, 2024

1 Anexo 8

1.1 Importar Librerias

[1]:
```
import numpy as np
from sympy import*
import matplotlib.pyplot as plt

init_printing()
```

1.1.1 Definicion Pendulo

[2]:
```
class Pendulo():
    def __init__(self, l, g, ts):
        super(Pendulo,self).__init__()
        self.l = l
        self.g = g
        self.ts = ts

    def f(self, x):
        l = self.l
        g = self.g
        theta = x[0]
        theta_p = x[1]
        theta_pp = -(g/l)*np.sin(theta)
        flow = np.array([theta_p, theta_pp])
        return flow

    def f_d(self, x):
        ts = self.ts
        x = x + ts*self.f(x)
        return x
```

1.1.2 Function Simulacion

[3]:
```
def simulation(ts, tf, x0):
    t = np.arange(0, tf + ts, ts, dtype=np.double)
    x = np.zeros((2, t.shape[0]), dtype = np.double)
    x[:, 0] = x0

    g = 9.8
    l = 1.0

    pendulo_1 = Pendulo(l, g, ts)

    for k in range(0, t.shape[0]-1):

        x[:, k+1] = pendulo_1.f_d(x[:, k])
    return x, t
```

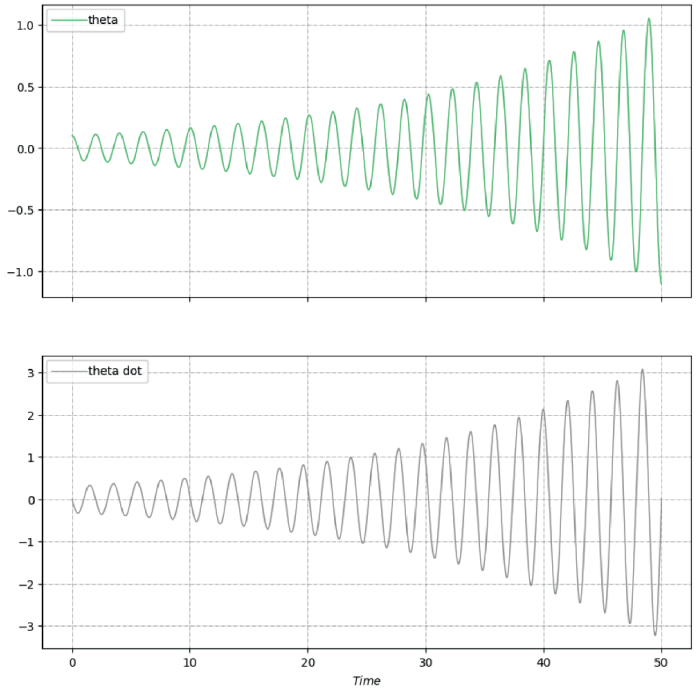
1.1.3 Inicializar parameteros simulacion

[4]:
```
ts = 0.01
tf = 50
x0 = np.array([0.1, 0])
x, t = simulation(ts, tf, x0)
```

1.1.4 Plot Resultados

[5]:
```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))

# Plot x_estimate with label 'x_estimate'
ax1.set_xticklabels([])
state_1, = ax1.plot(t, x[0, :], color='#3FB454', lw=1.0, ls="-")
state_2, = ax2.plot(t, x[1, :], color='#949494', lw=1.0, ls="-")
# Add a legend
ax1.legend([state_1],
           [r'theta'],
           loc="best",
           frameon=True, fancybox=True, shadow=False, ncol=2,
           borderpad=0.5, labelspacing=0.5, handlelength=3, handletextpad=0.1,
           borderaxespad=0.3, columnspacing=2)

ax2.legend([state_2],
           [r'theta dot'],
           loc="best",
           frameon=True, fancybox=True, shadow=False, ncol=2,
           borderpad=0.5, labelspacing=0.5, handlelength=3, handletextpad=0.1,
           borderaxespad=0.3, columnspacing=2)
```

```
ax1.grid(color='#949494', linestyle='-.', linewidth=0.5)
ax2.grid(color='#949494', linestyle='-.', linewidth=0.5)

ax2.set_xlabel(r"$Time}[s]$", labelpad=5)
# Show the plot
plt.show()
```



[6]:
```
fig.savefig('Pendulo.png')
```

[ ]:
```

```

Pendulo_simulacion-Runge-Kutta

February 14, 2024

1 Anexo 9

1.1 Importar Librerias

[1]:
```
import numpy as np
from sympy import*
import matplotlib.pyplot as plt

init_printing()
```

1.1.1 Definicion Pendulo

[2]:
```
class Pendulo():
    def __init__(self, l, g, ts):
        super(Pendulo,self).__init__()
        self.l = l
        self.g = g
        self.ts = ts

    def f(self, x):
        l = self.l
        g = self.g
        theta = x[0]
        theta_p = x[1]
        theta_pp = -(g/l)*np.sin(theta)
        flow = np.array([theta_p, theta_pp])
        return flow

    def f_d(self, x):
        ts = self.ts
        k1 = self.f(x)
        k2  = self.f(x + (ts/2)*k1)
        k3  = self.f(x + (ts/2)*k2)
        k4  = self.f(x + (ts)*k3)

        x = x + (ts/6)*(k1 + 2*k2 + 2*k3+ k4)
        return x
```

1.1.2 Function Simulacion

[3]:
```
def simulation(ts, tf, x0):
    t = np.arange(0, tf + ts, ts, dtype=np.double)
    x = np.zeros((2, t.shape[0]), dtype = np.double)
    x[:, 0] = x0

    g = 9.8
    l = 1.0

    pendulo_1 = Pendulo(l, g, ts)

    for k in range(0, t.shape[0]-1):

        x[:, k+1] = pendulo_1.f_d(x[:, k])
    return x, t
```

1.1.3 Inicializar parameteros simulacion

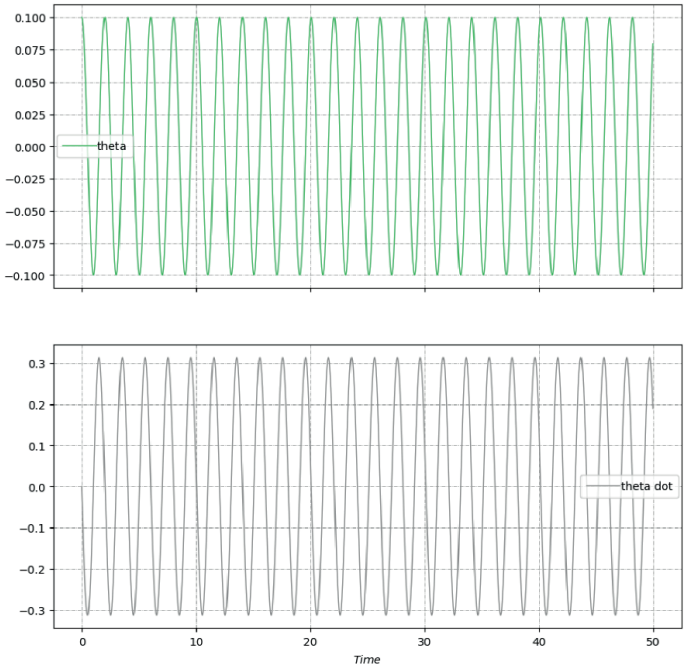
[4]:
```
ts = 0.01
tf = 50
x0 = np.array([0.1, 0])
x, t = simulation(ts, tf, x0)
```

1.1.4 Plot Resultados

[5]:
```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))

# Plot x_estimate with label 'x_estimate'
ax1.set_xticklabels([])
state_1, = ax1.plot(t, x[0, :], color='#3FB454', lw=1.0, ls="-")
state_2, = ax2.plot(t, x[1, :], color='#949494', lw=1.0, ls="-")
# Add a legend
ax1.legend([state_1],
           [r'theta'],
           loc="best",
           frameon=True, fancybox=True, shadow=False, ncol=2,
           borderpad=0.5, labelspacing=0.5, handlelength=3, handletextpad=0.1,
           borderaxespad=0.3, columnspacing=2)

ax2.legend([state_2],
           [r'theta dot'],
           loc="best",
           frameon=True, fancybox=True, shadow=False, ncol=2,
           borderpad=0.5, labelspacing=0.5, handlelength=3, handletextpad=0.1,
           borderaxespad=0.3, columnspacing=2)
```

```
ax1.grid(color='#949494', linestyle='-.', linewidth=0.5)
ax2.grid(color='#949494', linestyle='-.', linewidth=0.5)

ax2.set_xlabel(r"$Time}[s]$", labelpad=5)
# Show the plot
```

[5]:
```
Text(0.5, 0, '$Time}[s]$')
```



[6]:
```
fig.savefig('Pendulo_kutta.png')
```

[ ]:

[ ]:

La simulación de sistemas robóticos es un área de estudio en constante crecimiento. Gracias a los avances tecnológicos de la última década, se desarrollan algoritmos cada vez más complejos, ya sea por la estructura del robot o por estrategias avanzadas de control. En este contexto, los simuladores continúan ampliando sus capacidades con el objetivo de cubrir la mayor cantidad de necesidades posibles. Un ejemplo destacado es Webots, un simulador que ha tenido una evolución significativa desde su origen hasta la versión actual. Las últimas versiones ofrecen una variedad de prototipos robóticos comerciales ampliamente utilizados y la posibilidad de crear tu propio robot de múltiples formas.

Con este libro, esperamos contribuir a la formación de los estudiantes de ingeniería que desean desarrollarse en el apasionante mundo de la robótica.

ISBN: 978-9-9428-2188-1
9 789942 821881