

# Evaluación y simulación de un robot móvil omnidireccional de 4 ruedas (Astérix y youBot KUKA)

Cruz Juárez David Ricardo, Dionicio Pizarro Jorge Alberto, Hernández López Oliver Edson

**Abstract**—La teoría y la práctica siempre van de la mano, sin embargo, suelen haber muchas complicaciones cuando se busca aplicar ambas. El siguiente trabajo muestra el proceso realizado para simular un robot omnidireccional de 4 ruedas mecanum en el entorno Webots. A este punto el robot Astérix no es capaz de seguir una trayectoria de control, sin embargo, esto quiere decir que el paradigma de trabajo sea erróneo, si no que el conocimiento profundo acerca de un entorno de simulación es realmente importante. Comparamos 2 paradigmas de implementación de relaciones de movimiento y control, y analizamos los resultados de cada una de ellas. Finalmente, para comprobar que el control de trayectoria es completamente funcional, se le aplica al modelo de ejemplo youBot de la compañía KUKA disponible en la documentación de Webots.

**Index Terms**—CoppeliaSim, Control de Trayectoria, Robot Omnidireccional, Simulación de Robots, Webots.

## I. INTRODUCCIÓN

La robótica móvil reúne ciencias del ámbito de la ingeniería e informática, a la vez que ciencias cognitivas, inteligencia artificial y otras disciplinas. Solo mediante la cooperación entre todos estos ámbitos especializados es posible abarcar la complejidad de los robots móviles. Estos robots son capaces de moverse autónomamente y de ejecutar determinadas acciones, esto es, la independencia del robot respecto de la intervención humana. Por la importancia que presenta esta rama de la ingeniería existen diversos tipos de softwares que sirven de apoyo para poder simular tanto el entorno al que se sujetara el robot como el comportamiento funcional del mismo, tal es el caso de Webots que es una aplicación de escritorio multiplataforma de código abierto que se utiliza para simular robots. Proporciona un entorno de desarrollo completo para modelar, programar y simular robots en diversos entornos. El robot Astérix fue diseñado para ser un robot omnidireccional con seguimiento de trayectoria. Si bien los conocimientos teóricos acerca del comportamiento de los robots móviles omnidireccionales son claros, el estar poco familiarizado con el entorno y el factor de tiempo pueden generar muchos problemas a la hora de implementar todo el conocimiento adquirido en una simulación.

## II. ADECUACIONES DE ORIGEN

Inicialmente, nuestro robot estaba ensamblado como un conjunto de piezas sólidas con un origen fuera del centro del robot. Debido a que la posición exacta del punto central del robot es de suma importancia para el control de trayectoria,

fue necesario modificar el origen inicial de nuestro ensamble, obteniendo como prototipo final el mostrado en la figura 1.

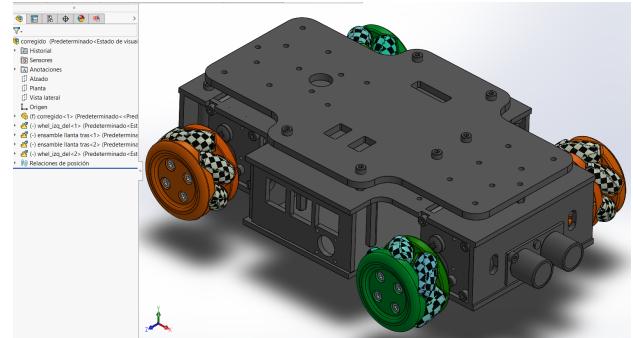


Fig. 1: Robot Omnidireccional ensamblado en el origen

### A. Adecuación de Diseño

En las primeras fases de prototipado se tomó la decisión de que las ruedas fueran piezas sólidas sin movilidad relativa. Esto imposibilita la opción de generar el movimiento natural de las ruedas mecanum, para ello se realizaron 3 subensambles de forma que los rodillos tuvieran un movimiento libre relativo a las ruedas (Véase figura 2).



Fig. 2: Subensambles de las piezas, Cuerpo, rueda Izquierda Inferior, rueda Derecha Superior

Debido a que la posición de la llanta superior izquierda tiene la misma geometría que la inferior derecha, característica particular de la configuración omnidireccional, se aplica lo mismo para la rueda superior derecha e inferior izquierda entonces el subensamble de las ruedas se puede reutilizar.

### B. Ensamble de Rodillos Individuales

Para el subensamble de los rodillos y su importación a Webots fue necesario exportar individualmente cada grupo de rodillos con sus respectivos ejes. Esto por cuestiones de orden y verificación de comportamiento dentro del entorno de Webots.

Al final se exportaron 24 rodillos con sus respectivos ejes en un archivo .VRML como se muestra en la figura 3.

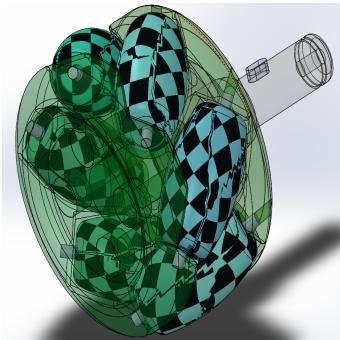


Fig. 3: Subensambles Rodillos y ejes forma individual

En la figura 4 se aprecia el resultado importado en el entorno de Webots.

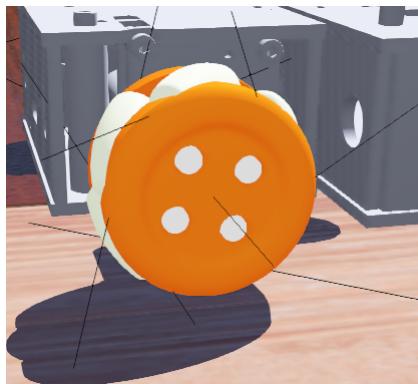


Fig. 4: Ejes de acción por rodillo

### III. PARADIGMAS DE TRABAJO

Para nuestro análisis utilizamos dos formas de movimiento relativo: asignando ejes y uniones independientes a cada rodillo de la rueda mecanum (forma convencional) como se muestra en la figura 4 y el uso de un eje virtual en dirección de la velocidad resultante (utilizado por CoppeliaSim) como se observa en la figura 5.

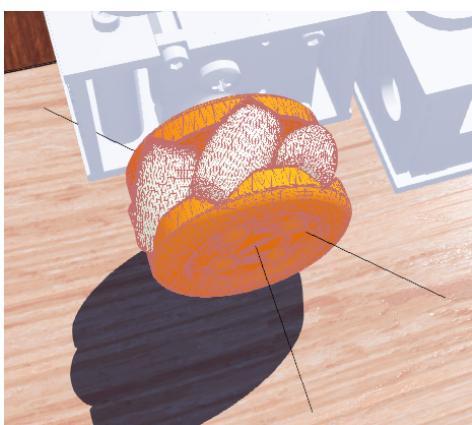


Fig. 5: Ejes de acción mitad de rueda

#### A. Configuración Individual por Rodillo

La figura 6 muestra como cada rodillo rodillo fue montado sobre su propio eje a 45 grados, debido a que al exportar un archivo .VRML se conservan las coordenadas de posición respecto al ensamble original cada rodillo y eje reconoce su posición original.

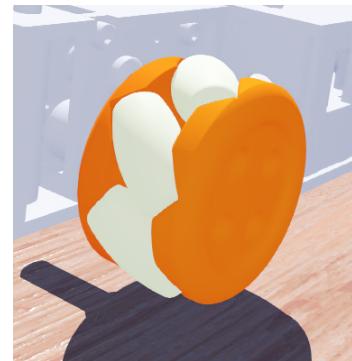


Fig. 6: Posición de los elementos sólidos una vez importados en Webots

1) *Configuración de Ejes de Acción por Rodillo:* Cada rodillo tiene un eje de acción sobre el cual rota cuando este se desliza, es necesario dentro de los parámetros en Webots indicar la información de este vector normalizado, así como el punto de anclaje de cada rodillo. Para lograr esto, se implementa un pequeño programa en Matlab que utiliza la información de posición de los centros de cada eje de cada rodillo y nos da como resultado el punto de anclaje (*Anchor*) y el vector de posición normalizado (*Axis*). Se muestra en la figura 7

```
%% Inicio del programa
C1 = v1-v2;
C1 = C1/norm(C1);
C2 = [(v1(1)+v2(1))/2 (v1(2)+v2(2))/2 (v1(3)+v2(3))/2];
C1 =
-0.5657 -0.4243 -0.7071
>> C2
C2 =
14.0000 21.5000 32.5000
```

Fig. 7: Programa Matlab distancia entre dos vectores

Una vez obtenidos los parámetros se puede colocar la información resultante en Webots como se muestra en la figura 8. Debemos recordar que este procedimiento se debe aplicar a los 24 rodillos que componen las 4 ruedas mecanum del robot.



Fig. 8: Programa Matlab distancia entre dos vectores

2) *Caracterización de Interacciones (Colisiones y Suspensión):* Se colocó como objeto colisionable los ejes y rodillos. Esto asegura que la unión entre ellos no se salga de su posición. Además de que permite a los rodillos interactuar (colisionar) con el entorno, permitiendo el movimiento del robot.<sup>9</sup>

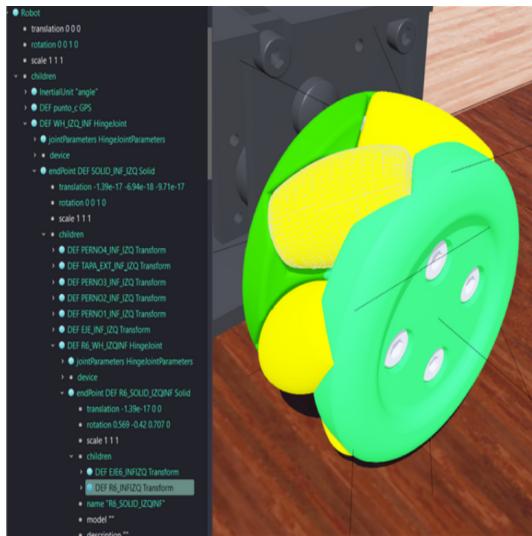


Fig. 9: Anidamiento de eje y rodillo en HingeJoint individuales

Debido a la jerarquización de las propiedades de colisión, el colocar la propiedad colisionable a los rodillos en una rama más alta provoca que no puedan interactuar correctamente con el resto de la rueda. Para solucionar este problema, se le agregó la propiedad de colisión a las ramas más bajas del árbol jerárquico, es decir, para cada rodillo de manera individual.

3) *Prueba con ecuaciones de control:* Al aplicar control básico de movimiento a la rueda inferior derecha y superior izquierda se esperaría que el robot se desplazara diagonalmente, sin embargo, esto no fue así. Como se puede apreciar en la figura 10, el movimiento relativo de los rodillos (movimiento libre) no se ejecutaba correctamente. Esto provocó que el robot realizara movimientos muy bruscos y propios de un conjunto de ruedas sin rodillos.

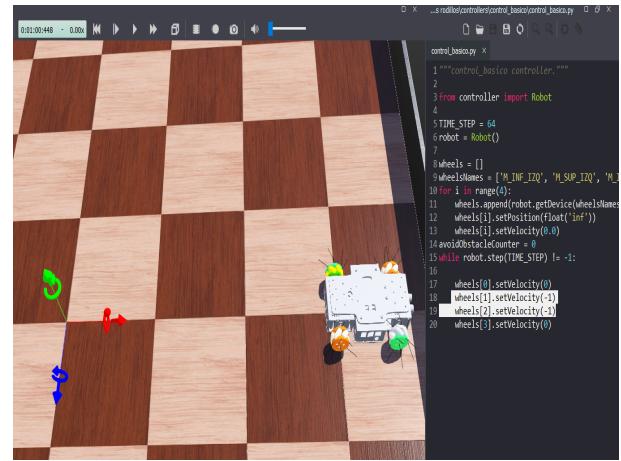


Fig. 10: Ejes de acción con rodillos independientes

Se decidió implementar el control al robot omnidireccional al robot con rodillos independientes con la esperanza de que el problema anterior se debiera únicamente a la programación. Si variamos el coeficiente K entre valores 0.1 a 2, el robot trata de aproximarse a la trayectoria deseada, llegando a un estado donde el robot no reconoce la trayectoria, y el error entre el punto deseado y el punto central del robot es demasiado grande, lanzando al robot fuera del mundo y lanzando un error en la consola. Esto se puede observar en la figura 11 y figura 12.

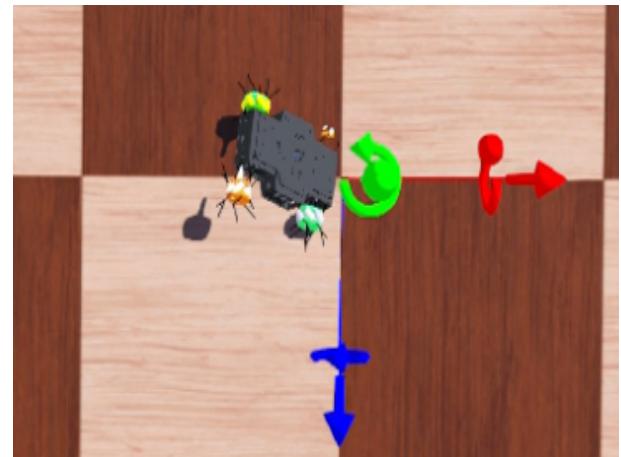


Fig. 11: Ejes de acción con rodillos independientes, Trayectoria Lemniscata

Estos resultados comprueban que el paradigma usado en base al comportamiento natural mecánico de los rodillos no es funcional dentro de Webots. Por lo que debemos pasamos al segundo paradigma de trabajo.

```

Console - All
WARNING: Robot > DEF WH_I2Q_INF HingeJoint > RotationalMotor: The requested velocity -172.187 exceeds 'maxVelocity' = 10.
WARNING: Robot > DEF WH_I2Q_SUP HingeJoint > RotationalMotor: The requested velocity -1081.8 exceeds 'maxVelocity' = 10.
WARNING: Robot > DEF WH_DER_INF HingeJoint > RotationalMotor: The requested velocity -1088.41 exceeds 'maxVelocity' = 10.
WARNING: Robot > DEF WH_DER_SUP HingeJoint > RotationalMotor: The requested velocity -170.798 exceeds 'maxVelocity' = 10.
-272.95776839742086 -226.24449167455901 0.256
WARNING: Robot > DEF WH_I2Q_INF HingeJoint > RotationalMotor: The requested velocity -2770.02 exceeds 'maxVelocity' = 10.
WARNING: Robot > DEF WH_I2Q_SUP HingeJoint > RotationalMotor: The requested velocity -2956.31 exceeds 'maxVelocity' = 10.
WARNING: Robot > DEF WH_DER_INF HingeJoint > RotationalMotor: The requested velocity -2954.6 exceeds 'maxVelocity' = 10.
WARNING: Robot > DEF WH_DER_SUP HingeJoint > RotationalMotor: The requested velocity -2768.3 exceeds 'maxVelocity' = 10.

Pause the simulation (Ctrl+O)

```

Fig. 12: Consola Rodillos independientes, Trayectoria Lemniscata

### B. Configuración por Eje Virtual en Webots

Uno de los softwares más populares para la simulación de robots de todo tipo es CoppeliaSim. Como muchos otros, este software cuenta con robots disponibles que múltiples compañías han desarrollado, con el fin de que el usuario se familiarice con el software realizando pruebas de funcionamiento, cambios en el entorno, diseño de algoritmos de control, entre muchas otras cosas. El robot omnidireccional disponible en CoppeliaSim es el modelo *youBot* de **KUKA**. Este robot fue diseñado utilizando mecanismos PROTO, por lo que al ser insertado en el entorno de trabajo ya cuenta con toda la configuración de los parámetros necesarios y se encuentra listo para ser programado, o en su defecto, ejecutar el programa de muestra.



Fig. 13: CoppeliaSim

La estructura del robot de **KUKA** está basada en el análisis cinemático del robot omnidireccional. Este no utiliza un eje de rotación por cada rodillo, en su lugar, configura un eje virtual encargado de darle la propiedad rotativa al conjunto de rodillos en la dirección de la velocidad resultante. (Véase figura 14).

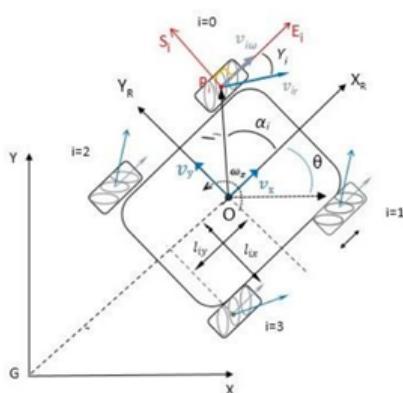


Fig. 14: Diagrama de cuerpo libre del robot omnidireccional de 4 ruedas mecanum,

Se observa en la figura 15 la estructura que tiene el robot. Se tienen configurados ejes individuales para cada una de las ruedas identificados como *rollingJoint rl*, *rollingJoint rr*, *rollingJoint fr* y *rollingJoint fl*. También podemos ver la unión asignada para el brazo manipulador (*youBotArmJoint0*) y finalmente el **Padre** de la jerarquía al que todos los elementos están unidos, *youBot*.

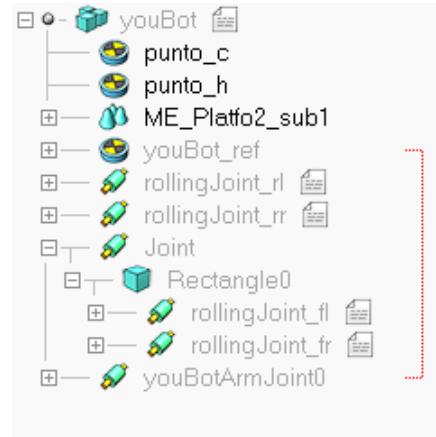


Fig. 15: Estructura jerárquica del youBot de KUKA

Al abrir la jerarquía de la que está compuesta cada una de las ruedas nos encontramos con 4 elementos fundamentales. (Véase figura 16).

- Una unión intermedia
- El sólido que representa a la rueda (SwedishWheel)
- Un eje de acción para el conjunto de rodillos (Wheel respondable)
- El sólido que representa a los rodillos



Fig. 16: Estructura jerárquica de la rueda mecanum en CoppeliaSim

Ahora bien, dentro del elemento más elevado en la jerarquía del árbol de relaciones y nodos (*rollingJoint rl*) podemos encontrar la configuración programada en LUA para la rueda mecanum. (Véase figura 17).

```

function sysCall_actuation()
    sim.resetDynamicObject(wheel)

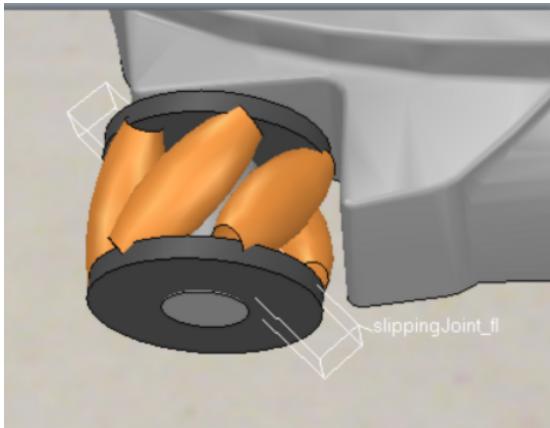
    sim.setObjectPosition(slipping+sim.handleflag_reljointbaseframe,rolling,{0,0,0})
    sim.setObjectOrientation(slipping+sim.handleflag_reljointbaseframe,rolling,[math.pi/4,0,math.pi])

    sim.setObjectPosition(wheel+sim.handleflag_reljointbaseframe,rolling,{0,0,0})
    sim.setObjectOrientation(wheel+sim.handleflag_reljointbaseframe,rolling,{0,0,0})
end

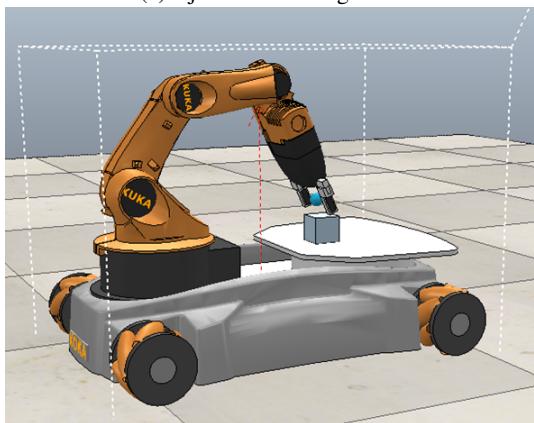
```

Fig. 17: Programación en LUA del comportamiento de la rueda

El objeto manejable Wheel es la estructura de la rueda completa y cómo podemos ver, se le asignan dos clases de movimientos: El movimiento relacionado directamente al eje del motor principal de la rueda y un movimiento 45 grados sobre un eje virtual (el eje de acción del conjunto de rodillos) sin entrada de potencia. El resultado en el entorno de simulación se puede observar en la figura 18.



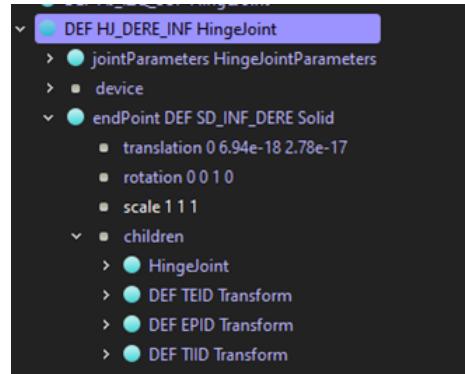
(a) Eje virtual a 45 grados



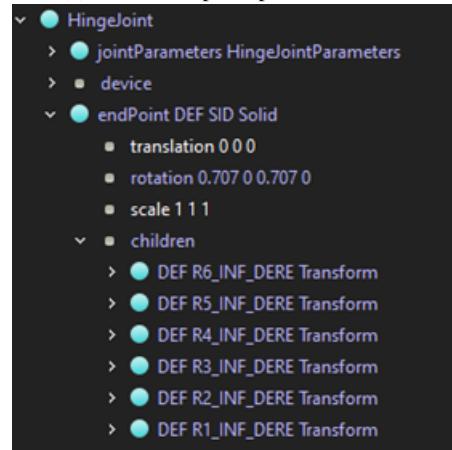
(b) youBot KUKA

Fig. 18: Resultado de la configuración del youBot en CoppeliaSim

1) *Configuración de Eje virtual a 45 grados en Astérix:* Lo primero que debemos identificar en Webots es el cómo implementar la estructura de las 4 partes fundamentales. Esto resulta ser poco complicado ya que CoppeliaSim y Webots utilizan casi la misma nomenclatura.



(a) Estructura principal de la rueda



(b) Estructura anidada de los rodillos

Fig. 19: Estructura jerárquica de la rueda completa

La figura 19 muestra el árbol de nodos aplicado a las ruedas de forma que se apliquen las mismas reglas que al modelo de ejemplo de CoppeliaSim. Podemos observar 4 elementos fundamentales en la estructura:

- El *endPoint* de la rueda general (La unión intermediaria)
- El conjunto de transformadas en el *children* de la unión principal: TEID, EPID, TIID (Los sólidos que representan la rueda)
- El *HingeJoint* dentro del *children* de la unión principal (El eje de acción para el conjunto de rodillos)
- El conjunto de transformadas dentro del *children* de la unión del eje de acción de los rodillos (Los sólidos que representan a los rodillos)

Es oportuno decir que el parámetro *Axis* son las coordenadas normalizadas por las cuales pasa el eje de acción de la unión y el parámetro *Anchor* es el punto en donde se “ancla” la unión. (Véase la figura 20).

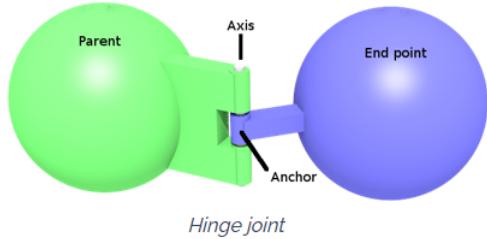


Fig. 20: Definición de *HingeJoint*

Una vez colocados los nodos de manera jerárquica es necesario configurar la dirección de los ejes de acción para ambos movimientos, esto se logra modificando dos parámetros: *Axis* y *Anchor*.

2) *Caracterización de Interacciones (Colisiones y Suspensión)*: Para agregar la interacción física con el mapa agregamos cada uno de los elementos que componen la rueda a un nodo de tipo *Group* en el parámetro *BoundingObject* como se muestra en la figura 22.

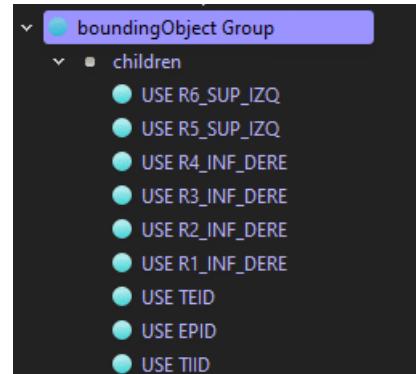


Fig. 22: Aplicación de propiedad de colisión a todos los elementos que componen la rueda mecanum

Puesto que no queremos que nuestro robot tenga las vibraciones de un auto sin suspensión debemos identificar sobre qué eje debe actuar la suspensión virtual, así como el coeficiente de amortiguamiento del resorte.

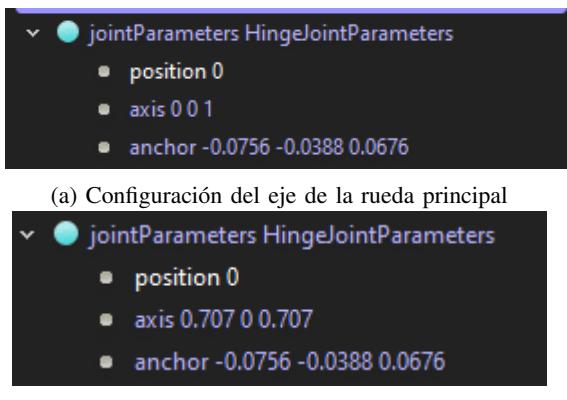


Fig. 21: Configuración de parámetros para el movimiento de la rueda y los rodillos

La figura 21a muestra los parámetros para el eje principal (al que se asignará la velocidad), mientras que la figura 21b muestra los parámetros para el eje virtual sobre el cual deberían girar los rodillos. Todos estos parámetros se obtuvieron del ensamblaje realizado en SolidWorks.

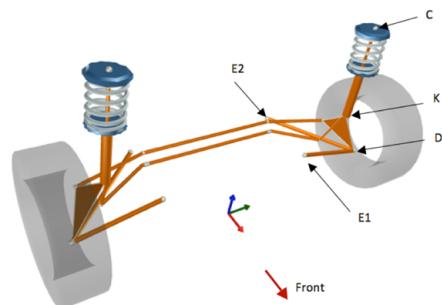
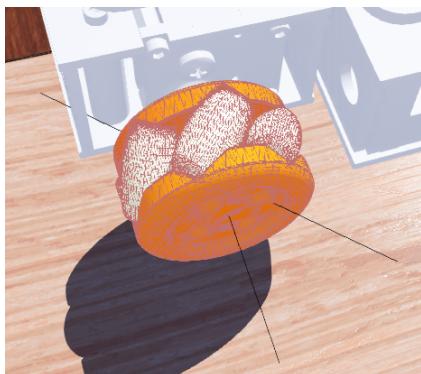


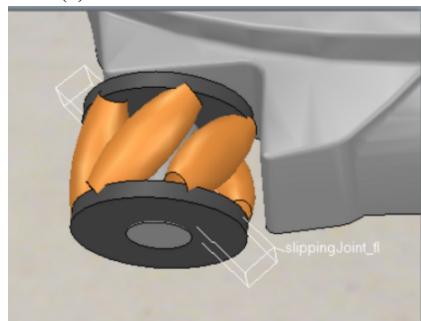
Fig. 23: Diagrama de una suspensión simple

La figura 23 muestra un diagrama de dos ruedas con una suspensión simple por resortes y cómo podemos observar, el eje de acción del eje de suspensión es perpendicular (o al menos con una ligera inclinación) al eje de acción del eje de movimiento principal. Para ello, modificamos el eje por defecto (eje x) y colocamos el eje correspondiente a nuestro diseño, el eje y.

Finalmente se puede apreciar la similitud del resultado entre el modelo *youBot* de CoppeliaSim y Astérix. (Véase figura 24).



(a) Rueda Mecanum en Webots



(b) Rueda Mecanum en CoppeliaSim

Fig. 24: Comparativa de rueda mecanum entre Webots y CoppeliaSim

3) *Prueba con Ecuaciones de Control:* Finalmente procedemos a aplicar este mismo procedimiento a las 3 ruedas restantes. Puesto que el propósito inicial es realizar pruebas de comportamiento no es necesario incluir los rodillos para todas las ruedas, basta con elegir 2 que estén sobre la misma diagonal y con ello podemos probar movimientos laterales, diagonales y de giro.



Fig. 25: Prototipo final previo a la implementación del control

Dados todos estos parámetros, el robot debería realizar el movimiento de manera correcta o al menos aproximada, sin embargo, esto no es así (Véase figura 26).

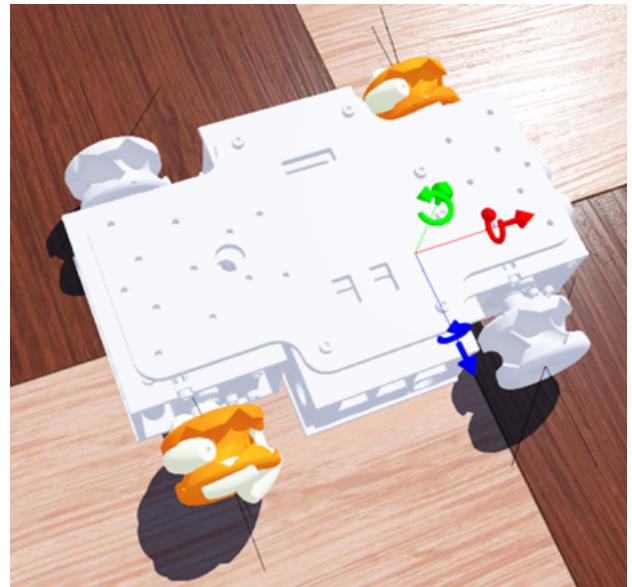


Fig. 26: Resultado de simulación

El robot trata de moverse en la dirección correspondiente al control, sin embargo, por alguna extraña situación, el entorno no respeta los parámetros de Anchor y Axis, además de que no respeta la jerarquización de los elementos. Como consecuencia, los rodillos no respetan el movimiento relativo a la rueda principal. Se realizaron múltiples pruebas cambiando la jerarquización de los objetos colisionables, modificando los parámetros de Anchor y Axis, así como realizando nuevas formas de ensamblaje de las piezas dentro del entorno de Webots como de SolidWorks.

#### IV. ANÁLISIS Y COMPARACIÓN DE RESULTADOS DE LOS PARADIGMAS

Debido a que todos los integrantes del equipo tuvieron experiencias diferentes al momento de idear, planificar e implementar estos paradigmas podemos expresar 3 puntos de vista.

**Cruz Juárez David Ricardo:** Comparando los resultados de ambas pruebas realizadas en Webots, se observa que aunque la forma de colocar, ensamblar o reconstruir al robot omnidireccional sea diferente e implementando propiedades que aparentemente brindarían un comportamiento correspondiente a un robot omnidireccional, solo obtuvimos resultados no favorables, pues en el primer método de rodillos independientes no se pudo lograr obtener el deslizamiento de cada rodillo esto se podía comprobar en la simulación, el robot se trababa o se comportaba como un robot diferencial. En el método donde imitábamos el comportamiento del entorno de CoppeliaSim resultó de igual manera insatisfactoria debido a que los rodillos rotaban con respecto a ese eje virtual colocado en el eje central de la llanta, se pensaba que esto se podía deber a que solo usábamos 2 ruedas para probar deslizamientos, pero al simular con 4 ruedas el resultado era el mismo, un robot con problemas de reconocimiento de trayectoria y sin el comportamiento esencial de las llantas.

**Dionicio Pizarro Jorge Alberto:** El primer paradigma utilizado es el que naturalmente debió funcionar. Existen videos promocionales de Webots mostrando características del robot youBot de KUKA y funcionalmente son bastante similares. Cada rodillo tiene una propiedad de colisión y ubicación de eje de acción independiente de todos los demás, permitiendo el movimiento adecuado de la rueda mecanum. Puede que lo que no permita que se realice el comportamiento adecuado sean pequeños detalles en la configuración de colisión o también podrían ser detalles de jerarquización. El segundo paradigma es un poco más nuevo, sin embargo, es el que mostro los mejores resultados. EL problema principal es que no respetaba las relaciones de colisión entre las ruedas y los rodillos por lo que, solucionado ese problema, el robot debería ser completamente funcional dentro del entorno. Ambos caminos requieren invertir tiempo y cerebro para estructurar mejor las ideas y ver los posibles errores que se estén cometiendo. Eventualmente podremos llegar a la solución.

**Hernández López Oliver Edson:** Se pudo observar que el entorno de simulación Webots presenta grandes complicaciones al momento de simular un movimiento en un robot culla orientación de movimiento es no paralelo con respecto al eje de coordenadas y del movimiento tal es el caso de las ruedas mecanum. lo anterior se puede sustentar con la diferentes pruebas realizadas, ya que como se explicó al momento de tener los ejes de acción independientes para cada rodillo estos no siguen la dirección del eje de acción aunque fuera independiente y con movimiento libre, al intentar poner un objeto "imaginario" a cada rueda en dirección al movimiento que debería realizar (idea ocupada en el entorno de CopeliaSim) donde este simularía la dirección del movimiento, esta última tampoco funciono ya que al momento de realizar una trayectoria o movimiento este colisionaba con la misma llanta lo que hacía que se saliera de su eje.

## V. PROGRAMACIÓN DE ROBOT YOUTBOT DE KUKA

De acuerdo con Webots, El youBot es un brazo robótico móvil desarrollado por KUKA. Su brazo tiene cinco grados de libertad y una pinza lineal. Su base tiene cuatro ruedas Mecanum que permiten un movimiento omnidireccional. Estas ruedas se modelan de manera eficiente utilizando fricción asimétrica.



Fig. 27: Robot Omnidireccional youBot de KUKA

La ventaja de usar el modelo de ejemplo como robot para aplicarle control de trayectoria es que este robot ya ha sido probado y evaluado en el entorno de Webots. El diseño mecánico del robot del robot es significativamente más complejo que el diseño y exportación tradicional de un archivo VRLM.

### A. Análisis de composición Mecánica

Las partes de componen al youBot, así como las propiedades geométricas, de colisión, de rotación y de traslación están diseñadas y expresadas por medio de archivos. PROTO. El mecanismo PROTO permite a los usuarios ampliar el conjunto de nodos agregando sus propios nodos. Así, los usuarios pueden construir y reutilizar objetos complejos. Además, los archivos PROTO pueden contener la información de la malla que cubre los sólidos en el caso en que se quiera tener una interacción de colisión con algún entorno.

Arm0Mesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	23 KB
Arm1Mesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	43 KB
Arm2Mesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	43 KB
Arm3Mesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	72 KB
Arm4Mesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	37 KB
Arm5Mesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	25 KB
BlackAppearance.proto	14/12/2020 04:29 p.m.	Archivo PROTO	1 KB
BlackMetalAppearance.proto	14/12/2020 04:29 p.m.	Archivo PROTO	1 KB
BodyMesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	80 KB
ExteriorWheel.proto	14/12/2020 04:29 p.m.	Archivo PROTO	3 KB
FingerMesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	7 KB
InteriorWheel.proto	14/12/2020 04:29 p.m.	Archivo PROTO	3 KB
KukaAlphaAppearance.proto	14/12/2020 04:29 p.m.	Archivo PROTO	1 KB
KukaBlackAppearance.proto	14/12/2020 04:29 p.m.	Archivo PROTO	1 KB
KukaBox.proto	14/12/2020 04:29 p.m.	Archivo PROTO	12 KB
MetalAppearance.proto	14/12/2020 04:29 p.m.	Archivo PROTO	1 KB
OrangeAppearance.proto	14/12/2020 04:29 p.m.	Archivo PROTO	1 KB
PlateMesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	9 KB
SubWheel.proto	14/12/2020 04:29 p.m.	Archivo PROTO	1 KB
SubWheelMesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	10 KB
WheelMesh.proto	14/12/2020 04:29 p.m.	Archivo PROTO	49 KB
Youbot.proto	14/12/2020 04:29 p.m.	Archivo PROTO	33 KB

Fig. 28: Archivos PROTO del robot youBot de KUKA

La figura 28 muestra el conjunto de archivos que componen al robot de KUKA. Cada uno de ellos contiene toda la información acerca de su comportamiento, geometría y posición de cada una de las partes del youBot. Podemos apreciar en la figura 29 todas las características que tiene el "programa principal" del youBot entre las que se encuentran rotación, traslación, sincronización, malla de cuerpo, etc. Solo el programa principal consta de 863 líneas de código

```
Robot {
    translation IS translation
    rotation IS rotation
    customData IS customData
    supervisor IS supervisor
    synchronization IS synchronization
    name IS name
    model "KUKA youBot"
```

Fig. 29: Programa PROTO principal del youBot

## B. Análisis de Librerías

El robot cuenta con 3 librerías propias (véase figura 30): La primera se encarga de la inicialización de motores y la generación de movimiento del brazo manipulador, la segunda de la base y la tercera del efecto final (la pinza). Para nuestro caso solamente usaremos la de la base, ya que no contamos con ningún brazo manipulador.

```
27
28 #include <arm.h>
29 #include <base.h>
30 #include <gripper.h>
31
```

Fig. 30: Librerías de control del youBot

Dentro de la librería se encuentran muchas funciones escritas en lenguaje C, las cuales se encargan de realizar tanto funciones básicas como complejas como: inicializar los motores, asignar velocidades a cada una de las ruedas, fijar un punto en el plano al cual el robot debe llegar, la activación del GPS o el Compás, entre muchas otras más.

Debemos prestar principal atención en 4 funciones principales dentro de la librería "base.h": *base goto init()*, *base goto set target* y *base goto run*. (Véase figura )

- La primera se encargará de inicializar los parámetros de GPS y compás del robot, sin los cuales el robot no podría saber ninguna información acerca de su posición. También coloca valores por defecto a sus trayectorias para que el robot permanezca quieto.(figura 31a)
- La segunda se encarga de configurar el punto dentro del plano x-z al que el robot debe llegar (figura 31b)
- La tercera es la más importante, ya que en ella a partir del punto deseado en la función anterior calcula los vectores de posición del robot y del punto deseado, y calcula las velocidades necesarias en cada rueda para que el robot este en la posición deseada con la orientación elegida. (figura 31c)

```
void base_goto_init(double time_step) {
    gps = wb_robot_get_device("gps");
    compass = wb_robot_get_device("compass");
    if (gps)
        wb_gps_enable(gps, time_step);
    if (compass)
        wb_compass_enable(compass, time_step);
    if (!gps || !compass)
        fprintf(stderr, "cannot use goto feature without GPS and Compass");

    goto_data.v_target.u = 0.0;
    goto_data.v_target.v = 0.0;
    goto_data.alpha = 0.0;
    goto_data.reached = false;
}
```

(a) Función *base goto init()*

```
void base_goto_set_target(double x, double z, double alpha) {
    if (!gps || !compass)
        fprintf(stderr, "base_goto_set_target: cannot use goto feature without GPS and Compass");

    goto_data.v_target.u = x;
    goto_data.v_target.v = z;
    goto_data.alpha = alpha;
    goto_data.reached = false;
}
```

(b) Función *base goto set target()*

```
void base_goto_run() {
    if (!gps || !compass)
        fprintf(stderr, "base_goto_set_target: cannot use goto feature without GPS and Compass");

    // get sensors
    const double *gps_raw_values = wb_gps_get_values(gps);
    const double *compass_raw_values = wb_compass_get_values(compass);

    // compute 2d vectors
    Vector2 v_gps = {gps_raw_values[0], gps_raw_values[2]};
    Vector2 v_front = {compass_raw_values[0], compass_raw_values[1]};
    Vector2 v_right = {-v_front.v, v_front.u};
    Vector2 v_north = {1.0, 0.0};
}
```

(c) CFuncióñ *base goto run()*

Fig. 31: Funciones principales de la librería *base.h*

## C. Adecuaciones del robot y el entorno

Dentro del entorno, el robot posee una serie de parámetros generales configurables a gusto del usuario, por ejemplo, el número de brazos que se desean que estén sobre el robot, un apartado para dispositivos (GPS, Compás, LiDar, etc.) llamado bodySlot, un apartado para seleccionar el controlador deseado para el robot y configuraciones espaciales de la posición del robot.

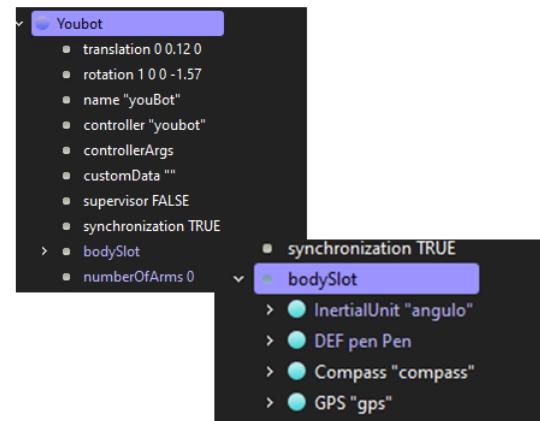


Fig. 32: Parámetros de configuración del robot

Para nuestro caso, no deseamos tener ningún brazo y deseamos tener un GPS, un lápiz y un Compás. Por lo que nuestro robot termina como se muestra en la figura 33:



Fig. 33: Modelo adecuado del youBot KUKA

#### D. Generación de código de trayectoria

El proceso demostrativo se dividió en 3 grandes etapas:

- Inicialización
- Configuración de parámetros
- Asignación de coordenadas de destino respecto a una trayectoria

De esta forma, es mucho más sencilla la implementación del control de trayectoria ya que todo el conjunto de librerías disponibles para el robot de KUKA son lo suficientemente abstractas para que no nos tengamos de preocupar por el bajo nivel de control. Nuestro código queda de la siguiente manera:

```
int main(int argc, char **argv) {
    wb_robot_init();
    double zhd,xhd,t,a;
    //double angle;

    a=2.0;
    base_init();
    //arm_init();
    //gripper_init();
    base_goto_init(TIME_STEP);
    passive_wait(1.0);

    pen = wb_robot_get_device("pen");
    wb_pen_write(pen,true);
    int rojo = rand() & 0xff;
    wb_pen_set_ink_color(pen,rojo,0.9);

    while(true){
        step();
        t = 0.01*wb_robot_get_time();

        //Lemniscata
        //zhd=a*cos(t)/(1+sin(t)*sin(t));
        //xhd=(a*sin(t)*cos(t))/(1+sin(t)*sin(t));
        //angle = atan2(xhd,zhd);
        //Corazón
        zhd=(13*cos(t)-5*cos(2*t)-2*cos(3*t)-cos(4*t))/10;
        xhd=(12*sin(t)-4*sin(3*t))/10;

        base_goto_set_target(-xhd,zhd,0);
        base_goto_run();
    }
    wb_robot_cleanup();
    return 0;
}
```

Fig. 34: Control de trayectoria del robot en lenguaje C

#### E. Análisis de resultados

El robot se inicializa con un paso de simulación de 64 milisegundos, un color de lápiz rojo, un paso de tiempo referente a la trayectoria alrededor de 640 microsegundos y dos trayectorias a elegir. La famosa Lemniscata o la forma de un Corazón.

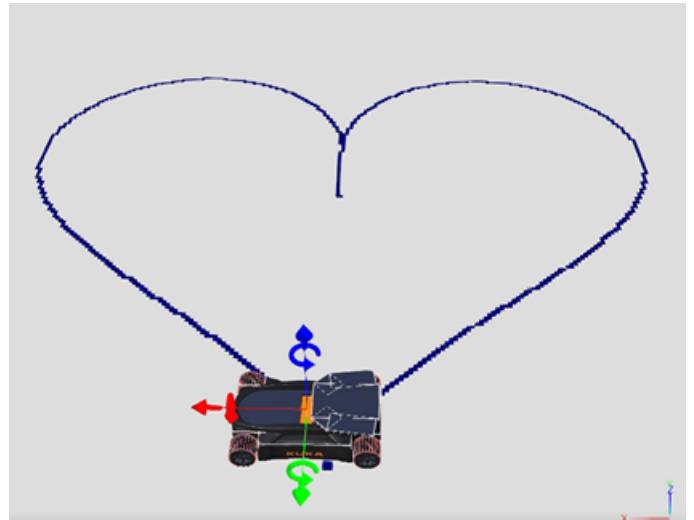


Fig. 35: Trayectoria de Corazón

Se observa en la figura 35, el corazón tiene una alta definición en el dibujo. Esto se debe al paso tan pequeño que utilizamos. Si este paso fuera más grande la definición de la trayectoria sería muy pobre y la forma difícilmente se logaría apreciar.

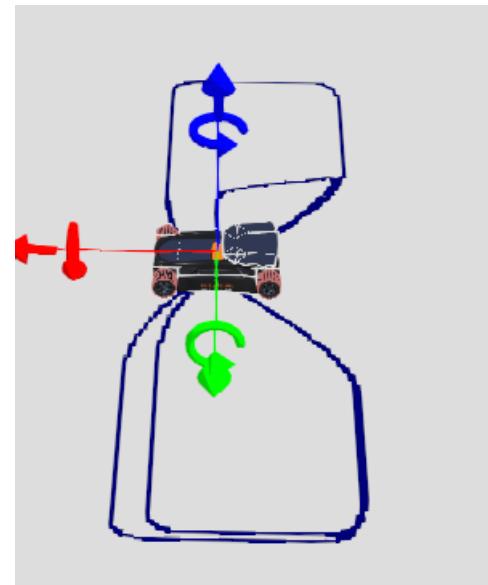


Fig. 36: Trayectoria de Lemniscata con baja definición

La falta de definición en el paso se ve más evidente a la hora de realizar la lemniscata, siendo la figura 36 el resultado para cuando existe un paso de 6.4 milisegundos. Mientras que para el paso de 640 microsegundos el resultado mejora significativamente (figura 37).

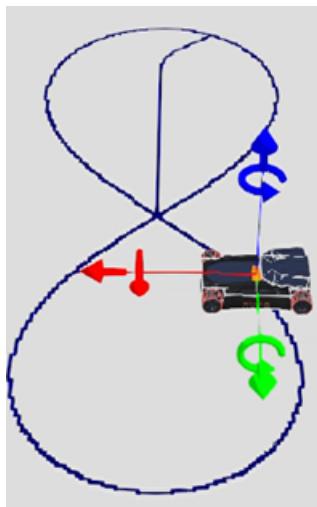


Fig. 37: Trayectoria de Lemniscata con alta definición

## VI. COMPARACIÓN ENTRE ASTÉRIX Y YOUBOT KUKA

Definitivamente el proceso de diseño entre ambos robots es diferente. Los parámetros de funcionamiento que tiene el robot de KUKA está a un nivel mucho menos abstracto que el que ofrece SolidWorks y la extensión VRLM. Lamentablemente, el paradigma de diseño que toma *youBot* no puede ser utilizado ya que es bastante desconocido y tomaría demasiado tiempo realizar el robot en archivo PROTO. Tampoco podemos utilizar este robot más que para pruebas de algoritmos de control y arquitecturas hibridas de funcionamiento, ya que carece del elemento que utilizamos al momento de configurar el robot Astérix. Sin embargo, todo el conocimiento y análisis del código de los elementos que componen al *youBot* seguramente serán de utilidad a la hora de buscar alguna clase de error en el prototipo principal.

## VII. CONCLUSIONES Y OBSERVACIONES

Si bien el entorno de Webots es una herramienta bastante poderosa al momento de simular, presenta puntos negativos en comparación con otros entornos del mismo rubro. El simular ruedas tipo Mecanum se vuelve una tarea bastante compleja por alguna razón, cuando solamente debería depender del comportamiento mecánico y no de cuestiones de entorno. Esto se puede comprobar después de haber realizado las pruebas de nuestro robot omnidireccional y obtener resultados no favorables en este entorno, se opta por la alternativa de controlar el robot *youbot* de KUKA disponible en el entorno. La comparación entre el robot de KUKA y Astérix es visible desde el momento en el que se analiza el diseño. El objetivo de realizar tantas pruebas era hacer funcionar al robot de manera adecuada, si bien esto no fue posible a este momento, descubrimos nuevas funciones y propiedades de Webots nos hacen entender mejor el funcionamiento del simulador, además de ofrecernos más alternativas para hacer el robot completamente funcional. El encontrar una solución adecuada al problema de las ruedas mecanum llegará eventualmente con el suficiente tiempo y las suficientes pruebas al robot y al entorno. Se podrían simular comportamientos

menos complejos pero que comparten la característica de ser movimientos relativos rotacionales.

## VIII. REFERENCIAS

How to design a 2 wheel differential drive robot in Webots?. 2020. [video] Directed by K. Kajal Gada. YouTube. Disponible en <https://www.youtube.com/watch?v=ebGJzymXv-o>

Gai, V., 2020. what can i do to solve Webots error message. [Internet] Stack Overflow. Disponible en: <https://stackoverflow.com/questions/61214114/what-can-i-do-to-solve-webots-error-message> [Recuperado el 4 de Mayo 2021].

Webots. <http://www.cyberbotics.com>. Open-source Mobile Robot Simulation Software.

E. Rohmer, S. P. N. Singh, M. Freese, "CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework", IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2013. [www.coppeliarobotics.com](http://www.coppeliarobotics.com)

**Cruz Juárez David Ricardo** estudiante de 10 semestre de la carrera de Mecatrónica en la Universidad Tecnológica de la Mixteca.

**Dionicio Pizarro Jorge Alberto** estudiante de 10 semestre de la carrera de Mecatrónica en la Universidad Tecnológica de la Mixteca.

**Hernández López Oliver Edson** estudiante de 10 semestre de la carrera de Mecatrónica en la Universidad Tecnológica de la Mixteca.