Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño mecánico de un modelo de robot bípedo: Estructura y configuración

Autor

Fernando Pellicena García

Directores

Manuel González Bedía

Juan Lladó París

Escuela de Ingeniería y Arquitectura, Universidad de Zaragoza

Junio de 2016

AGRADECIMIENTOS

Varias han sido las personas que me han ayudado en la realización de este trabajo, por ello escribo estas líneas de agradecimiento a su apoyo durante la realización del mismo.

En primer lugar quisiera agradecer a Manuel, la oportunidad de realizar este proyecto que ha logrado fusionar nuestros campos, informática y mecánica. Manuel no solo ha sido un tutor sino que ha sido un gran apoyo en todo momento durante estos duros últimos meses de grado. Quisiera agradecer también a Juan, quien estuvo dispuesto a ayudarme cuando necesite un apoyo mecánico.

Este trabajo, no es un trabajo más, supone el final de una etapa, por ello no quiero olvidarme de aquellas personas que compartieron este largo camino a mi lado, ellos fueron mis compañeros de universidad, a los cuales les agradezco su apoyo, cercanía y compañerismo durante estos años.

En estas líneas quiero acordarme y dar las gracias a mi padre, a mi madre y a mi hermana, a mi familia y a ese pilar tan importante de mi vida que son mis amigos.

Por último, quiero hacer una mención especial a Julio, Lola, Nati y Paco, ellos son mis abuelos, a quienes quiero dedicarles este trabajo, y todo lo que él significa.

Diseño mecánico de un modelo de robot bípedo:

Estructura y configuración.

RESUMEN

El presente trabajo consiste principalmente en el desarrollo de la cinemática de un robot bípedo, que servirá como base para posibles estudios futuros.

En este proyecto se aplican las bases de la mecánica al conocimiento de la caminata de un robot bípedo y se desarrollan en un entorno virtual desconocido hasta el momento, Webots, --seleccionado entre diferentes software similares que se encuentran en el mercado- un software de aplicación robótica, el cual ha sido estudiado hasta llegar a ser controlado para la correcta realización de este trabajo.

El robot bípedo estudiado constará de seis grados de libertad que son el mínimo número de grados de libertad que permiten realizar la acción de caminar. Para poder desarrollar su estructura, previa al desarrollo del movimiento en el software Webots, han sido necesarios softwares de diseño como AutoDesk Inventor y AutoCAD.

En este trabajo se incluyen todos los datos necesarios, que permiten alcanzar el modelo cinemático en nuestro robot, a su vez se asientan las bases del modelo dinámico, que serán las bases que permitirán que interactúe nuestro robot en un entorno real.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. FERNANDO PELLICENA GARCÍA,

con nº de DNI 17768687-Z en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) GRADO EN INGENIERÍA MECÁNICA, (Título del Trabajo) DISEÑO MECÁNICO DE UN MODELO DE ROBOT BÍPEDO: ESTRUCUTURA Y CONFIGURACIÓN

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, JUNIO DE 2016

Fdo: FERNANDO PELLICENA GARCÍA

ÍNDICE

1. Introducción

1.1 Introducción

Un robot es una máquina o ingenio electrónico programable capaz de realizar tareas repetitivas de forma más rápida, barata y precisa que los seres humanos, pudiendo además interaccionar con su entorno. La robótica humanoide es el área de la ingeniería dedicada al desarrollo de sistemas robotizados que buscan imitar las peculiaridades del ser humano.

Buscando imitar al ser humano, incluso en aspecto, el robot humanoide estará compuesto por dos piernas que le proporcionarán el movimiento, dos brazos que le ayudarán a equilibrarse o sujetarse y por una cabeza que, aunque no sea imprescindible, le aportará un aspecto más humano.

La principal ventaja de los robots humanoides sobre otro tipo de robots es que pueden trabajar directamente en el mismo entorno que los humanos, sin que se deban realizar modificaciones sobre dicho entorno, sin embargo, la complejidad en el diseño y control aumentan considerablemente. Debido a su complejidad de diseño y construcción dichos autómatas son muy caros y delicados ya que se pueden dañar fácilmente. Por esta razone es conveniente proporcionar a los usuarios un simulador de robots que les proporcione un entorno donde poder realizar pruebas sin riego de daño sobre los autómatas. En este proyecto se utilizará el simulador Webots para crear un modelo del robot humanoide.

Este simulador es un programa que permite al usuario crear y / o modificar un robot manipulador que imita el comportamiento del mismo. Las ventajas de la simulación son entre otras la posibilidad de evaluar su posible desempeño y realizar infinidad de pruebas ahorrando muchos recursos, costes y tiempo.

1.2 Motivaciones

La importancia que pueden llegar a tener en un futuro los robots humanoides, la complejidad de estas máquinas y el deseo que desde hace años se tiene en conseguir que su comportamiento y rendimiento sea lo más parecido al del ser humano, ha despertado en mí el deseo de investigar desde cero la creación de uno de ellos. En este trabajo se abarcará los primeros pasos básicos que se deben dar para su desarrollo dejando el trabajo abierto para futuros trabajos, ya que este es un mundo con infinidad de posibilidades, que evoluciona día tras día.

El estudio de estos sistemas robóticos puede tener además, otras aplicaciones como puede ser en la medicina, para casos como miembros amputados. El desarrollar un proyecto base que pueda servir como futuro trabajo en este campo, es otra de las motivaciones que encuentro.

Por otra parte, dado que este trabajo se desarrolla desde el grado de la ingeniería mecánica, con este proyecto quiero demostrar como en nuestro grado adquirimos una serie de conocimientos que nos permiten cruzar ciertos umbrales y enfrentarnos en campos menos conocidos a diferentes problemas, solventándolos y alcanzando nuestros objetivos.

Este trabajo, se convierte así en un reto personal, queriendo resolver así toda serie de problemas que me surjan en el desarrollo del mismo, un preludio de lo que ocurrirá en mi vida laboral de ingeniero, donde se esperará de mí; soluciones ante problemas, teniendo como herramienta mis conocimientos adquiridos hasta el momento.

Por último, a nivel universitario, una última motivación que señalar, la cual surgió cuando comenzamos con este proyecto. Durante las primeras reuniones que tuvieron lugar con el fin de planificar el trabajo, nos dimos cuenta de la ausencia de software de robótica en la Universidad de Zaragoza, no se imparten clases para su manejo, pero tampoco existe gran conocimiento de ellos entre los internos de la Universidad. Aquí nace la necesidad de conocer un programa de estas características y controlarlo con cierta soltura para poder explicarlo a otros investigadores para futuros proyectos de la Universidad, y lograr mayores avances en investigación.

1.3 Objetivos del proyecto y contexto de trabajo

Existía la necesidad de contar con un entorno de simulación robótica genérico, el cual pudiera ser una herramienta de utilidad para la creación de equipos de robots para distintos tipos de competencias y categorías. Como mínimo, se debía contar con la posibilidad de construir simulaciones que resultaran herramientas útiles al desarrollar robots. El simulador debería proveer una serie de funcionalidades que facilitaran la tarea de desarrollo y depuración de estrategias. Algunas de ellas se detallan a continuación:

- Creación genérica de robots: Contar con la posibilidad de modelar realidades muy diversas.
- Control de robots: Exponer alguna *interfaz de comunicación* mediante la cual se puede controlar los robots presentes en la simulación mediante estrategias implementadas por desarrolladores, a través por ejemplo, de motores conectados a ruedas.
- Manipulación del paso del tiempo de las simulaciones.
- Manipulación de los robots.
- Grabación y reproducción de estados de la simulación.

El **principal objetivo de este trabajo** es encontrar la plataforma virtual que nos permita desarrollar nuestro robot gracias al control de este software con habilidad. En este contexto, crearemos a XHO, un robot que se espera que cumpla una serie de conocimientos mecánicos implementados en el programa por nosotros, ya que lo que se espera es que esta realidad

"virtual" desarrollada en la interfaz llegue un día a implementarse en la realidad con buenos resultados, para ello debemos empezar a desarrollar el robot desde los conceptos más básicos, la cinemática, y desde aquí progresar en trabajos futuros de la mano de la mecánica y otras ramas de la ingeniería como son la electrónica y la informática.

Este proyecto pretende una serie de objetivos pero hay que ser conscientes que se va a desarrollar un estudio universitario básico desde un punto de vista cinemático ya que el modelado dinámico es un asunto complejo, con el cual, como se explica en el Anexo A "Historia de la robótica", empresas internacionales que invierten millones de euros en investigación cada año tienen graves problemas respecto a dos puntos clave:

- La locomoción bípeda y estable no está resuelta totalmente.

- No existen robots humanoides de tamaño humano en el mercado, ya que la complejidad del problema de una locomoción bípeda y estable crece exponencialmente con la altura y peso del humanoide. Por lo tanto, predominan los humanoides pequeños, ya que pueden llevar a cabo un movimiento de caminar complejo de forma más sencilla y, además, son más baratos y más fáciles de controlar.

1.4 Organización del proyecto

El presente documento se ha dividido en capítulos, los mismos se describen a continuación.

Capítulo 1 – Introducción: Este capítulo es introductorio a todos los demás capítulos.

Capítulo 2 – Estudio mecánico: Estudio del problema desde un punto de vista físico-mecánico

Capítulo 3 – Simuladores para robots: Se presentan diferentes software disponibles en el mercado para el diseño de robots, y explicamos las razones por las que hemos utilizado el simulador Webots en nuestro trabajo.

Capítulo 4 – Implementación del problema físico en Webots: Nos adentramos en el mundo virtual de Webots y basándonos en diferentes aspectos mecánicos diseñamos nuestro robot.

Capítulo 5 – Trabajos futuros: Capítulo que presenta posibles proyectos futuros, para estudiantes interesados en adentrarse en este mundo para posibles TFGs y Tesis.

Capítulo 6 – Conclusiones: Exponemos y analizamos todas las impresiones surgidas a lo largo de la elaboración del trabajo y las surgidas al finalizar este.

Capítulo 7 – Anexos: Colección de documentos que completarán la información del trabajo.

Capítulo 8 – Bibliografía: Listado en el que aparecen todos los documentos, que han servido de apoyo para la elaboración del proyecto.

2. Estudio mecánico. Marco teórico

En este apartado se muestran los antecedentes históricos de la locomoción bípeda, así como los conceptos necesarios para tener un panorama más amplio sobre esta disciplina de la biomecánica y como resolverla, con el fin de llegar a darle movimiento a nuestro robot.

2.1 Locomoción

Locomoción, la capacidad de un cuerpo para moverse de un lugar a otro, es una característica definitoria de la vida animal. Esto se logra mediante la manipulación del cuerpo con respecto al medio ambiente. En condiciones normales, la locomoción tiene muchas formas, ya sea el nadar de los peces, el volar de los pájaros o el caminar de los humanos. La diversidad de la locomoción animal es realmente sorprendente y extremadamente compleja. Lo mismo es cierto en los objetos elaborados por el hombre: los aviones tiene alas que los hacen ascender para el vuelo, los tanques tienen pistas para recorrer terreno irregular, los coches tienen ruedas para rodar eficientemente y los robots suelen ahora caminar sobre sus propias piernas en entornos con tierra discontinua, tales como un pendiente rocosa o los peldaños de una escalera, se puede argumentar que el medio más adecuado y versátil para la locomoción son las piernas.

Las piernas permiten la facilidad de apoyo en el medio ambiente al pasar por encima de las discontinuidades. Además, las piernas son una opción obvia para la locomoción en entornos creados para la marcha humana: correr y trepar.

Los robots bípedos forman una subclase de robots con patas. En el aspecto práctico, el estudio de la locomoción con patas mecánicas ha sido motivado por su uso potencial como medio de locomoción en terreno accidentado, o ambientes con discontinuidades. También hay que reconocer que gran parte del interés actual en robots con patas se deriva de la existencia de máquinas que operan en formas antropomórficas o de forma animal (tenemos en cuenta varios bípedos conocidos y juguetes cuadrúpedos). La motivación para el estudio de robots bípedos, en particular, surge de diversos intereses sociológicos y comerciales, que van desde el deseo de reemplazar a los humanos en ocupaciones peligrosas (remoción de minas, la inspección de las centrales nucleares, las intervenciones militares, etc.), a la restauración del movimiento en las personas con discapacidad (controlado dinámicamente los miembros inferiores, prótesis robótica de rehabilitación y estimulación neural funcional).

El proceso de caminar es mucho más complejo de lo que parece ya que para dar pasos están involucrados pies, cadera, torso, brazos, hombros, cabeza, etc. Tomando en cuenta el objetivo principal de este proyecto, conocer los principios básicos que forman parte de la caminata humana es fundamental además de constituir una base para dar pie al modelo dinámico del robot bípedo.

2.1.1 La marcha bípeda y la teoría del equilibrio bípedo

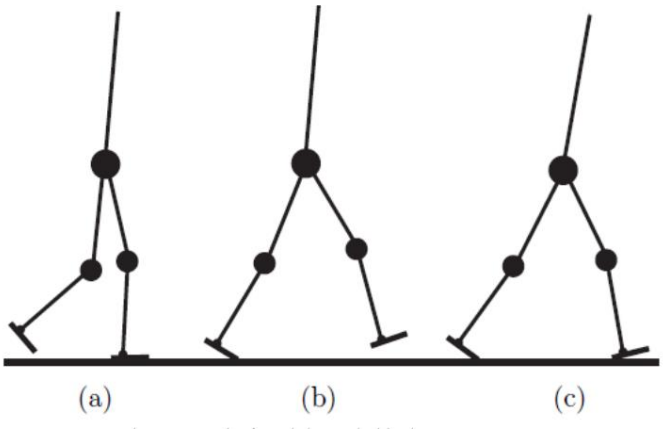


Figura 1. Varias fases de la marcha bípeda

Como se muestra en la Figura 1 la fase de Soporte Simple (FSS) (también llamada fase de simple apoyo o fase de oscilación) se muestra en (a) y (b), mientras que una fase de doble soporte (FDS) o de doble apoyo se representa en (c). La caminata comienza con los dos pies extendidos y sobre el suelo, en donde el equilibrio no es muy significativo, el gran problema comienza al levantar uno de los dos pies para realizar el movimiento de simple apoyo ya que la tendencia del bípedo a caer es alta, para evitar que el robot bípedo caiga se deben realizar correctivos a los movimientos del robot permitiendo así la estabilidad dinámica de la caminata.

El estudio de la caminata bípeda es un tema muy amplio, relacionado de manera directa con la dinámica del robot. En el Anexo B “modelo dinámico” se explican conceptos básicos para entender este complejo campo, introduciendo así el proyecto a futuros trabajos.

2.2 Métodos de Cálculo. Modelo cinemático

La cinemática es la rama de la física que estudia las leyes del movimiento de los objetos solidos sin considerar las causas que lo originan, las fuerzas, y se limita, principalmente, al estudio de la trayectoria en función del tiempo. Para ello, utiliza la velocidad y la aceleración, que son las dos principales magnitudes que describen como cambia la posición en función del tiempo.

Dos son los métodos de los que disponemos para desarrollar el modelo cinemático, la cinemática directa y la cinemática inversa.

La cinemática directa, se refiere a ciertos ángulos dados y los parámetros de los elementos del sistema para encontrar las posiciones y orientaciones deseadas de los elementos. Mientras que la cinemática inversa desde un punto de vista analítico se refiere a dada la posición y orientación del efector final de un robot así como sus parámetros de articulación y elementos, encontrar los ángulos de articulación correspondientes del robot de manera que se pueda posicionar como se desee el efector final. (Figura 2).

La cinemática inversa consiste en determinar los valores de las articulaciones que satisfagan condiciones deseadas de posición, velocidad y aceleración en el espacio cartesiano. El

problema que presenta la cinemática inversa es que presenta sistemas de ecuaciones no lineales debido a que cada ecuación depende de las variables de cada GDL además de ser un sistema sobredeterminado porque tenemos más ecuaciones que incógnitas.

Para este trabajo utilizaremos la **cinemática inversa**, debido a que basamos nuestro estudio en la rama de la biomecánica, y queremos imitar el caminar de una persona, encontrar el valor de la articulación para una posición deseada, es decir cinemática inversa. Pero antes de enfrentarnos al problema de las ecuaciones no lineales y tener que llegar a "linealizar" las ecuaciones mediante el uso de algún tipo de método iterativo como puede ser el método matemático de Newton-Rhapson, optaremos por ayudar a la resolución analítica con la resolución gráfica, para ello usaremos el software de diseño AutoCad.

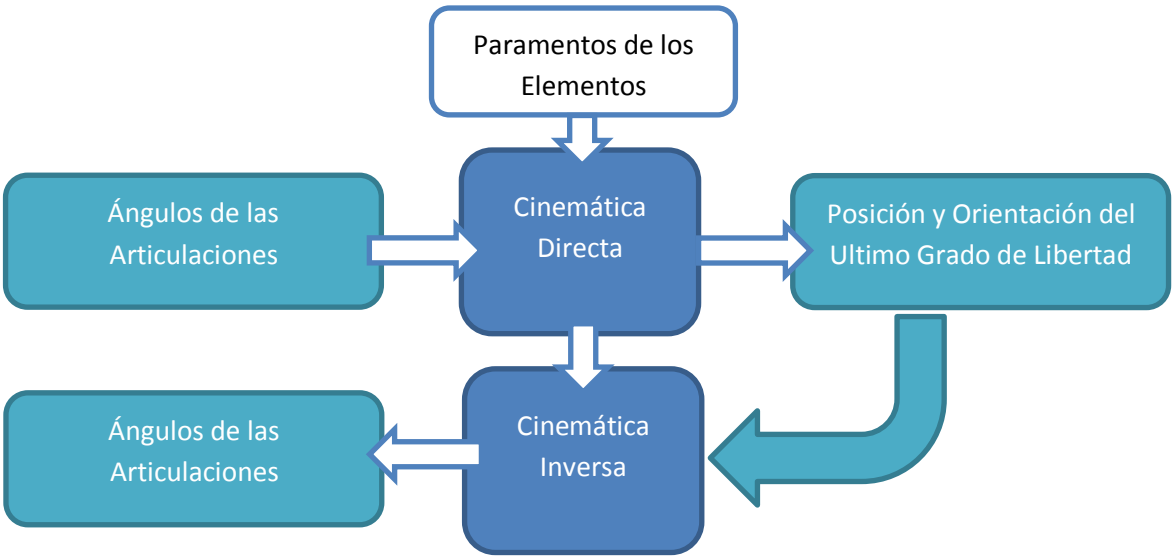


Figura 2. Esquema cinemática directa e inversa.

A priori, la cinemática puede parecer de poca aplicación en un modelo que se quiere conseguir que interactúe en un entorno real, debido a que desprecia las fuerzas presentes en cualquier situación en el mundo que nos rodea, pero no es así. La cinemática asienta la base de la dinámica, y describe las trayectorias que describirán el movimiento, las cuales serán modificadas en el estudio dinámico para soportar las diferentes fuerzas que se efectúan sobre un cuerpo en un entorno real.

Por esto el modelo cinemático se restringe al estudio virtual inicial, este modelo es el que se va a desarrollar en este trabajo. Se desarrollará el modelo cinemático en base a una serie de restricciones mecánicas y este modelo se aplicará a un robot virtual en un software a estudiar, obteniendo unos resultados que serán trayectorias de cuerpos, en concreto, lo que se busca es obtener la trayectoria de marcha del tren inferior de un robot bípedo.

"En la medida en que una máquina equipada con dos piernas puede imitar la marcha de un ser humano, mejor puede interactuar con él en su medio ambiente"

Westervelt, Grizzle, Chevallereau & Ho Choi.

Esta cita recoge la idea principal de nuestro trabajo. Dos piernas son un elemento extraordinario para superar cualquier terreno con cualquier tipo de irregularidades. Hasta ahora los robots han sido dotados con ruedas para su transporte, pero estas ruedas están limitadas a ciertos entornos. Por esto surge la necesidad de atribuir a los robots de extremidades como las de las personas que les permitan llegar a lugares a los cuales hasta ahora no podían acceder. Las personas, al igual que los animales terrestres, están dotadas de extremidades, en el caso de los seres humanos, piernas, estas piernas dotadas de un gran número de músculos y terminaciones nerviosas, permiten superar infinidad de obstáculos. Aquí reside la necesidad de imitar el movimiento humano, esta imitación se consigue gracias al dibujo de trayectorias proporcionado por el modelo cinemático.

Es verdad que para lograr el movimiento de las piernas de los seres humanos, nuestro robot debería estar dotado de una enorme cantidad de actuadores que le proporcionara un gran número de grados de libertad, un asunto que tiene sus límites, debido a la arquitectura actual de los actuadores y toda la serie de complementos que requiere la cual no cabría en los habitáculos dispuestos en las extremidades para ello. Por esto y porque queremos estudiar los asuntos básicos en un robot básico, buscaremos aquel robot que con el menor número posible de grados de libertad satisfaga nuestras necesidades.

2.3 Modelo dinámico

El modelo dinámico, es aquel que desarrollará el robot bajo un comportamiento dinámico, siendo la dinámica esa rama de la física que describe la evolución en el tiempo de un sistema físico en relación con los motivos o causas que provocan los cambios de estado físico y/o estado de movimiento.

Lograr la implementación del modelo dinámico en el robot, implica conseguir un comportamiento virtual que se asemejaría al del mundo real, es decir, conseguir un correcto modelado dinámico implicaría el hecho de que nuestro robot se comportaría en el estado virtual de la misma manera que se comportaría en un entorno real. Pudiendo así pasar al siguiente paso, invertir económicamente en una estructura física ya que tendríamos la seguridad de que el modelo no va a fallar en la realidad, todo esto es gracias al software utilizado y a una base mecánica correctamente aplicada.

Pero el desarrollo del modelo dinámico no es sencillo, antes hay que desarrollar todo un modelado cinemático, y una vez concluido este, desarrollarlo con una serie de factores más a tener en cuenta como son cargas, factor equilibrio y otra serie de aspectos desarrollados en el Anexo B, donde se explica de manera teórica como desarrollar el modelo y factores a tener en cuenta.

3. Simuladores para robots

Cuando se realiza un estudio completo para construir un robot humanoide es necesario el uso de la simulación mediante algunos simuladores para poder evaluar el funcionamiento del robot una vez diseñado y así evitar los posibles fallos, colisiones y desajustes con el sistema robótico real montado. El principal motivo para realizar la simulación es la prevención de fallos y posibles daños en los mecanismos del robot y así ahorra el coste del diseño.

En la actualidad existe una gran cantidad de herramientas que permiten realizar simulaciones de robots en un ambiente de tres dimensiones (3D), disponen diversos sensores, actuadores y objetos. Los sensores que se encuentran en estas herramientas son capaces de generar una realimentación entre la interacción del robot con el entorno, el cual debe ser creado también por el usuario. Además, la interacción entre los objetos creados se realiza mediante modelado, teniendo en cuenta la física del sólido rígido.

La mayoría de simuladores utilizan interfaces gráficas para la construcción de los robots y el ambiente en el cual interactúan, y todas admiten diferentes lenguajes para la programación de los controladores, tales como PHYTON, Java, C /C++, etc. A continuación, comentamos los simuladores principales en el mercado.

3.1 SimRobot

Este simulador no está limitado a una clase en especial de robots móviles. El lenguaje utilizado en este simulador está basado en XML, en donde los usuarios son libres de especificar cualquier robot y sus ambientes completamente sin la necesidad de otros lenguajes de programación. Diferentes partes del cuerpo de un robot, actuadores y múltiples sensores permiten la composición libre de un robot. Para simular la dinámica de cuerpos rígidos, usa Open Dynamics Engine (ODE) y la visualización de las imágenes está basada en OpenGL.

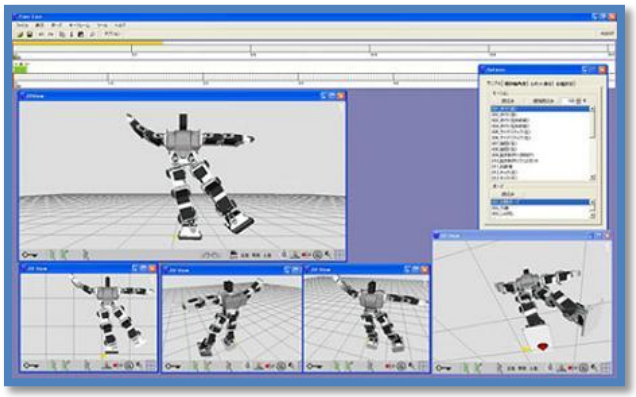


Figura 3. Simulador SimRobot

3.2 Gazebo

Gazebo es un simulador 3D, que permite cargar diferentes tipos de robots, sensores y actuadores dentro de un mundo artificial. Gazebo ofrece las primitivas para leer de los sensores e interactuar sobre sus actuadores. También se usa ODE y OpenGL para la simulación de cuerpos rígidos y la visualización de las imágenes.

Sus características principales son que dispone el modelo de cámara estéreo, genera mapas estéreos de profundidad de la imagen y el interfaz gráfico de usuario escrito en wxPython se puede usar sin necesidad de instalar el servidor Player.

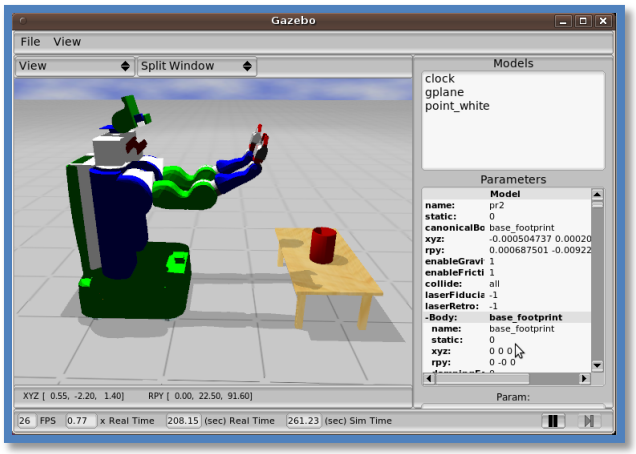


Figura 4. Simulador Gazebo

3.3 OpenHRP3

Indudablemente el “Open Architecture Humanoid Robotics Platform 3” (OpenHRP3) es un proyecto muy ambicioso y de gran alcance.

La descripción de los modelos del robot y del entorno se realiza con el formato VRML. Permite la comprobación de colisiones. También tiene una función para la implementación de diversas leyes de control del robot “Controller”. Existe una función independiente para soportar la dinámica de los mecanismos robóticos y cuenta con una función de visualización de la simulación. Finalmente, dos de los módulos fundamentales son el de planificación del movimiento “MotionPlanner”, que genera trayectorias libres de colisiones y el de generación del patrón de paso “PatternGenerator” que genera movimientos estables para la locomoción del robot basándose en el control del ZMP. Englobando todas esas funcionalidades el OpenHRP3 cuenta con un entorno integrado de simulación que podemos ver en la Figura 5.

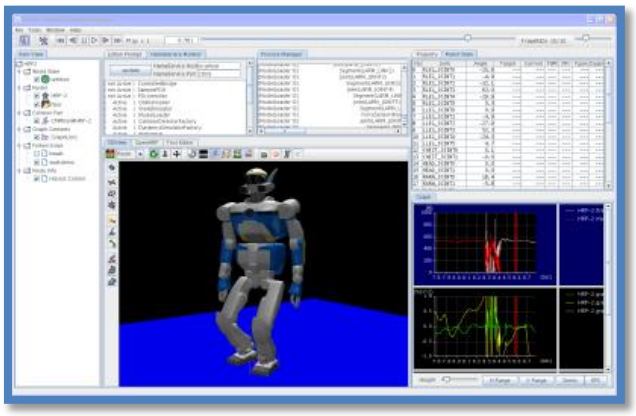


Figura 5. Simulador OpenHRP3

3.4 Marilou Robotics Studio

Marilou Robotics Studio es un modelador y ambiente de simulación 3D para robots móviles, humanoides, brazos articulados y robots paralelos que funcionan en condiciones verdaderas respetando las leyes de física.

En un ambiente realista gráfico, Marilou nos permite crear la jerarquía necesaria para construir y probar ensambles de formas simples (cajas, esferas, cilindros, superficies, redes de elevación) y formas complejas, tales como triángulo de malla, con geometrías y formas convexas, posee una simulación en tiempo real o simulación acelerada.

La programación del robot utilizando varios idiomas C, C++ y C#. en Windows y Linux.

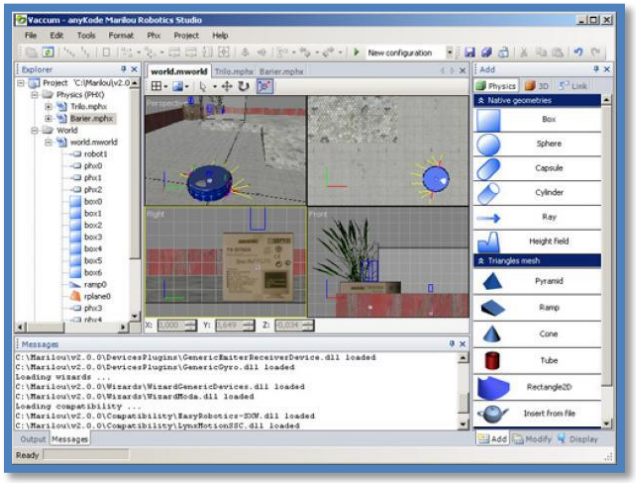


Figura 6. Simulador Marilou Robotics Studio

3.5 Microsoft Robotics Developer Studio 4

Es un entorno basado en Windows para el control robótico y simulación. La simulación realística está provista por el motor PhysX de AGEIA, el cual posibilita la emulación por software y la aceleración por hardware.

Es una herramienta de programación visual para crear y depurar aplicaciones robóticas. El desarrollador puede interactuar con los robots mediante interfaces basadas en web o en Windows.

La aplicación es una multi-plataforma robótica en donde se permiten varios lenguajes como Visual C#, Visual Basic .NET, JScript, IronPython y lenguajes de terceras partes que se adecuen a la arquitectura basada en servicios. El desarrollador puede acceder fácilmente a los sensores y actuadores de los robots, el simulador proporciona una librería de implementación de concurrencia basada en .NET. La comunicación está basada en mensajes, permitiendo la comunicación entre módulos.

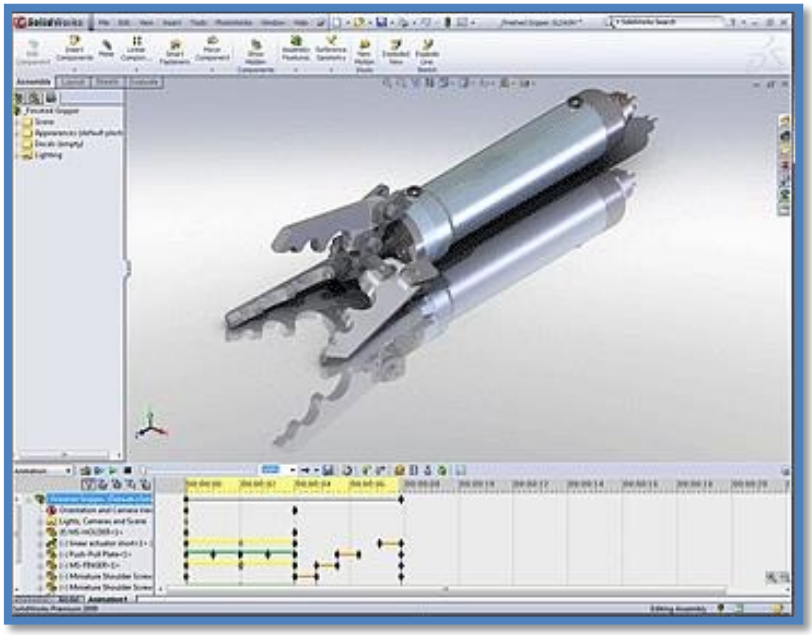


Figura 7. Simulador Microsoft Robotics Developer Studio

3.6 Webots

Webots es un paquete de software profesional para modelar, programar y simular robots móviles. Con este software, el usuario puede diseñar configuraciones complejas de robots, en un entorno virtual en 3D. El usuario puede elegir las propiedades de cada objeto, como forma, color, textura, masa, fricción, etc.

Por otra parte, los robots pueden ser equipados con un amplio número de sensores y actuadores, tales como sensores de distancia, ruedas motrices, cámaras, servomotores, sensores de contacto y fuerza, emisores y receptores de señales de radiofrecuencia, etc. Finalmente, el usuario es capaz de programar cada robot individualmente para que exhiba el comportamiento deseado.

Además, los controladores de los robots se pueden programar en un entorno de desarrollo externo, o en el que hay disponible en Webots. El comportamiento del robot puede ser

testeado en mundos con física realística. Y los programas de los controladores se pueden trasladar a robots físicos reales.

3.7 Conclusión. Software seleccionado Webots

Tras estudiar los diferentes Software presentados, decidimos que el mejor entorno de trabajo será Webots, este software brinda un mundo lleno de posibilidades. Pues de entre todos los estudiados es el que presenta mayor versatilidad y potencial. Por esta razón, para este proyecto hemos conseguido la versión 6.44 de este software. De este modo, estudiaremos a fondo el programa.

Producto del estudio del entorno Webots, editamos una guía que servirá como base teórica para el que quiera estudiar este software. Este documento se encuentra disponible al final de este trabajo en el Anexo C.

4. Implementación del problema físico en Webots.

En este capítulo se detallarán los pasos seguidos en la plataforma Webots en base a la mecánica, con el objetivo de modular a XHO, un robot bípedo que simule el movimiento de caminar, teniendo siempre presente que se tratará de un robot que se pretende que sirva de base para estudios posteriores .

La primera decisión que se toma es diseñar sólo el tren inferior del robot, ya que con dos piernas es suficiente para modular el proceso "caminata", por ello nuestro robot constará de dos piernas y una cadera que realizará la función de nexo entre las dos. Otra decisión que se toma antes de comenzar a desarrollar en Webots, trata del número de grados de libertad que tendrá la máquina. De esta manera, tras estudiar las diversas opciones, se ha concluido que el modelo básico el cual puede cumplir nuestros objetivos, estará formado por un total de 6 grados de libertad, 3 en cada pierna, que permitirán el giro en el eje x. Estos giros se darán por cada pierna en la cadera, en la rodilla y en el tobillo. Estos grados de libertad harán posible que el robot camine de manera recta, sin giros. Nuestro objetivo entonces, será lograr la modulación de XHO en Webots y lograr que realice los movimientos de la caminata desde un punto de vista cinemática.

4.1 Construcción del modelo.

El robot estará formado por dos piernas y una cadera que funcionará como nexo de ambas extremidades, como se ha señalado antes y cada pierna tres grados de libertad, que permitan el giro en el eje x.

El primer problema es permitir el giro de las diferentes articulaciones que conforman el sistema, para permitirlo, conformamos un sistema formado por 6 cilindros, los cilindros sustituyen al futuro servomotor que irá alojado en esa cavidad, que es lo que genera el movimiento de giro controlado. Por otra parte, tendremos el conjunto de piezas, que completarán el modelo, estas piezas son; el pie, la pierna inferior (nexo entre el servomotor tobillo cuya identificación será "C" y el servomotor rodilla identificado como "B"), la pierna superior (nexo entre "B" y servomotor de la cadera "A"), el conector pierna-cadera (diseñado para permitir el giro de la pierna respecto de la cadera), y finalmente la cadera, excepto el elemento cadera, el resto de elementos están duplicados para obtener las dos extremidades.

Las dimensiones que se han atribuido han sido diseñadas de manera que el robot tenga una altura total aproximada a la de un ser humano. De este modo, si se fabrica el robot, habrá más espacios para en un futuro, poder implantarle sistemas de sensores y de movimiento para mejorar su eficiencia. El tema del dimensionamiento no es arbitrario ya que si se hace mucho

mayor, es verdad que tendremos más espacios, pero también aparecerán más problemas parar resolver el problema dinámico. Por eso tras modelar las diferentes partes del robot, se ha obtenido un tren inferior con una altura total de 866 mm.

Diseñado en un primer boceto el robot, se utilizará el software de diseño mecánico Autodesk Inventor, antes de crearlo en Webots, con el fin de comprobar que el sistema es mecánicamente correcto, en cuanto a los grados de libertad diseñados para atribuirle el problema cinemático, que será implementado en Webots.

4.1.1 Construcción del modelo en Autodesk Inventor

Previo a la construcción del modelo en Webots, construiremos el modelo mecánico en Autodesk Inventor, un software lleno de posibilidades para el diseño y estudio de sistemas mecánicos, dónde comprobaremos que el sistema es mecánicamente correcto, ya que una vez ensamblado todas las partes que componen la máquina se observará los grados de libertad que tiene el sistema, una de las muchas posibilidades que tiene el programa.

Este modelado nos sirve también para obtener los planos del robot, que hemos realizado previamente en papel, por una parte, y por otra, para obtener un primer modelo que desarrollar en el futuro.

Comentar que este trabajo que se ha realizado en Inventor no contempla el estudio de un ensamblaje mecánico correcto, ya que no era necesario para el proyecto que estamos tratando, pero habrá que tener en cuenta este tema en proyectos futuros.

De este modo, obtenemos el siguiente resultado en Autodesk Inventor, en el Anexo F, se pueden ver los planos de cada una de las piezas que compone el sistema mecánico que conforma el robot y un plano del robot ensamblado.

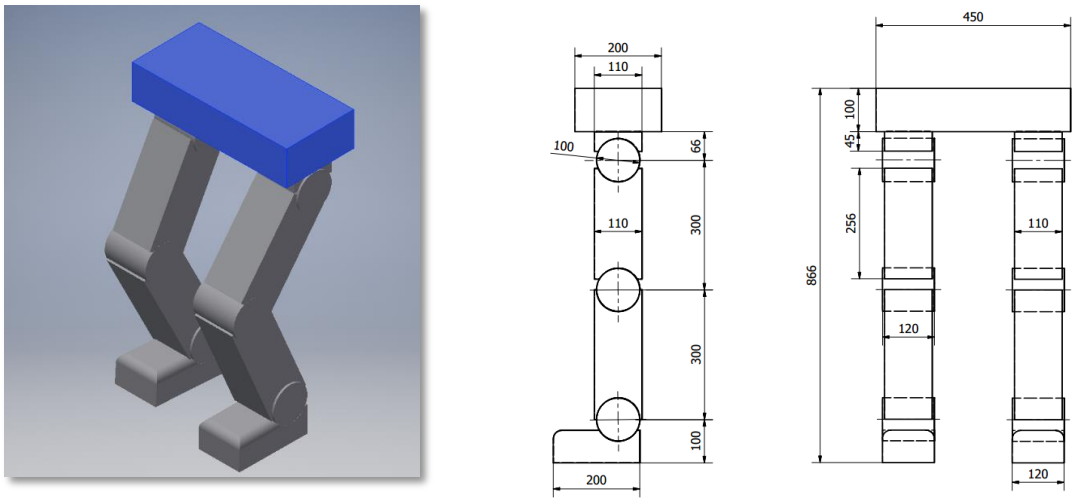


Figura 8. Robot trabajando con 6 GDL en Autodesk Inventor (izda). Robot con dimensiones producido en Autodesk Inventor (dcha).

El proceso de implementación del modelo en Inventor se realiza creando cada pieza de manera independiente, cuando se tienen todas las partes del robot, se ensamblan dando lugar al resultado final. Para la creación de cada pieza, realizaremos la pieza en el plano y luego utilizaremos la herramienta extrusión, para conseguirla en tres dimensiones, para las cavidades utilizaremos la herramienta agujero. Las piezas no deben ser creadas en ningún orden concreto, el resultado del ensamblaje no dependerá del orden de creación. Esta última puntualización, es interesante ya que como veremos a continuación, en el software Webots, el orden de la creación de las piezas sí que importa.

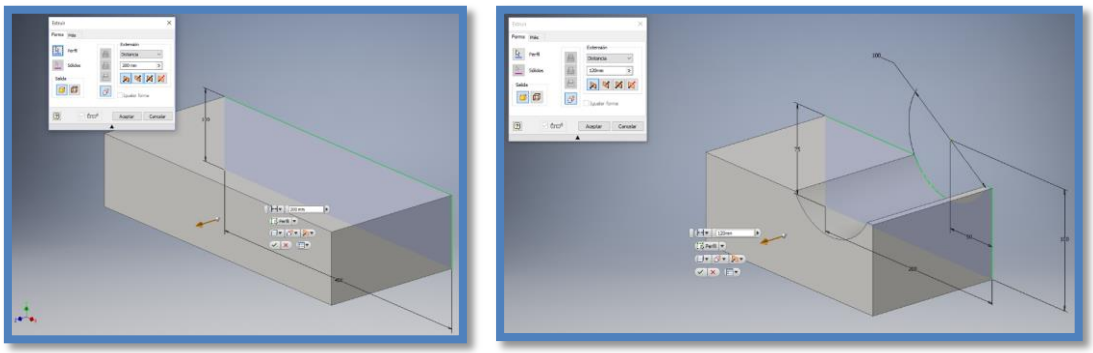


Figura 9. Extrusión de la pieza Cadera (izda). Extrusión de agujero en la pieza Pie (dcha).

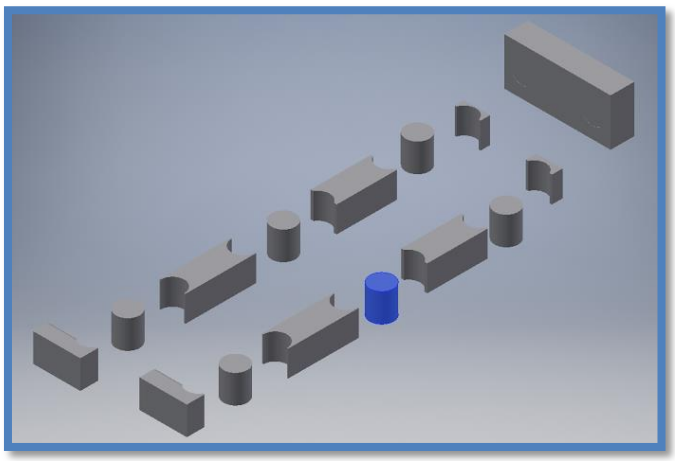


Figura 10. Todas piezas terminadas dispuestas para el proceso de ensamblaje.

4.1.2 Construcción del modelo en Webots.

Tras tener un mecanismo correcto modelado en Inventor, creamos el mismo sistema en Webots. Una vez en este software tenemos más libertad a la hora de modular ya que, en este programa no importa tanto los acabados mecánicos como era en el caso anterior.

En este software nuestras preocupaciones, será preservar el dimensionamiento general del robot, y establecer las jerarquías y funcionalidades, que permitirán escribirle al robot un código que permita simular la tarea “Caminata”.

Un punto muy importante de este capítulo, es la modificación que se ha debería realizar al concepto del robot, explicado hasta el momento:

Nuestro robot se quiere que funciones con 6 grados de libertad, los cuales serán los encargados de efectuar el movimiento de las piernas. Pero, para resolver el problema cinemático en Webots de la caminata será necesario un grado de libertad más. El motivo es que una persona cuando anda, lo hace gracias a que la fuerza que ejerce el pie sobre el suelo, provoca una reacción en el pie (Tercera ley de Newton: Principio de acción-reacción), que hace que el cuerpo se desplace en el eje z, en nuestro caso se desplazaría la cadera. Al hablar de fuerzas, estaríamos haciendo referencia a un problema dinámico el cual no contemplamos en este trabajo. Este problema de la reacción del suelo, es uno de los mayores problemas que se encuentran en los grandes proyectos que estudian el caso dinámico en la realidad. La solución de este problema, sería añadir un Servo (1GDL) lineal en la cadera, que provoque el movimiento del robot en el eje z, sin este Servo, el robot realizaría los movimientos sin avanzar en el espacio a lo largo del eje z.

El motivo por el que no es necesario un acabado detallado de las piezas, se debe a que al estudiar el problema cinemático, no nos preocupa ni la gravedad, ni masas, ni las distintas fuerzas que pudieran interactuar desde el entorno, por este motivo podemos no activar el campo “BoundingObject”. Este campo es el responsable de dar unas características físicas, entre ellas el volumen y masa, por lo tanto si no hay volumen físico no hay problema de contactos entre las piezas, pudiendo superponerse.

El resultado de la modulación de Webots lo podemos ver en la figura 11. El siguiente paso en Webots será escribir el código que haga posible el movimiento de caminar del robot.

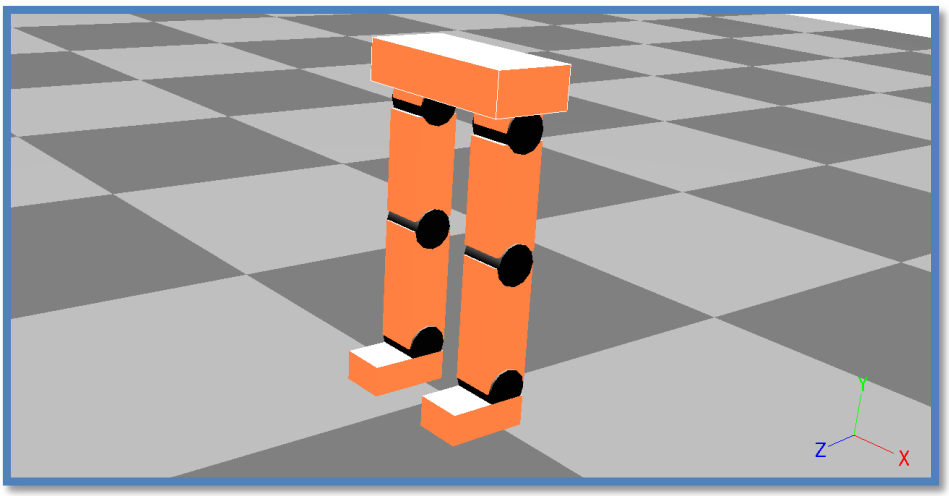


Figura 11. Robot XHO modelado en el software Webots

La creación del modelo en Webots no se realiza como se realizaría en Inventor donde se realizaban todas las piezas y se ensamblaban.

En Webots la creación sigue un orden, una jerarquía, una estructura de padres e hijos. Tal y como se ve en la figura 12, debido a que los objetos en Webots se conforman mediante nodos, estos nodos son descritos utilizando el lenguaje VRML, donde un nodo es la estructura mínima indivisible de un fichero VRML y tiene como misión la de definir las características de un objeto o bien las relaciones entre distintos objetos. (Más información en el Anexo C).

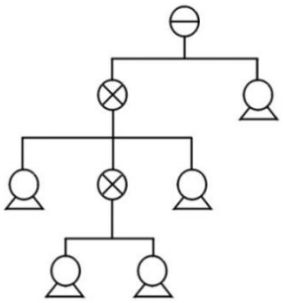


Figura 12: Estructura jerárquica de un mundo virtual en VRML

De esta forma, empezamos creando la cintura y continuamos hacia los pies, así se crea una serie de dependencias, donde los hijos dependen de los padres, y es así como luego podremos controlar diferentes elementos ("servos") para que el robot reproduzca un determinado movimiento.

A partir del conocimiento de la estructura jerárquica del árbol de nodos, explicar, que para crear nuestro robot en Webots, emplearemos como nodo padre el nodo "Robot", este nodo es utilizado ya que posee entre otros el campo "controller"(controlador), este campo consiste en un archivo binario, el cual es usado para controlar a un robot que se ha descrito en un archivo "world". Los controladores pueden ser de diferentes tipos; C, C++, Java. En este trabajo utilizaremos lenguaje C.

El nodo padre "Robot" va a jerarquizar una estructura conformada por nodos "Transform" y nodos "Servo". En primer lugar el nodo "Transform", se utiliza ya que se trata de un nodo de agrupación que define un sistema de coordenadas de nodo hijo respecto al sistema de coordenadas del nodo padre, a este nodo "Transform" se le establecerá un nodo hijo "Shape", encargado de dar forma, el cual tiene dos campos, el campo "Appearance" (apariencia) y el campo "Geometry" (geometría), que utilizaremos para crear objetos en el mundo. Por esto nos será tan útil el nodo "Transform" ya que con él crearemos objetos, pudiendo controlar perfectamente su posición gracias al control de los sistemas de coordenadas (campo "Translation"). La estructura jerárquica que presentará el nodo "Transform" a lo largo de nuestro trabajo, puede verse en la figura 13.

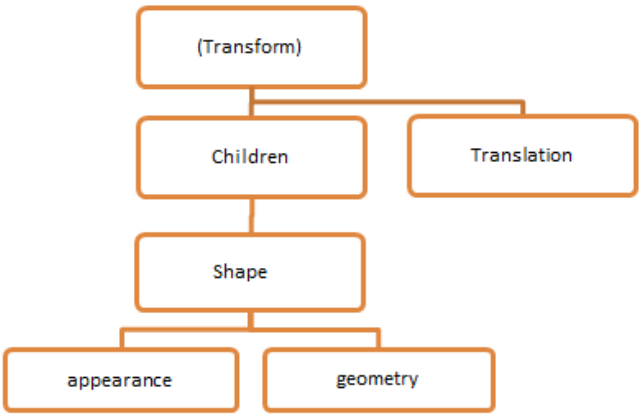


Figura 13. Estructura árbol del nodo "Transform" en Webots.

En segundo lugar, el nodo "Servo" es utilizado porque sirve para para añadir una 1 grado de libertad (GDL) en una simulación mecánica. El movimiento de servo puede ser de tipo "rotacional" o "lineal". Un servo rotacional se utilizaría para simular un movimiento de rotación, como un servomotor o una bisagra mientras que, un servo lineal se utilizaría para simular un movimiento de deslizamiento, como un motor lineal, un pistón, un cilindro neumático, un resorte o un amortiguador. En este proyecto utilizaremos el "Servo Rotacional" para generar el desplazamiento angular en el eje x, que conforman los 6 grados de libertad del robot. Este "Servo" realiza la función que realizaría un servomotor en la realidad para darle movimiento a la máquina.

Una vez presentados los nodos empleados, presentamos a continuación en las Figuras 14 la jerarquía que presenta nuestro robot en el mundo VRML de Webots.

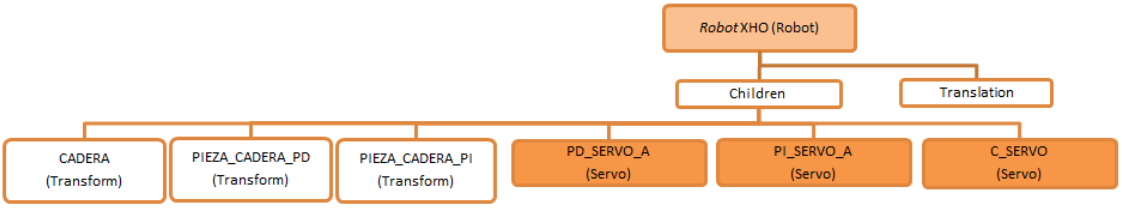


Figura 14. Estructura árbol del nodo padre Robot.

En la Figura 14 vemos el nodo padre "Robot XHO", formado por 6 nodos hijos, 3 nodos "*Transform*" que relacionaremos con tres cuerpos geométricos, por lo señalado antes, y otros tres nodos *Servo*, los cuales explicaremos a continuación. "Robot XHO" tiene un campo "*Translation*", este campo es modificado para situar al *Robot* respecto del sistema de coordenadas globales de Webots de esta manera todos sus nodos hijos deberán situarse entendiendo por el centro de sistema de coordenadas, el fijado por el nodo padre.

Como se ha explicado antes, la creación del robot se realiza en sentido descendente, desde la cadera hacia pos pies, por ello situamos en el centro del robot, el centro de la cadera.

El nodo Padre está definido tres nodos “Transform” y tres nodos “Servo”, donde el “Servo C_SERVO” servirá para resolver el problema explicado anteriormente de la traslación del robot en el eje z. Los nodos “Transform” dan forma a la cadera, mientras que los “Servo”, generarán un movimiento, rotacional de los cuerpos generados en su nodo. La figura 15 muestra los “Transform” del nodo Padre, cuyos “Shape” tienen por geometría una caja con unas determinadas dimensiones (“geometry Box”) mientras que la Figura 16, presenta la jerarquía de nodos hijos del nodo Servo “PD_Servo_A” que hace alusión a la pierna derecha (PD), la creación de la pierna izquierda (PI) será idéntica.

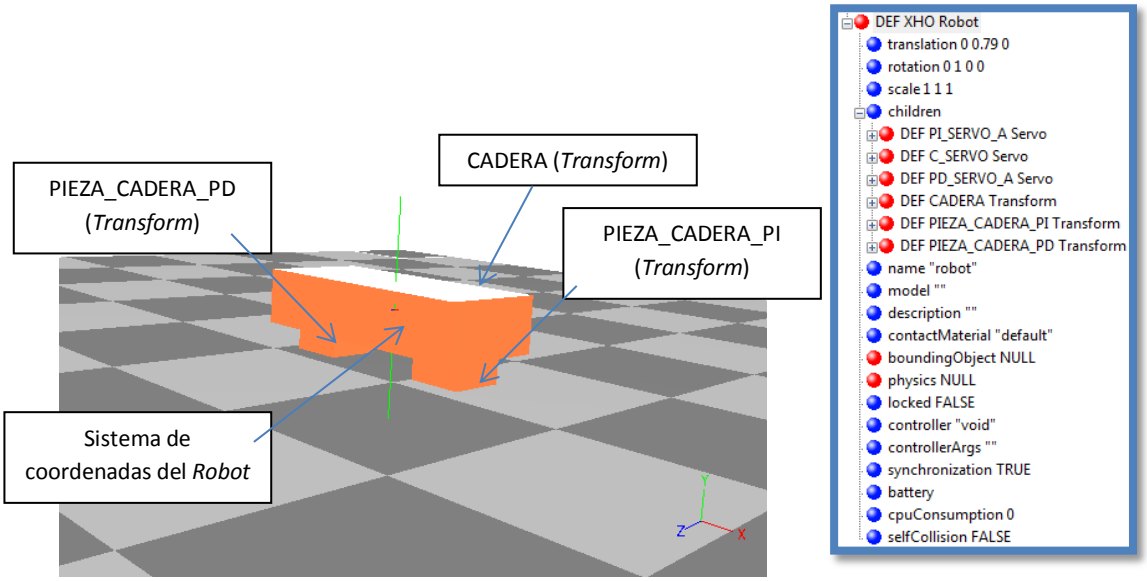


Figura 15. Representación gráfica de los primeros nodos “Transform” creados en el modelo.

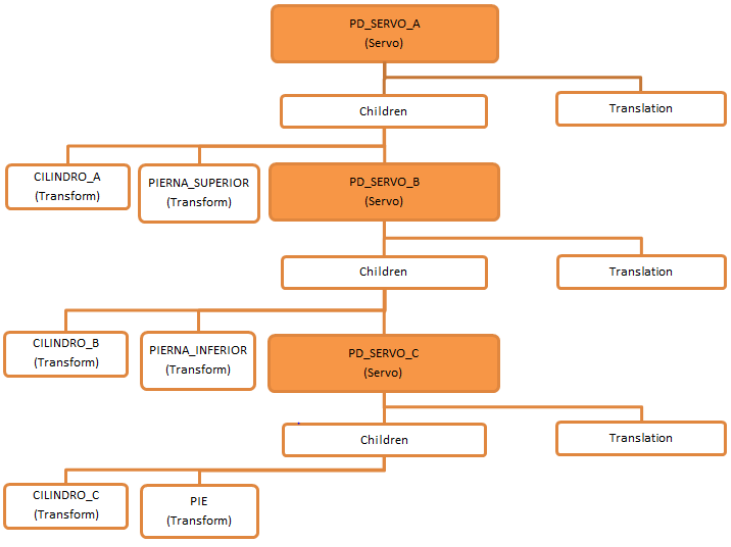


Figura 16.Estructura árbol del nodos padre “PD_SERVO_A”.

En la Figura 16, vemos como cada "Servo" tiene su sistema de coordenadas, por lo que el nodo hijo deberá establecer el sistema de coordenadas respecto de su padre.

Las casillas cuya función es la de nodo "Servo", están coloreadas, ya que su importancia es mayor, tienen un campo llamado "name", este campo lo nombraremos como hemos llamado al nodo por ejemplo, en el caso del Servo "PD_SERVO_A" el campo "name" lo nombraremos "PD_SERVO_A" la importancia de completar este campo se debe a que cuando escribamos el código las llamadas las haremos al "name" del "Servo". De esta manera controlaremos el robot.

Con la intención de que el Servo tenga una apariencia física creamos un "children" (hijo) "Transform", con forma de cilindro ("Shape Cylinder"), cilindro cuerpo geométrico que permite giro respecto un eje.

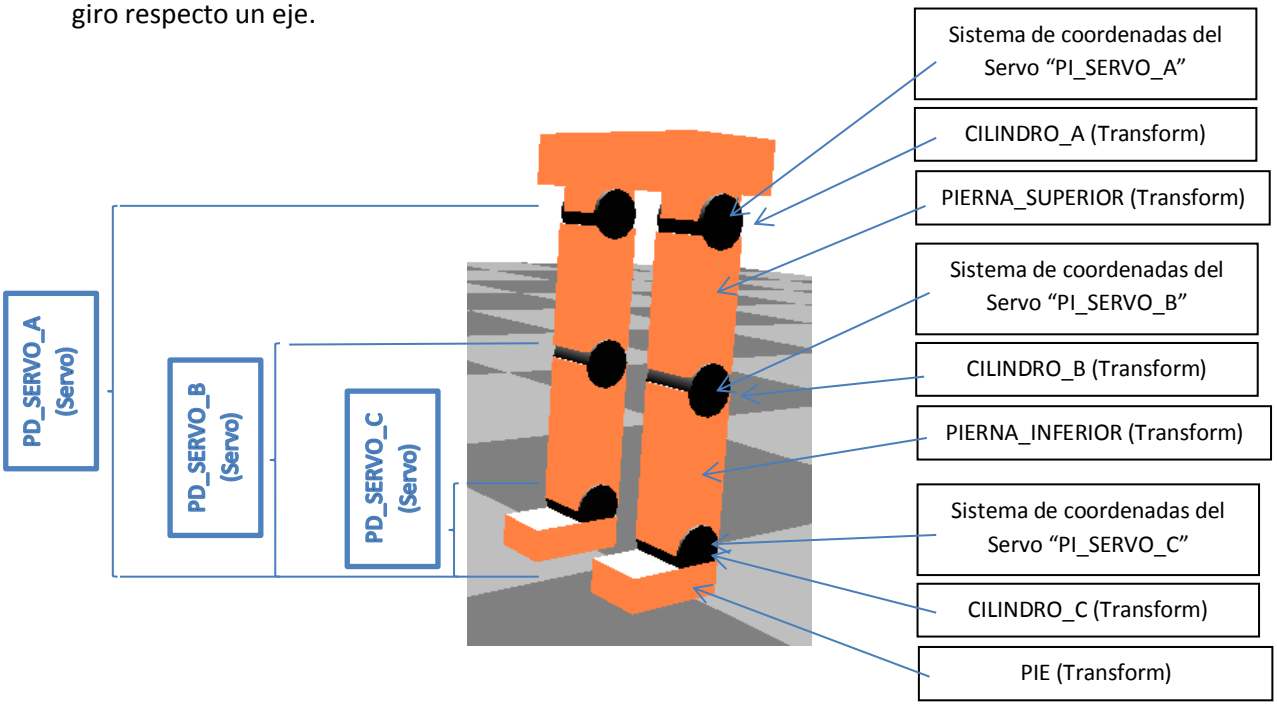


Figura17. Zona pilotada por cada "Servo", gracias a la jerarquía padre-hijo.

4.2 Programación

En este apartado presentamos el código escrito en lenguaje C, acompañado de explicaciones puntuales que permitan entender el código.

En la ventana "Editor" de Webots se editará el movimiento que debe realizar un "Servo", ya sea movimiento lineal o rotacional, con el objeto de mover un elemento o un conjunto de elementos, dependiendo de la jerarquía establecida.

Antes de escribir el código, deberemos editar los diferentes campos necesarios del "Servo", los campos de interés en este asunto, son el campo "name", el campo "type" y por último el campo "rotation". En el campo "name" escribimos el nombre con el cual llamaremos al "Servo" desde el editor, en nuestro caso coincidirá con el nombre con el que hemos definido al "Servo", aunque no tendría por qué ser el mismo. El campo "type" diferencia si el "Servo"

genera un movimiento rotacional (“rotation”) o un movimiento de traslación (“lineal”), en nuestro caso todos los “Servo” son rotacionales a excepción del “Servo C_SERVO” (cadera del robot)

Finalmente el campo “rotation”, tiene gran interés en nuestro estudio ya que pese a haber hecho un cilindro, esto solo es un cuerpo geométrico que utilizamos para darle un aspecto a un elemento invisible como es el “Servo”, pero el cilindro no implica que realice el giro respecto su eje geométrico, el sentido del giro se debe activar en “rotation” activando el eje sobre el cual se realice el giro.

Para poder escribir el código de la tarea, todavía hay que realizar otra tarea más. Esta tarea será la de calcular los ángulos que queremos que mueva cada “Servo”.

Gracias a la cinemática inversa resolveremos la incógnita de los ángulos, para ello seleccionamos un proceso de caminata bípeda, de los que se estudian en la rama de la biomecánica, con el objetivo de imitarlo. En este caso intentaremos imitar el desplazamiento de la figura 13.

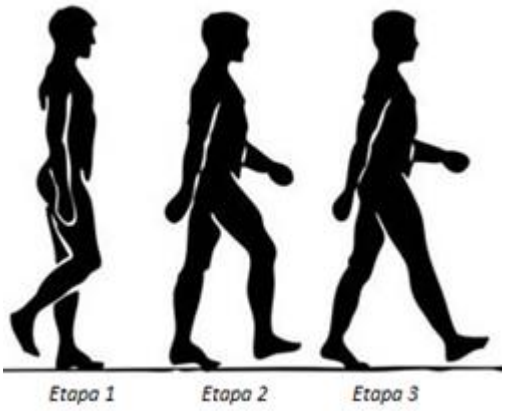


Figura 18. Estapas de la caminata bípeda.

En base a esta figura, donde vemos una caminata estándar de una persona, nosotros crearemos unos sistemas gráficos a estudiar. Estableceremos seis etapas, tres para cada pierna, en base a la figura 18, una vez estudiado las tres primeras etapas, obtendremos las restantes de manera automática, ya que se tratará de cambiar las dos piernas entre sí, y realizar unos pequeños ajustes.

Apoyándonos en la cinemática inversa resolvemos las siguientes etapas. Señalar que debido a que los cálculos podrían convertirse complejos debido a que trabajamos con senos y cosenos los cuales dan pie a un gran número de decimales difíciles de controlar, fijaremos nosotros mismos determinados valores de determinados ángulos, ayudándonos de la resolución gráfica, la cual resolveremos mediante el software de diseño gráfico AutoCAD, a su vez también fijaremos determinadas condiciones para simplificar los cálculos como que la cadera (“servo A”) no desplaza movimiento en el eje “y”, solo se desplazará en el eje “z”. Esta simplificación también ayuda a la hora de fijar el servo lineal en la cadera, ya que de no existir esta simplificación, la cadera realizaría un movimiento de sinusoide, difícil de programar.

La nomenclatura con la que se va a trabajar será (X) para la pierna izquierda (PI), (X') para la pierna derecha (PD).

- **Etapa 0:** Posición inicial, posición de reposos en la que encontramos el robot.

- **Etapa 1:** Posición en la cual la pierna derecha se flexiona, mientras que la pierna izquierda permanece en reposo. A continuación, en la figura 19 puede verse el sistema estudiado mediante resolución gráfica ayudado de unas ecuaciones que completan la resolución analíticamente.

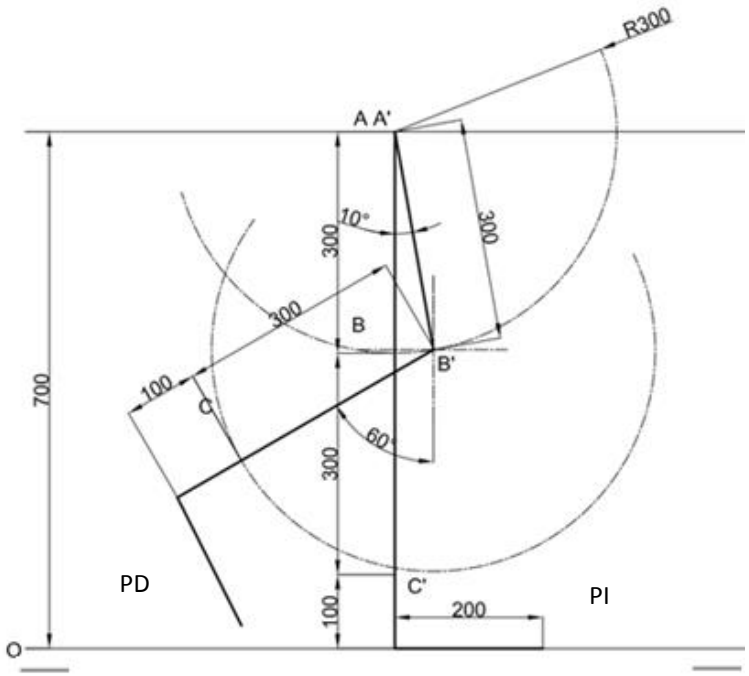


Figura 19. Estudio gráfico de la etapa 1, en AutoCAD.

En este estado solo modificarán su posición los Servos A' y B' correspondientes a la pierna derecha.

$$300\cos\theta A' + 300\cos\theta B' + d(OC') = 700$$

Para= θA´= 10° y d(OC´)=250 → θB´= 60°

A continuación, se recoge en la Tabla 1, todos los valores -en radianes (SI) -que debe tomar cada servo de cada pierna en cada posición para alcanzar la posición deseada. Si se desea ver el cálculo de las demás etapas, este cálculo se recoge en el Anexo F.

	Estado 1	Estado 2	Estado 3	Estado 4	Estado 5	Estado 6
PI_A	0	-0,087	0,43	-0,174	-0,698	-0,436
PI_B	0	0,261	0,61	1,04	0,261	-0,17
PI_C	0	0,057	0,489	0	0	-0,463
PD_A	-0,174	-0,698	-0,436	0	-0,087	0,43
PD_B	1,04	0,261º	-0,17	0	0,261	0,61
PD_C	0	0	-0,463	0	0,057	0,489

Tabla 1. Ángulos obtenidos para cada servo de cada pierna en cada etapa estudiada.

4.2.1 Implementación del código

El código del controlador se escribirá en lenguaje C, y dentro de la función principal ('main') se utilizará una estructura iterativa "do-while", que asegura, que al menos el bloque de instrucciones que contiene se ejecutará una vez, antes de la comprobación de condición de bucle.

A continuación, se muestra el código comentado por partes.

```
1  /*
2   * File:          my_controller.c
3   * Date:
4   * Description:
5   * Author:        FPellicena
6   * Modifications:
7   */
8
9  /*
10  * You may need to add include files like <webots/distance_sensor.h> or
11  * <webots/differential_wheels.h>, etc.
12  */
13 #include <webots/robot.h>
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <math.h>
17 #include <string.h>
18 #include <webots/servo.h>
19
20
21 /*
22  * You may want to add defines macro here.
23  */
24 #define TIME_STEP 64
25 #define SPEED 0.05
26 #define NUM_STATES 6
27
```

Figura 23. Primera parte del código

En esta parte declaramos las librerías necesarias para poder utilizar las funciones de webots, como son, las relacionadas con el nodo robot <webots/robot.h> y con el nodo servo <webots/servo.h>, además de otras librerías auxiliares.

También definimos campos que serán útiles en el posterior código como TIME_STEP, tiempo de muestreo en ms, SPEED, la velocidad a la que se moverán los servos (hemos fijado 0,05 rad/s para todos los servos en todos los estados, será constante, unidades en el sistema internacional en este software) y NUM_STATES que hace referencia al número de estados, como bien hemos señalado antes este valor es seis.

```
/*
 * You should put some helper functions here
 */
int max_pulse(int p1, int p2, int p3, int p4, int p5, int p6)
{
  int parray[] = {p1, p2, p3, p4, p5, p6};

  int i = 0;
  int mp = parray[i];
  for(i = 1; i < 6; i++)
  {
    if(mp < parray[i]){
      mp = parray[i];
    }
  }

  return mp;
}
```

Figura 24. Segunda parte del código

En Figura anterior, se muestra el código de una función que devuelve el mayor de los números que se le pasarán como parámetro. Esta función se utilizará de forma auxiliar en una para del código posterior.

```
/*
 * This is the main program.
 * The arguments of the main function can be specified by the
 * "controllerArgs" field of the Robot node
 */
int main(int argc, char **argv)
{
  /* necessary to initialize webots stuff */
  wb_robot_init();

  /*
   * You should declare here WbDeviceTag variables for storing
   * robot devices like this:
   *  WbDeviceTag my_sensor = wb_robot_get_device("my_sensor");
   *  WbDeviceTag my_actuator = wb_robot_get_device("my_actuator");
   */

  int i = 0, t = 0, state = 0;
  double pulse_ratio = 500.0/1.571;

  double pi_servo_a_current_position = 0.0;
  double pi_servo_b_current_position = 0.0;
  double pi_servo_c_current_position = 0.0;
  double pd_servo_a_current_position = 0.0;
  double pd_servo_b_current_position = 0.0;
  double pd_servo_c_current_position = 0.0;

  double pi_servo_a_positions[NUM_STATES] = {0, -0.087, 0.43, -0.174, -0.698, -0.436};
  double pi_servo_b_positions[NUM_STATES] = {0, 0.261, 0.61, 1.04, 0.261, -0.17};
  double pi_servo_c_positions[NUM_STATES] = {0, 0.057, 0.489, 0, 0, -0.463};
  double pd_servo_a_positions[NUM_STATES] = {-0.174, -0.698, -0.436, 0, -0.087, 0.43};
  double pd_servo_b_positions[NUM_STATES] = {1.04, 0.261, -0.17, 0, 0.261, 0.61};
  double pd_servo_c_positions[NUM_STATES] = {0, 0, -0.463, 0.057, -0.057, 0.489};

  int ppia;
  int ppib;
  int ppic;
  int ppda;
  int ppdb;
  int ppdc;
```

Figura 25. Tercera parte del código

En la Figura 25, se puede ver el comienzo del “main”, como su nombre indica la parte principal del código. Se definen una serie de variables, algunas de ellas auxiliares y otras más importantes como, “pulse_ratio”, de tipo “double”, ratio que hemos definido para ayudarnos a calcular el número de pulsos que un servo cualquiera tarda en realizar su movimiento de rotación.

Existe un problema a la hora de asociar un tiempo a la reproducción cinemática de webots, independiente de la velocidad asignada al servo, este problema se debe a que Webots trabaja sus reproducciones mediante "pulsos". Para que una reproducción sea continua y completa, hace falta que el número de pulsos sea suficiente para que la reproducción no quede abortada o contrarrestada por la siguiente. Este valor se ha encontrado, realizando experimentos en el mismo Webots. Para un valor máximo que puede realizar una articulación en nuestro caso, que es 90° (1.571 rad), el mínimo número de pulsos que permite una reproducción adecuada es 500. De esta manera damos valor a "pulse ratio" (500.0/1.571), y realizando una regla de tres calcularemos el número de pulsos necesarios para todos los servos de la etapa, quedándonos con el mayor valor para que a todos les dé tiempo a terminar su movimiento.

El siguiente grupo de variables (pi_servo_a_current_position, …), contendrá la posición actual de cada servo respectivamente, inicialmente todas se encuentran en la posición 0.0.

El conjunto posterior, está formado por una serie de vectores, que contienen las posiciones objetivo a alcanzar en cada uno de los distintos estados definidos, para cada uno de los servos. Estos valores son los que se calcularon en el apartado anterior, y quedan recogidos en la Tabla1 que se encuentra en el apartado anterior (apartado 4.2).

Por último las variables, una por servo, que contendrán el número de pulsos necesario para realizar las rotaciones de forma correcta en el estado correspondiente.

```
87
88    int npulse;
89
90    WbDeviceTag pi_servo_a = wb_robot_get_device ("PI_SERVO_A");
91    WbDeviceTag pi_servo_b = wb_robot_get_device ("PI_SERVO_B");
92    WbDeviceTag pi_servo_c = wb_robot_get_device ("PI_SERVO_C");
93    WbDeviceTag pd_servo_a = wb_robot_get_device ("PD_SERVO_A");
94    WbDeviceTag pd_servo_b = wb_robot_get_device ("PD_SERVO_B");
95    WbDeviceTag pd_servo_c = wb_robot_get_device ("PD_SERVO_C");
96
```

Figura 26. Cuarta parte del código

Definición de variables, necesaria para poder controlar los "Servos", el parámetro de entrada para la llamada es el campo "name" del servo,

```
/* main loop */
do {

  wb_servo_set_velocity(pi_servo_a, SPEED);
  wb_servo_set_velocity(pi_servo_b, SPEED);
  wb_servo_set_velocity(pi_servo_c, SPEED);
  wb_servo_set_velocity(pd_servo_a, SPEED);
  wb_servo_set_velocity(pd_servo_b, SPEED);
  wb_servo_set_velocity(pd_servo_c, SPEED);

  for(state = 0; state < NUM_STATES; state++)
  {
    ppia = (int)(fabs(pi_servo_a_positions[state] - pi_servo_a_current_position) * pulse_ratio);
    ppib = (int)(fabs(pi_servo_b_positions[state] - pi_servo_b_current_position) * pulse_ratio);
    ppic = (int)(fabs(pi_servo_c_positions[state] - pi_servo_c_current_position) * pulse_ratio);
    ppda = (int)(fabs(pd_servo_a_positions[state] - pd_servo_a_current_position) * pulse_ratio);
    ppdb = (int)(fabs(pd_servo_b_positions[state] - pd_servo_b_current_position) * pulse_ratio);
    ppdc = (int)(fabs(pd_servo_c_positions[state] - pd_servo_c_current_position) * pulse_ratio);

    npulse =max_pulse(ppia,ppib,ppic,ppda,ppdb,ppdc);

    //Estado state
    for (t=0; t<npulse; t++)
    {
      //Pierna izquierda
      wb_servo_set_position(pi_servo_a, pi_servo_a_positions[state]);
      wb_servo_set_position(pi_servo_b, pi_servo_b_positions[state]);
      wb_servo_set_position(pi_servo_c, pi_servo_c_positions[state]);
      //Pierna derecha
      wb_servo_set_position(pd_servo_a, pd_servo_a_positions[state]);
      wb_servo_set_position(pd_servo_b, pd_servo_b_positions[state]);
      wb_servo_set_position(pd_servo_c, pd_servo_c_positions[state]);

      wb_robot_step(TIME_STEP);
    }
```

Figura 27. Quinta parte del código

Comienzo del “do”, primero se fijan las velocidades que serán para todos servos iguales en todas etapas, en vez de un valor numérico se hace referencia a “SPEED”, definida al principio del código con un valor determinado. En estudios futuros, se podrían variar las velocidades de los servos en las diferentes etapas en busca de mejores resultados.

Tras esto, abrimos el primer “for”, que estudia cada etapa, primero calculamos el número de pulsos necesario para cada uno de los servos en cada etapa, para ello se multiplica el valor absoluto de la diferencia entre el ángulo objetivo y el ángulo actual por el ratio definido antes. Luego se utiliza la función “max_pluse()” para el número de pulsos mayor, impidiendo así reproducciones defectuosas, y se le asigna a la variable o “npulse” que marca el límite de iteraciones del segundo “for”.

El segundo “for”, definirá las posiciones que ha de alcanzar cada servo en cada etapa en relación a lo definido antes. Y ejecutará la función “wb_robot_step()” para sincronizar el controlador con los sensores de Webots, básicamente se establece el tiempo de muestreo con el valor de “TIME_STEP”, 64 ms.

```
133         pi_servo_a_current_position = pi_servo_a_positions[state];
134         pi_servo_b_current_position = pi_servo_b_positions[state];
135         pi_servo_c_current_position = pi_servo_c_positions[state];
136         pd_servo_a_current_position = pd_servo_a_positions[state];
137         pd_servo_b_current_position = pd_servo_b_positions[state];
138         pd_servo_c_current_position = pd_servo_c_positions[state];
139
140       }
141       i++;
142     } while (i<=10);
143
144     /* Enter here exit cleanup code */
145
146     /* Necessary to cleanup webots stuff */
147     wb_robot_cleanup();
148
149     return 0;
150   }
151
```

Figura 28. Sexta parte del código

Finalmente antes de cerrar el segundo" for", se actualiza el valor de la posición actual de los servos. Y así se repetirá el ciclo de seis etapas (definido en el primer "for") un total de 11 veces.

En el Anexo E se encuentra todo el código implementado.

4.3 Simulación de tareas en Webots

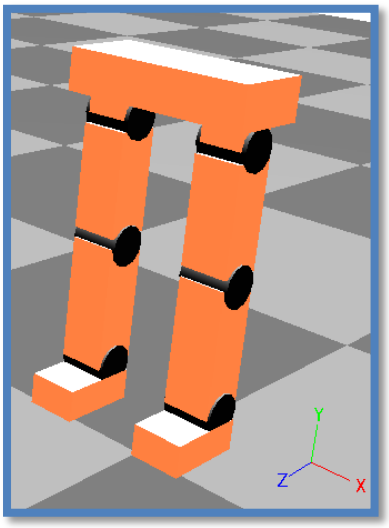


Figura 29. Posición de resposo del robot XHO.

Partiendo de la posición de reposo, activamos la simulación controlada por el código antes escrito y podemos visualizar, a continuación, caputardas durante la simluación, seis intantaneas que definen el proceso, estas instantaneas refelejan las seis estapas que se estudiaron en el modelo cinemático.

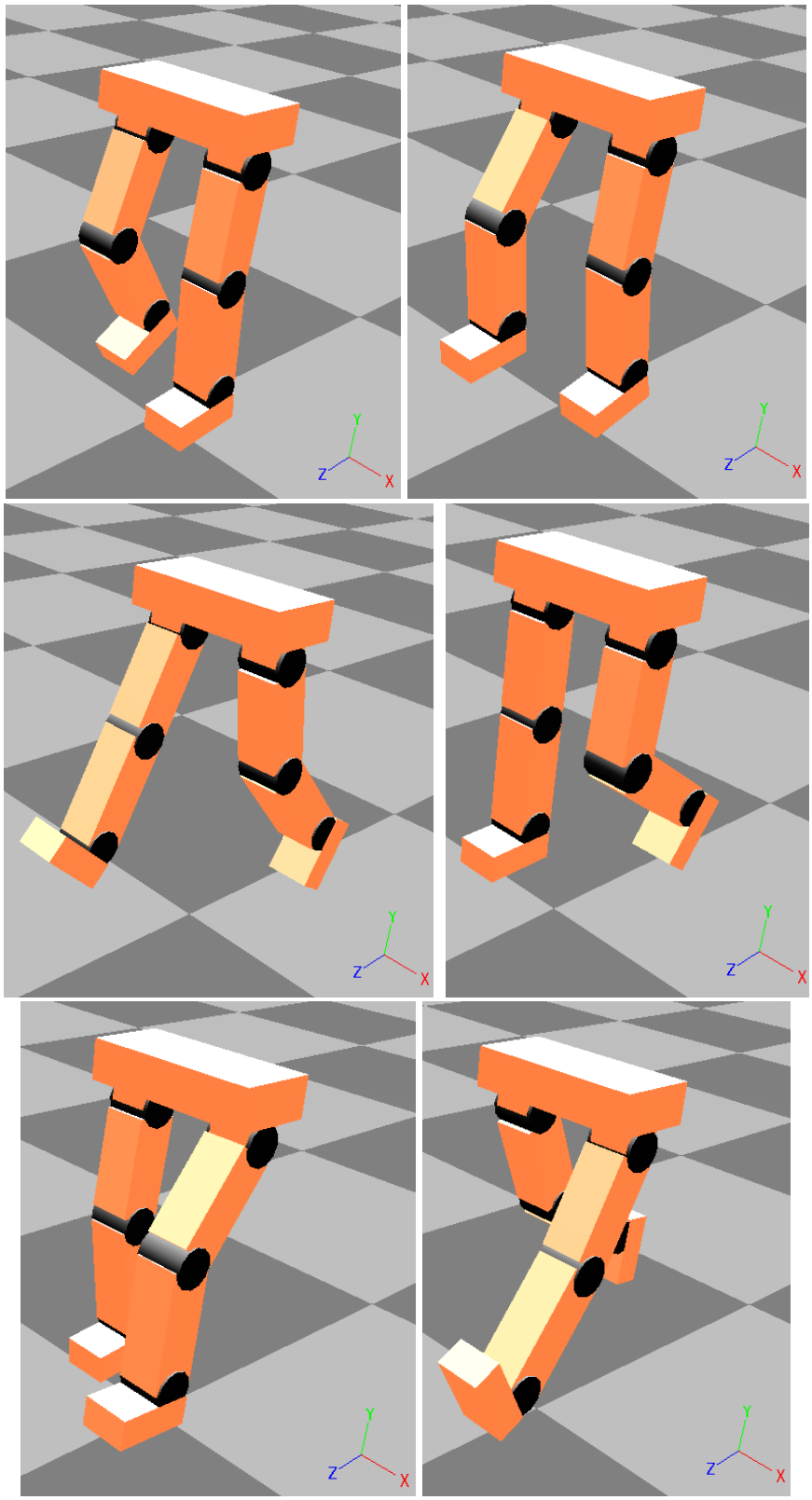


Figura 30. Las seis etapas de la actividad "caminata" capturadas de la simulación del software Webots.

5. Trabajos Futuros

Este trabajo deja abierto un amplio rango de posibles futuros trabajos, desde la implementación eléctrica del robot, hasta el estudio de materiales para su construcción real, pero el trabajo futuro inmediato que se entiende ya que es la continuación propia de este TFG, será el estudio dinámico del robot. Para el estudio de este campo, en los anexos plasmamos información referente a este tema, desarrollada para nuestro robot. (Más información Anexo B).

Otros posibles trabajos, que se pueden desarrollar a partir de este trabajo, son por ejemplo, el desarrollo del robot en Webots o en Autodek Inventor. En Webots nos podemos adentrar más en la arquitectura del robot, y desarrollar un sistema más complejo que permita a nuestro robot, elaborar actividades de mayor complejidad, incluyendo sensores por ejemplo. Mientras que en Autodesk Inventor se puede elaborar un estudio mecánico del ensamblaje de las diferentes partes que conforman el robot.

En el contexto de Webots hemos dejado iniciado un posible trabajo futuro, ya que en este proyecto hemos llegado a elaborar un robot nuevo, el cual tiene más grados de libertad que permitirán solventar el problema del equilibro de cara al estudio dinámico, o poder realizar estudios cinemáticos de trayectorias con curvas para la caminata de la máquina. Nuestro robot solo podía describir trayectorias de caminado recto, por lo que para pasar a una siguiente fase en la cual el robot pueda interactuar con el medio mediante sensores y esquivar obstáculos, es necesario la realización de un sistema más complejo. En los Anexo G podemos encontrar esta versión modificada.

Un trabajo que da pie a una gran variedad de posibilidades trabajos por estudiar, ya que la robótica es un campo que alberga diversos conocimientos de la ingeniería, desde la mecánica hasta la informática pasando por la electrónica y a su vez, un campo el cual crece día a día.

6. Conclusiones

En un principio, este proyecto se abordó con una perspectiva diferente (dinámica), creyendo que el problema que se enfrentaba era más asequible. Sin embargo, el problema del caminado de un robot bípedo es un problema complejo y ha sido abordado cinemáticamente en esta primera etapa.

En este trabajo se ha conseguido hacer frente a un entorno virtual desconocido hasta el momento, como era el entorno Webots, este software robótico que fue escogido de entre una selección de programas similares que se encuentran en el mercado, nos ha permitido modelar mundos virtuales en lenguaje VRML. “Mundos” que han sido controlados gracias al uso de la informática de programación, por lo que para la realización de este TFG también ha sido necesario el uso del lenguaje de programación “C”.

Además de la plataforma robótica Webots, otras han sido las plataformas que se han sido

en el transcurso de este TFG. AutoDesk Inventor y AutoCAD han sido utilizadas para obtener planos y esquemas del robot diseñado. Asentando de este manera nuestro control sobre estos softwares de diseño.

Finalmente, a parte de este control que hemos logrado conseguir sobre diferentes plataformas informáticas, para el desarrollo del trabajo ha sido necesario llegar a adquirir los conocimientos básicos de los entornos de la robótica bípeda y de la biomecánica, así como el conocimiento de los modelos cinemático y dinámico para la resolución del problema de la caminata en nuestro robot bípedo.

Con el TFG finalizado las sensaciones son bastantes positivas. Los conocimientos adquiridos en un tema desconocido para un ingeniero mecánico han sido muchos y me han permitido enfrentarme a condiciones reales de trabajo de un ingeniero: resolver problemas a partir de la investigación y la búsqueda autónoma de soluciones.

7. Anexos

7.1 Anexo A. Historia de la robótica

La palabra "robot", es de origen checo y significa siervo o esclavo; fue inventada por el escritor checo Karel Capek (1890-1938) en su obra teatral R.U.R., estrenada en Europa en 1920.

El ser humano lleva siglos soñando con la creación de máquinas autómatas y obedientes, capaces de llevar a cabo los trabajos duros y repetitivos que no requieran capacidad de improvisación.

El inicio de la robótica actual puede fijarse en la industrial textil del siglo XVIII, cuando Joseph Jacquard inventó en 1801 una maquina textil programable mediante tarjetas perforadas. Luego, la Revolución Industrial impulsó el desarrollo de estos agentes mecánicos.

En 1805, Henri Maillardert construyo una muñeca mecánica que era capaz de hacer dibujos, una serie de levas se utilizaban como “programa” para el dispositivo durante el proceso de escribir y dibujar.

La Revolución Industrial y la creación del sistema de cadenas de montaje, que divide la fabricación de cualquier elemento en pequeñas tareas, fue el primer gran paso hacia el logro de la robotización total en las industrias.

El concepto popular de un robot es el de aquel elemento de apariencia humana que actúa como tal. Este concepto de humanoide ha sido inspirado y estimulado por varias narraciones de ciencia ficción.

Actualmente, están desarrollando robots altamente inteligentes con más y mejores extensiones sensoriales, para entender sus acciones y captar el mundo que los rodea. Esto permite una toma inteligente de decisiones y el control del proceso en tiempo real.

El objetivo principal de algunos investigadores en robótica es construir robots que se parezcan a las personas, tanto en su cuerpo como en su comportamiento. Sin embargo, hasta ahora, los robots más utilizados en investigaciones robóticas han sido los robots manipuladores, los móviles y los robots con más de 2 patas.

La característica básica de robot humanoide está compuesto por dos piernas que le proporcionaran el movimiento, por dos brazos que le ayudaran a equilibrarse o a sujetarse y por una cabeza que aunque no sea imprescindible, le aportará un aspecto más humano.

La principal ventaja de los robots humanoide sobre otro tipo de robots es que pueden trabajar directamente en el mismo entorno que los humanos, otro aspecto importante es que el medio de locomoción de los robots humanoides se puede adaptar a su entorno, ya que la mayoría de

utensilios, maquinarias y escenarios (oficina, vivienda, metro, cine, calle, etc.) están adaptados para el uso humano, también estarán para el uso de robots humanoides.

Respecto al estado actual de las investigaciones en robots humanoides he de destacar que:

- La locomoción bípeda y estable no está resuelta totalmente.
- No existe robots humanoides de tamaño humano en el mercado, ya que la complejidad del problema de una locomoción bípeda y estable crece exponencialmente con la altura y peso del humanoide. Por lo tanto, predominan los humanoides pequeños, ya que pueden llevar a cabo un movimiento de caminar complejo de forma más sencilla y, además, son más baratos y más fáciles de controlar.

A continuación se presenta una lista de los mejores robots humanoides desarrollados hasta la actualidad.

- **Nao Robot**

Nao, un robot humanoide increíble desarrollado por la empresa francesa Aldebaran robotics, que en su última versión de marzo 2008, llamado ***Nao Robocup Edition***, muestra muchísimas cualidades motrices y una gran interactividad con el entorno. Muestra los grandes avances en el desarrollo de robots humanoides, donde no solo puede hacer tareas repetitivas sino que puede realizar muchísimas tareas dependiendo el entorno y que se puede adaptar a nuestras necesidades ya que es programable.

Entre sus cualidades principales se destacan su reconocimiento de voz y de ordenes, puede detectar las diferentes formas de los objetos y rostros y seguirlo su moviendo, es sensible al tacto en muchas partes de su cuerpo, tiene conectividad Wi-Fi que inclusive le permite comunicarse con otros robots de su misma clase, entre muchos otro. Pero en vez de nombrar sus cualidades, que mejor que ver sus capacidades en acción, en los siguientes videos vemos muchas de sus grandes características robóticas.

- **NEW ASIMO**

ASIMO, es más famoso de los robots humanoides creado por Honda, que desde el principio mostró grandes cualidades para la translación bípeda. Fue el que dio el gran primer paso en lograr que un robot caminara y es uno de los mejores robots de la actualidad.

Este robot tiene 5 capacidades principales, y son las siguientes:

1. Reconocimiento de objetos en movimiento. Entre las funcionalidades más importantes se incluye la capacidad de seguir los movimientos de las personas con su cámara, para seguir a una persona, o saludar a una persona cuando él o ella se acerca.
2. Reconocimiento de las posturas y los gestos. Debido a esto puede reaccionar y además de poder ser dirigido por comandos de voz, también sigue los movimientos naturales de los seres humanos. Esto permite, por ejemplo, reconocer cuando se ofrece un saludo de manos.
3. Reconocimiento de Medio Ambiente. Esto sirve, por ejemplo, para detectar los peligros potenciales, tales como escaleras, y evitar golpear a los seres humanos u otros objetos en movimiento.
4. Distinguir los sonidos. Puede responder muchas preguntas, ya sea por un breve movimiento del cuerpo en general o de solo la cabeza, o una respuesta verbal.
5. El reconocimiento del rostro. Se puede reconocer de forma individual aproximadamente 10 caras diferentes. Una vez que estén guardados en su memoria puede responderles por su nombre.

- **HRP-4C**

HRP-4C es una de los últimos robots considerado como ***ginoide***, es decir, un robot humanoide con apariencia femenina, fue presentado al público el 16 de marzo de 2009 por el *AIST* de Tokio. Este robot no fue diseñado para el servicio del hombre así como lo hace ASIMO, sino que fue diseñado puramente para el entretenimiento, especialmente en el campo de la moda. Actualmente este instituto se encuentra en el desarrollo de una versión mejorada, HRP-5C que se habla que tiene un costo de desarrollo de unos 3 millones de dólares.

Su altura es de 1.58 metros y pesa 43 Kg. Incluyendo la batería. Cuenta con 42 motores en total, que sirven para que realice

todos sus movimientos de elegancia y coqueteo tratando de imitar a las modelos. En su rostro cuenta con 8 motores que le permiten realizar gestos (permitiéndole mostrar varias emociones) e imitar el movimiento de los labios cuando habla.

Posee una inteligencia artificial que le permite el reconocimiento del habla, es decir, puede percibir cuando una persona le está hablando y puede entender varias órdenes sencillas.

También cuenta con la capacidad de síntesis del habla, que en otras palabras se refiere a que su voz no es producida por grabaciones, sino que posee un software y hardware que convierte texto en sonido, lo que permite que sus diálogos sean fácilmente programable.

- **Partner robot**

Este robot humanoide bastante amigable y de ahí su nombre Partner Robot creado por Toyota tiene como principales áreas de aplicación la asistencia y el cuidado de ancianos. Tiene una altura de 120cm y pesa 35 kg. Y tiene la capacidad de utilizar sus manos para muchas tareas. Su gran avance es que puede correr a casi 7 Km/h y se puede mantener en pie si es empujado.

- **QRIO**

QRIO ("**Q**uest for cu**RIO**sity") un robot Humanoide creado por **Sony**, mide apenas 60 cm, dispone de una tecnología denominada "Intelligent Servo actuador" que es lo que le permite andar dinámicamente es decir puede variar las r.p.m. y el torque en las articulaciones, y emplea una técnica denominada "Zero Moment Point" para mantener la estabilidad.

Con respecto a las capacidades de QRIO (tales como velocidad, autonomía, entre otras) no se puede nombrar alguna en particular ya que SONY ha estado en continuo desarrollo y cambio de versiones, pero a continuación se muestra una tabla con muchas de sus especificaciones más generales.

7.2 Anexo B. Modelo dinámico

7.2.1 Teoría del equilibrio bípedo.

El proceso de caminar es complejo, no sólo es necesario dar pasos. En este proceso están involucrados tanto pies, caderas, espina, brazos, hombros, cabeza, etc. La coordinación perfecta de todos los anteriores elementos hace que la caminata humana sea eficiente.

Como nuestro objetivo fundamental es crear un robot bípedo, es necesario conocer los principios básicos sobre el proceso de caminata de los humanos, para crear un sistema mecánico y electrónico que sea capaz de emular en cierta forma la caminata de un humano.

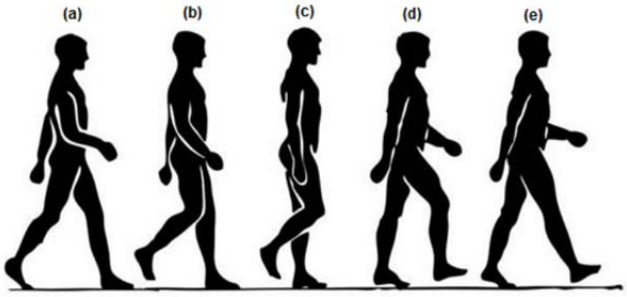


Figura 31. Caminata humana

La caminata comienza con los dos pies extendidos y sobre el suelo, en donde el equilibrio no es muy significativo (a). El gran problema comienza al levantar uno de los dos pies para realizar los movimientos siguientes (b, c y d), ya que la tendencia es caer hacia ese lado y hacia delante o atrás dependiendo del estado dinámico del robot en ese instante. Para evitar que el cuerpo caiga se deben realizar correctivos a los movimientos del robot (por ejemplo, mover la cadera al lado contrario del pie levantado), permitiendo así la estabilidad dinámica de la caminata. El medio ciclo se completa cuando los dos pies vuelven a estar en el suelo (e). El otro medio ciclo es idéntico, solamente que el pie que se levanta es el que anteriormente servía de apoyo y viceversa.

A continuación, se definen previamente algunos conceptos fundamentales relacionados con el análisis dinámico de una caminata humana.

- El equilibrio.
- Planos anatómicos y ejes de referencia.

7.2.2 El equilibrio:

Si quisiéramos dar un paso hacia delante con la pierna derecha, y con el objetivo de no perder el equilibrio, moveríamos el peso del cuerpo hacia la izquierda. De este modo la pierna izquierda se convierta en el apoyo del cuerpo y podamos levantar la derecha sin caernos. Si no hiciéramos esto simplemente nos caeríamos. Esto se debe a que el centro de gravedad del

cuerpo, cuya proyección sobre el suelo se encuentra en el punto medio entre los pies, se encontraba en el interior del polígono de sustentación hasta que levantamos el pie derecho. Luego de levantarlo, el polígono de sustentación queda limitado a la superficie de apoyo del pie izquierdo. Como la proyección del centro de gravedad no cae dentro de esta región, existe un momento angular neto de nuestro cuerpo, lo que implica que existe una velocidad angular no nula, y nos caemos.

Si por el contrario, movemos el centro de gravedad hacia la pierna izquierda antes de levantar el pie, la situación del centro del polígono de sustentación, y dentro de la superficie de apoyo del pie izquierdo. Cuando levantemos el pie derecho, la proyección del centro de gravedad estará dentro del nuevo polígono de sustentación y no perderemos el equilibrio.

7.2.3 Planos anatómicos y ejes de referencia:

Nos sirven para estudiar el cuerpo humano, son líneas imaginarias usadas en ciertas estructuras anatómicas reconocidas con el fin de dividir en planos al ser humano para localizar estructuras anatómicas.

Los planos fundamentales son: el sagital, es el plano que divide el cuerpo longitudinalmente; el frontal, cualquier plano vertical que sea perpendicular al sagital; y el transversal, es todo aquel plano que sea perpendicular al eje vertical.

Un eje es la línea alrededor de la cual se realiza el movimiento y plano es la superficie que se haya en ángulo recto con aquel y en la que se produce el movimiento. Estos términos se usan para facilitar la descripción del movimiento o dirección y por lo que se refiere a ejes y planos del movimiento articular, se describen con relación al cuerpo en posición anatómica.

La flexión y extensión, movimientos que se estudian en este proyecto, utilizan como plano de deslizamiento el sagital y el eje z, programa Webots (Figura 32). El plano sagital está trazado de forma vertical de atrás hacia delante de manera que divide al cuerpo en lado derecho y en lado izquierdo.

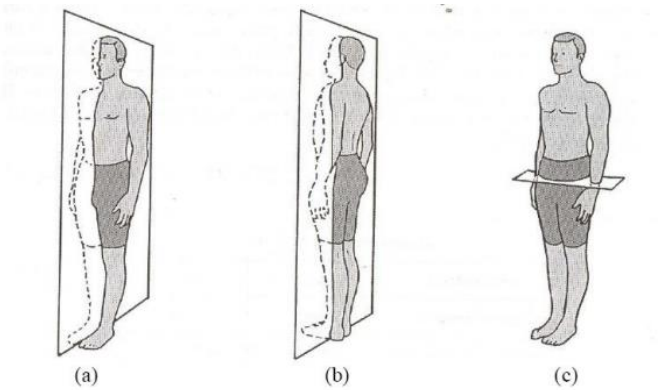


Figura 32. Planos de referencia: a) plano sagital, b) plano frontal, c) plano transversal

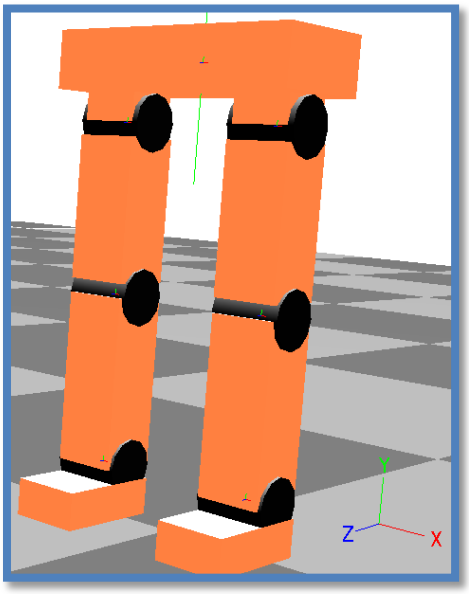


Figura 33. Ejes de referencia considerados en el estudio Webots.

7.2.4 Análisis dinámico

Elaborar el análisis dinámico de un sistema mecánico implica encontrar las relaciones matemáticas entre sus coordenadas "q" y fuerzas "Q" generalizadas. De acuerdo a "Spong" en la deducción de dichas relaciones se pueden distinguir dos objetivos principales:

- Obtener ecuaciones de forma cerrada que describan la evolución en el tiempo de las coordenadas generalizadas q(t).
- Conocer cuáles son las fuerzas generalizadas Q(t) que deben ser aplicadas para producir una evolución particular en el tiempo de las coordenadas generalizadas q(t) sin preocuparse de la relación funcional entre ambas.

Aunque se sabe que las formulaciones de "Lagrange-Euler" y "Newton-Euler" para deducir las ecuaciones dinámicas de un sistema conducen a los mismos resultados; se considera que el método basado en energía de "Lagrange-Euler" es superior para alcanzar el primer objetivo, mientras que el método recursivo de "Newton-Euler" resulta más apropiado para el segundo objetivo.

Otra de las consideraciones a tomar es el tiempo de cálculo de la dinámica, en este caso de la dinámica inversa para obtener las torques necesarias en las articulaciones y así el bípedo pueda caminar de manera más suave. (Fu et al 1994).

El modelo más difícilmente computable es el "Lagrange-Euler", el modelo más rápido es el de "Newton-Euler" esto se puede apreciar en la tabla 2 que muestra el número de operaciones, aunque siendo realistas los dos modelos son bastante tardados a la hora de computarlos.

Método	Lagrange-Euler	Newton-Euler
Multiplicaciones	$\frac{128\,n^4}{3}+\frac{512\,n^3}{3}+\frac{739\,n^2}{3}+\frac{160\,n}{3}$	132n
Sumas	$\frac{98\,n^4}{3}+\frac{781\,n^3}{6}+\frac{559\,n^2}{3}+\frac{245\,n}{6}$	111n-4
Representación cinemática	Matrices homogéneas 4x4	Matrices de rotación y vectores de posición
Ecuaciones de movimiento	Ecuaciones diferenciales de forma cerrada	Ecuaciones recursivas

Tabla 2. Comparación del número de operaciones realizadas por los métodos de "Lagrange-Euler" y "Newton-Euler".

La gran desventaja del modelo de "Newton-Euler" es que solo funciona para velocidades bajas donde los elementos de Coriolis y centrífugo, que no toman en cuenta, son despreciables. Y la gran desventaja del método de Lagrange-Euler es la cantidad de cálculos que se deben de hacer por lo que visto la gran dificultad para analizar y modelar el robot íntegramente, varios científicos han optado por simplificar el modelo dinámico del robot bípedo, quedando las siguientes simplificaciones, presentadas en la figura 34:

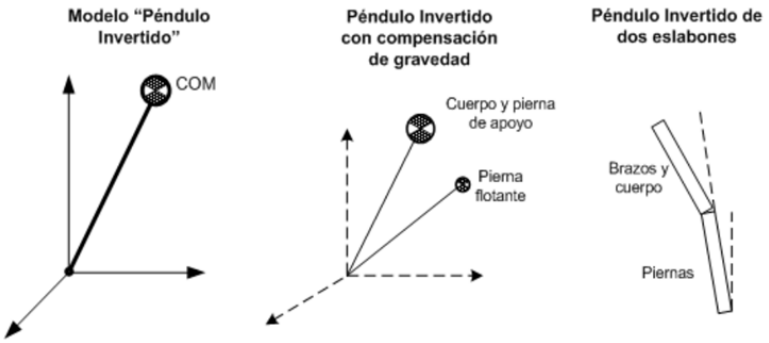


Figura 34. Modelos simplificados de un Robot Bípedo.

Obviamente el modelo más simple es el "péndulo invertido", ya que considera al robot como un punto de masa concentrado en el COM, pero presenta algunas desventajas, por ejemplo, no considera la acción de la pierna flotante, la misma que modifica de manera significativa la dinámica del robot. El segundo modelo considera la acción de la pierna flotante, en cambio, el tercer modelo separa el robot en dos partes; piernas y cuerpo suiperior.

El modelo que tentativamente elegiremos será el "Péndulo invertido" y considerando la desventaja mencionada anteriormente, se propone el cálculo dinámico del COM del robot durante la caminata, es decir, a partir de la posición de cada uno de los eslabones del robot durante un determinado instante se formará el modelo simplificado, tal como muestra la figura 35.

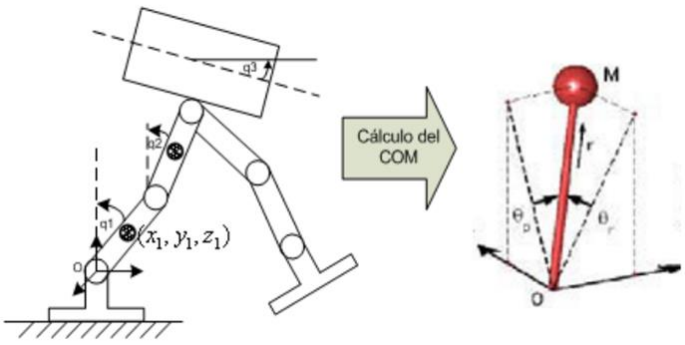


Figura 35. Cálculo dinámico del COM.

En donde,

$$x = \frac{\sum_i m_i x_i}{\sum_i m_i} \; ; \; y = \frac{\sum_i m_i y_i}{\sum_i m_i} \; ; \; z = \frac{\sum_i m_i z_i}{\sum_i m_i}$$

Ecuación 1,

Siendo x, y, z las coordenadas del COM del robot, durante su movimiento, y, [xi,yi,zi] las corrdenadas de los COM de cada uno de los eslabones del robot. El origen de las coordenadas está en la pierna de apoyo (punto O).

Para encontrar las coordenadas de cada eslabón, se utilizará cinemática directa en este caso, de modo que

xi= f(q1,q2,q3…qn)

yi= f(q1,q2,q3…qn)

zi= f(q1,q2,q3…qn) ecuación 2,

7.2.4.1 Cálculo del COP

El COP (Punto P) es definido como el punto en la superficie de contacto en donde la fuerza de reacción neta del piso actúa. Algunos autores conocen al COP como C-ATGRF (“Center of the actual ground reaction force” – Punto de reacción actual del suelo). En la Figura 36 se esquematiza las fuerzas que actúan sobre un pie de apoyo.

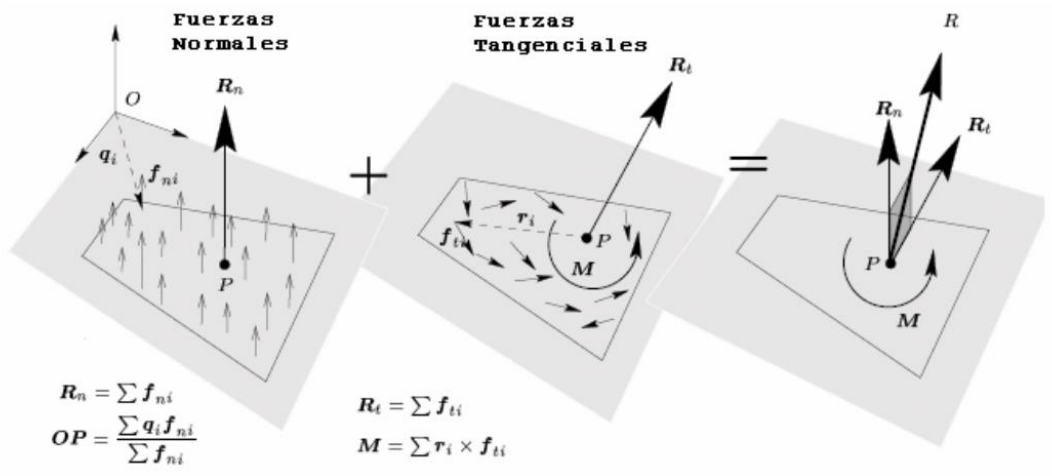


Figura 36. Fuerzas de reacción del suelo que actúan sobre un pie de apoyo.

Las fuerzas tangenciales de la fuerza de reacción constituyen la fricción, la misma que debe compensar a las fuerzas del robot en esa dirección, como se analizará más adelante, es decir, las fuerzas que son de nuestro interés para el cálculo del COP son las fuerzas normales. Llevando esto a la práctica, se dispondrá de cuatro sensores de fuerza en cada uno de los pies, los mismos que permitirán el cálculo del COP en todo momento. La disposición de los sensores se muestra en la figura 37.

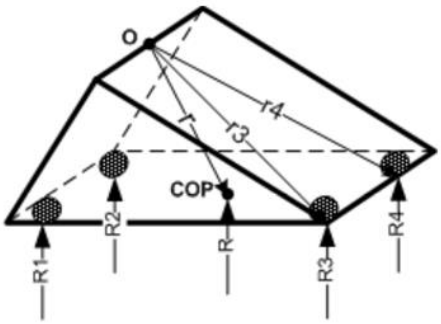


Figura 37. Disposición de sensores.

A continuación, se muestra un ejemplo de sensor de fuerza que se puede utilizar para este caso. Estamos hablando de un sensor de fuerza flexible.

Figura 30. Sensor de fuerza flexible

Analizando los momentos generados por las fuerzas:

$$r\ x\ R = \sum_{i=1}^{4} r\ x\ R \quad en\ donde, R = R_1 + R_2 + R_3 + R_4 \qquad \text{Ecuación 3}$$

Ahora analizando escalarmente en el polígono de soporte:

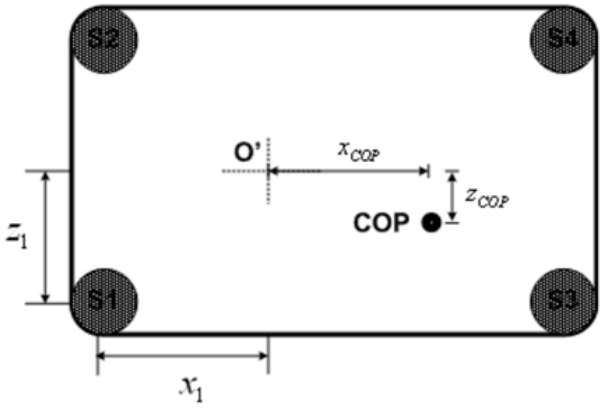


Figura 38. Vista superior del COP en el polígono de soporte.

En donde, X_{COP} Y Z_{COP} son las coordenadas del COP, O´es la proyección del punto O dentro del polígono de soporte y S_1, S_2, S_3 y S_4 son los sensores de fuerza. Las coordenadas del COP con respecto a O´serán:

$$M_{R_Z} = \sum_{i=1}^{4} M_{i_Z} \qquad M_{R_x} = \sum_{i=1}^{4} M_{i_x}$$

$$R \cdot X_{COP} = \sum_{i=1}^{4} R_i \cdot X_i \qquad R \cdot Z_{COP} = \sum_{i=1}^{4} R_i \cdot Z_i$$

$$X_{COP} = \frac{\sum_{i=1}^{4} R_i \cdot X_i}{\sum_{i=1}^{4} R_i} \qquad Z_{COP} = \frac{\sum_{i=1}^{4} R_i \cdot Z_i}{\sum_{i=1}^{4} R_i}$$

Conjunto de ecuaciones número 4,

Se puede notar claramente que las distancias xi y zi de los sensores permanecen constantes, entonces, lo que hace que varíe la posición del COP es la "repartición" de fuerzas en cada sensor, es decir, la forma en la que hace contacto el pie con el suelo, que está relacionado con la dinámica del robot en su caminata.

7.2.4.2 Método de control de equilibrio

La idea fundamental del control de equilibro (aunque parezca obvia) es evitar que el robot caiga, para esto, se dispone de algunos indicadores tales como el ZMP y el FRI. A continuación, se realiza un análisis de estos puntos.

Partamos del análisis de todas las fuerzas y momentos que actúan sobre el pie de apoyo:

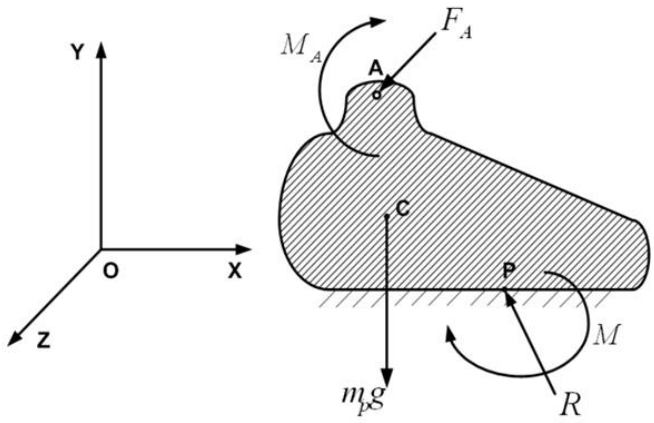


Figura 39. Fuerzas y momentos sobre el pie de apoyo.

En donde MA y FA son el momento y la fuerza resultante generado por el cuerpo en movimiento, M y R son el momento y fuerza de reacción del suelo, mp es la masa del pie. Para que el pie se encuentre en equilibrio estático:

$$\sum F = 0 \quad , \quad R + F_A = 0$$

$$\sum M_0 = 0$$

$$OP\ x\ R + OA\ x\ F_A + M + M_A + OC\ x\ (m_p\ g) = 0$$

Conjunto de ecuaciones número 5

A partir de ahora se asumirá que la fricción de la superficie es lo suficientemente grande, para no permitir que exista deslizamiento entre el pie y el piso, entonces, R_x R_z y M_y del suelo compensan las fuerzas y momentos del cuerpo en esas direcciones, por lo que;

$$\sum F_y = 0 \quad , \quad R_y + F_{A_y} = 0$$

$$(\sum M_0)^H = 0$$

$$(OP\ x\ R)^H + (OA\ x\ F_A)^H + (M)^H + (M_A)^H + (OC\ x\ (m_p\ g))^H = 0$$

Es decir, se ha podido reemplazar a la fuerza de reacción neta del piso por una que actúa solo en el eje y. También los momentos se han reducido al plano horizontal (ejes x,z)

Una condición fundamental para que el sistema esté en equilibrio es que en el punto P,

$M_x = M_z = 0$

Es decir, los momentos generados por la fuerza de reacción son cero, siendo posible reemplazar a la reacción del suelo por una única fuerza en el eje y.

Cambiando el punto de referencia de O hacia A y despreciando la masa del pie:

$$(\sum M_A)^H = 0$$

$$(AP\ x\ R)^H + (M_A)^H = 0$$

$$(AP\ x\ R)^H = -(M_A)^H$$

Conjunto de ecuaciones número 6.

Claramente se puede notar que la igualdad anterior se cumple eligiendo el punto P adecuado, de tal forma que el momento generado por la fuerza de reacción del suelo compense al momento total generado por el cuerpo en su caminata; este punto es conocido como ZMP – Es discutible el nombre de ZMP ya que únicamente dos componentes del momento de reacción del piso son iguales a cero (en los ejes x, z). El ZMP existe únicamente dentro del polígono de soporte. Cuando existe el ZMP el robot se encuentra dinámicamente estable, siendo este punto coincidente con el COP

En el caso de que el momento generado por el cuerpo (MA) sea demasiado grande, puede darse que el punto P se encuentre fuera del polígono de soporte (Figura 40), en este caso la fuerza de reacción del piso se encuentra en el borde del pie, el ZMP no existe y evidentemente el robot tiende a perder su estabilidad. Un indicador físico de la pérdida de estabilidad del robot es la rotación del robot a partir de los bordes del polígono de soporte.

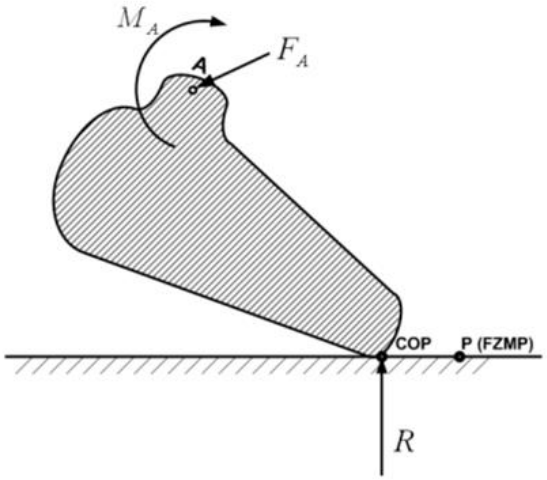


Figura 40. Rotación del pie a partir de uno de sus bordes. Robot inestable

Dado que el ZMO existe únicamente dentro del polígono de soporte, su cálculo nos da una idea de cómo de estable se encuentra el robot en un determinado instante, sin embargo existe un concepto más general y es el FRI

"El FRI es el punto en la superficie, donde la fuerza de reacción neta del suelo tendría que actuar para mantener el pue estacionario"

El FRI puede existir ya sea dentro o fuera del polígono de soporte, o sea, que puede ser también un indicador de como de instable es el robot. Cuando el FRI está dentro del polígono de sporte, el robot esta dinámicamente estable y ese punto es coincidente con el ZMP y el COP.

Entonces, de aquí nace la idea fundamental del control de equilibrio de la caminata bípeda:

"Para que un robot bípedo se encuentre dinámicamente estable durante su caminata, el FRI debe estar siempre dentro del polígono de soporte"

Despes de revisar estos conceptos, la pregunta podría ser: ¿Cuál es la diferencia entre el ZMP y el COP?. Según "Goswami" no existe diferencia entre el ZMP y el COP, sin embargo, "Vukobratovic" realiza las siguientes observaciones:

1. El COP siempre existe (ya sea en el borde del polígono del soporte), sin embargo el ZMO no existe cuando el robot está dinámicamente insestable.
2. El ZMP peude existir en otras partes del cuerpo del robot (por ej. en el hombro)

Dada esta diferente manera de interpretar al ZMP, nosotros hemos decidido adoptar el concepto de FRI en nuestros análisis, ya que creemos que es un término más adecuado y general para analizar la estabilidad dinámica de un robot bípedo.

Se han realizado varios análisis de este punto (FRI) y han surgido principalmente dos tendecnias para el cálculo.

Una de las formas de calcular el FRI es a partir de la dinámica de cada uno de los eslabones que conforman el robot, de modo que:

$$X_{FRI} = \frac{\sum_i m_i z_i (a_{y_i} + g) - \sum_i m_i y_i a_{z_i - \sum_i H_{G_{x_i}}}}{\sum_i m_i (a_{y_i} + g)}$$

$$Z_{FRI} = \frac{\sum_i m_i x (a_{y_i} + g) - \sum_i m_i y_i a_{x_i - \sum_i H_{G_{z_i}}}}{\sum_i m_i (a_{y_i} + g)}$$

Conjunto de ecuaciones número 7.

En donde, se llega a calcular con precisión las coordenadas del FRI, sin embargo, se requiere las coordenadas, aceleraciones, momentos angulares, etc. de cada eslabón del robot.

Una manera más simple de calcular el FRI es a partir de un modelo simplificado del robor – en nuestro caso, un péndulo invertido 3D, en donde, el cálculo se realiza a partir de las coordenadas, aceleraciones del COM y momento angular total del robot.

$$X_{FRI} = \frac{M(a_{y_i} + g)x - Ma_x y + H_z}{M(a_y + g)}$$

$$Z_{FRI} = \frac{M(a_{y_i} + g)z - Ma_z y + H_x}{M(a_y + g)}$$

Conjunto de ecuaciones número 8.

Nosotros sugerimos utilizar las formulas anteriores, dado que se supone que se utilizará el modelo de pendulo invertido 3D para la implementación del sistema de control, por tratarse del modelo más elemental y sencillo de aplicar tratandose de un primer estudio.

Según estos conceptos, existen dos tipos de caminata:

Caminata estática. Considera que las fuerzas de gravedad son lo suficientemente grandes, por lo que se podría despreciar las fuerzas dinámicas producidas por el movimiento del robot. Eliminando las componetnes dinámicas de las ecuaciones del FRI (Ecuación 8), se tiene:

$X_{FRI} = X$

$Z_{FRI} = Z$

Entonces, en la caminata estática, la posición del FRI es la proyección del COM sobre el suelo, de aquí que si el FRI está fuera del polígono de soporte, el robot caerá, En la figura 41 se esquematiza la proyección del COM sobre el poígono de soporte, en el plano sagital.

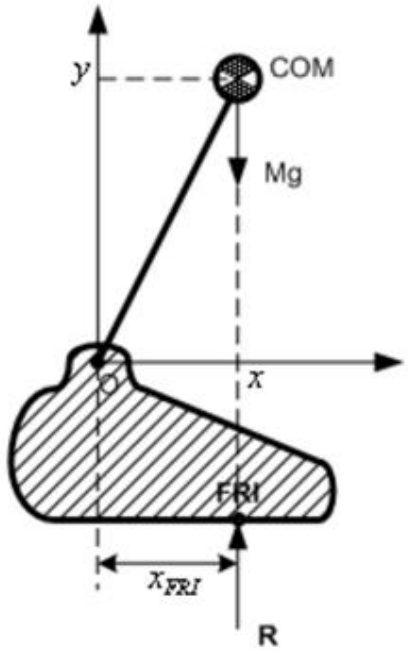


Figura 41. Proyección del COM sobre el polígono de soporte

Caminata dinámica. Ajusta la caminata de acuerdo a la situación en la cual se encuentra el robot. Toma en cuenta las aceleraciones lineales y angulares del robot, ya sea en el modelo completo (Ecuación 7) o en el modelo simplificado (Ecuación 8).

Por lo tanto la caminata dinámica no funciona como la estática, ya que la proyección la proyección vertical del centro de masa puede encontrarse fuera del área de soporte formada por las bases del robot bípedo en periodos de tiempo. Esto permite que el sistema presente cortos instantes de caída; sin embargo estos periodos de tiempo deben ser pequeños y bien controlados para que el sistema no se torne inestable.

Al comparar los dos métodos de balance, es claro que el método estático es altamente restrictivo y genera movimientos poco eficientes y lentos. Al contrario, aunque el método dinámico es mucho más efectivo y rápido, éste presenta un comportamiento inestable, lo cual obliga a tener algoritmos de control altamente eficientes y complejos.

7.2.5 Controladores en Robots Bípedos

La idea de un controlador en un robot bípedo, está vinculado al estudio dinámico ya que es hacer que el bípedo pueda caminar sin caerse, esto se logra haciendo que ciertos puntos del bípedo (como pueden ser la cadera o el pie) sigan trayectorias definidas tales que emulen la marcha humana, esto aunado a que el ZMP siempre se mantenga dentro del polígono de soporte, es por eso que los puntos importantes del controlador son los siguientes:

- **Hacer que el robot bípedo llegue a los ángulos calculados por la cinemática inversa de cada articulación para así seguir una trayectoria deseada de la cadera y el pie**
- Tomando en cuenta las trayectorias de la cadera y el pie, hacer que el ZMP no salga del polígono de soporte.

Estos puntos son la base para poder generar un diagrama de control que tenga por lo menos el control de las articulaciones y el control del ZMP como lo podemos ver en la figura 42 que tiene los elementos básicos para poder hacer que el robot bípedo camine sin caerse, claro está que dentro de los diagramas de control propuestos se puede hacer uso de métodos heurísticos con el fin de proporcionar robustez y que el bípedo pueda caminar por diferentes terrenos como se muestra en el diagrama en la figura 43 que hace uso de una lógica difusa.

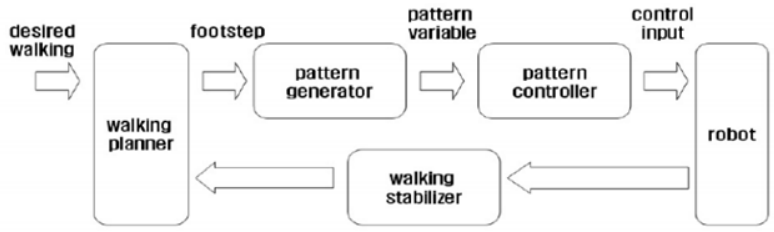


Figura 42. Caminata deseada (Jonghoon Park).

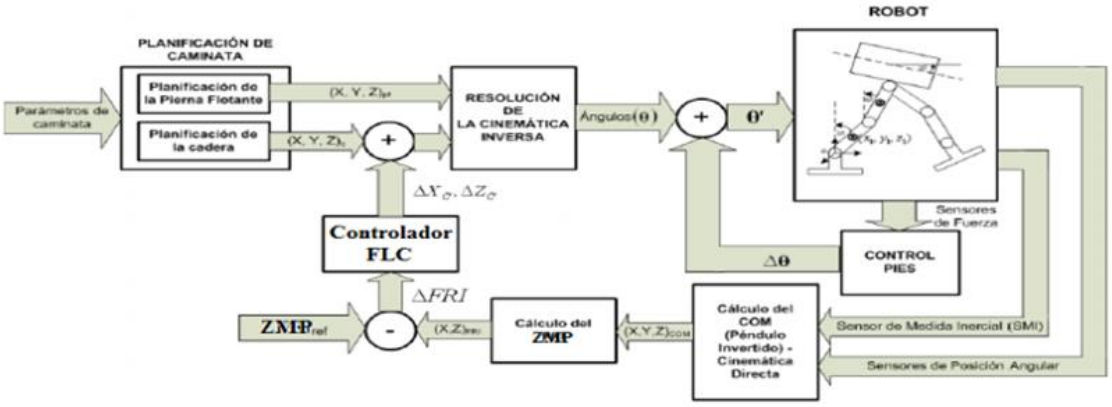


Figura 43. Diagrama del controlador ZMPL-FLC (Narong and Manukid).

7.3 Anexo C. Presentación del software Webots

7.3.1 Descripción general

Webots es un paquete de software profesional para modelar, programar y simular robots móviles. Con este software, el usuario puede diseñar configuraciones complejas de robots, en un entorno virtual en 3D. El usuario puede elegir las propiedades de cada objeto, como forma, color, textura, masa, fricción, etc.

Por otra parte, los robots pueden ser equipados con un amplio número de sensores y actuadores, tales como sensores de distancia, ruedas motrices, cámaras, servomotores, sensores de contacto y fuerza, emisores y receptores de señales de radiofrecuencia, etc. Finalmente, el usuario es capaz de programar cada robot individualmente para que exhiba el comportamiento deseado.

Además, los controladores de los robots se pueden programar en un entorno de desarrollo externo, o en el que hay disponible en Webots. El comportamiento del robot puede ser testeado en mundos con física realística. Y los programas de los controladores se pueden trasladar a robots físicos reales.

La primera vez que se inicia Webots, hay que establecer cuál será nuestro directorio de trabajo. El asistente creará en este directorio una serie de carpetas, las más importantes de las cuales son “worlds” y “controllers” (ver figura 44), y en ellas se almacenarán los mundos y los controladores creados por el usuario.

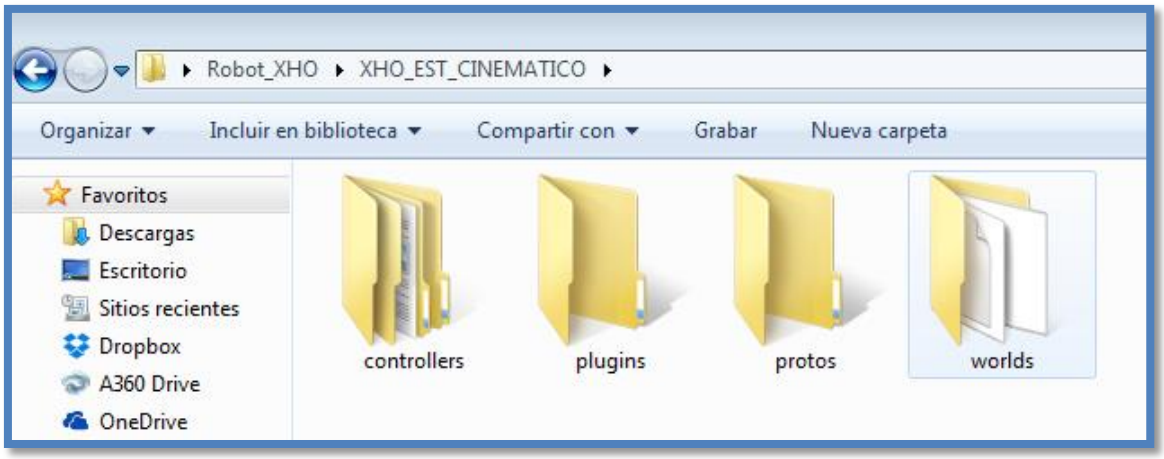


Figura 44. Imagen de nuestro directorio de trabajo

En la figura 45. Se muestra la estructura jerárquica principal de este simulador.

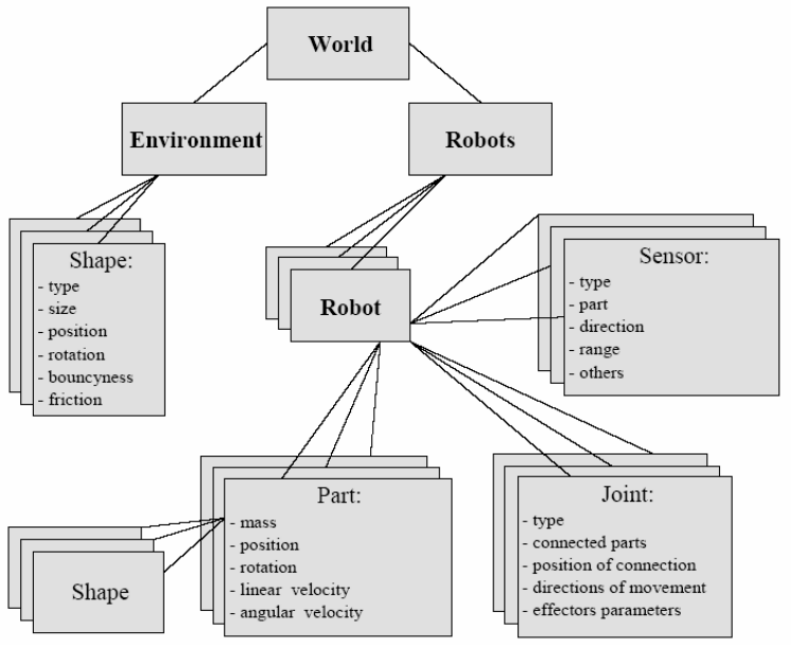


Figura 45. Jerarquía simulador Webots

7.3.2 Interfaz

Webots ha sido desarrollado por Cyberbotics Ltd. Tiene versiones tanto para Windows, como para Linux y Mac; para la realización de este TFG hemos utilizado la versión 6.4.4 de Webots que procesaremos bajo el sistema operativo Windows 7. En esta versión la interfaz gráfica se divide en 4 partes (ver figura 46), a continuación, vamos a tratar de explicar la función de cada una de ellas y cómo la hemos utilizado para este proyecto en concreto.

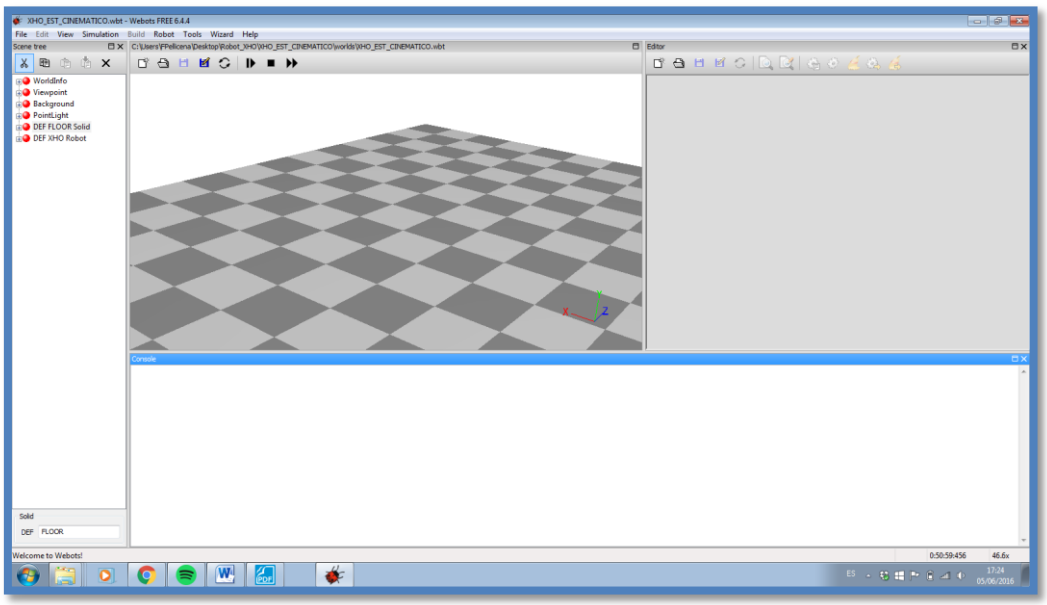


Figura 46. Interfaz de Webots.

7.3.2.1 Scene Tree

Para poder crear el mundo virtual sobre el que trabajaremos, tenemos una ventana donde se nos muestra el "Scene Tree" o "Árbol de Nodos". Podemos mostrarla u ocultarla haciendo click en Windows>Scene Tree en la ventana principal (o Ctrl + T). Se trata de una ventana con la jerarquización de nodos VRML que forman todo el mundo. En esta ventana modificaremos el mundo añadiendo y modificando los nodos que sea necesario.
En la parte superior de esta ventana, tenemos una serie de botones con los que manipular los elementos del árbol de escena (ver figura 47). De izquierda a derecha son: Cortar, copiar, pegar, pegar después del nodo actual, eliminar nodo, resetear a la última configuración guardada, transformar un nodo en otro, insertar un nuevo nodo después del actual, crear un nuevo nodo, exportar, importar y la ayuda.

Figura 47. Menú de la ventana Scene Tree.

El árbol de nuestra escena contiene los nodos típicos de todo mundo virtual creado con VRML: "WorldInfo" y "Viewpoint". Con ellos proporcionamos información de cómo es el mundo y establecemos un punto de vista predeterminado.

7.3.2.1.1 Nodo WorldInfo

Este nodo proporciona el ambiente de ejecución de los movimientos del robot, y está subdividido en los siguientes campos (ver figura 48):

- **Campo información del mundo ("*info*"):** en este campo se puede almacenar información referente al mundo (p.ej. para qué se diseñó, quién lo creó y la fecha del trabajo). Al seleccionar este campo, en la parte inferior se encuentra un cuadro de texto, el cual permite cambiar la información al nodo seleccionado.
- **Campo gravedad ("*gravity*"):** en el campo gravedad existen tres valores X, Y y Z respectivamente. Para mantener la simulación lo más realista posible la gravedad de la tierra se mantiene con un valor de -9.81 en el eje de las Y, mientras que en el eje X y Z se mantiene en cero.
- **Campo de paso de tiempo ("*basicTimeStep*")**: este campo permite determinar la velocidad en que se ejecutará un movimiento; el valor predeterminado que se usará es de 32 milisegundos.

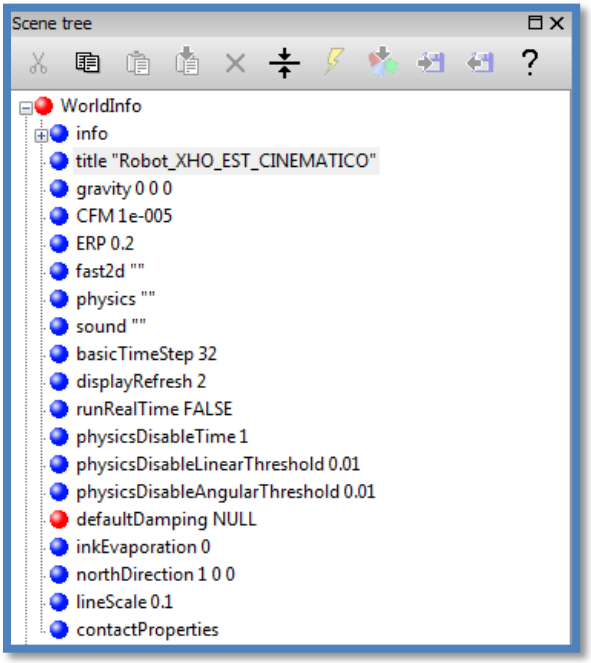


Figura 48. Nodo WorldInfo

7.3.2.1.2 Nodo Viewpoint

El nodo "***Viewpoint***" permite posicionar a la cámara para interactuar con el mundo de la realidad virtual en un punto determinado (ver figura 49). Para ubicar la cámara es necesario conocer los siguientes campos:

- **Campo de visión ("*fieldOfView*"):** es el zoom de la cámara, le permite alejar o acercar la cámara del mundo virtual.
- **Campo orientación ("*orientation*"):** este campo permite hacer una rotación a la cámara en los ejes de X, Y y Z a un ángulo Alfa.
- **Campo posición ("*position*"):** el campo en cuestión permite hacer una traslación al lugar que se le indique.

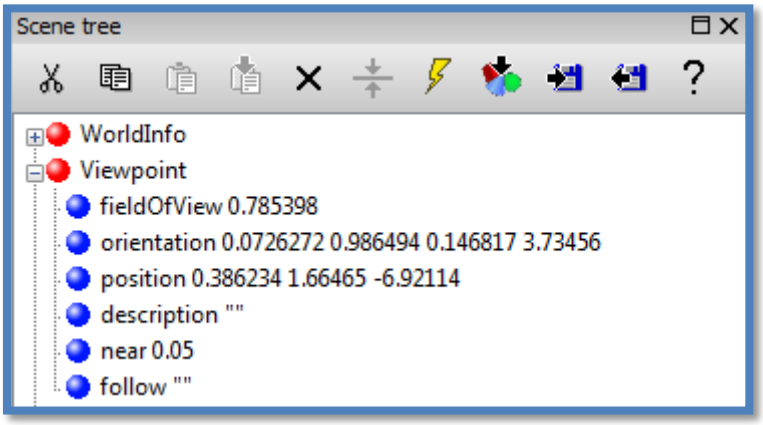


Figura 49. Nodo Viewpoint.

7.3.2.1.3 Nodo Background

Este nodo permite establecer un color de fondo gracias al campo que se llama color de cielo (*"skycolor"*). El color está predeterminado en azul por el sistema (0.4, 0.7, 1). (Ver figura 50).

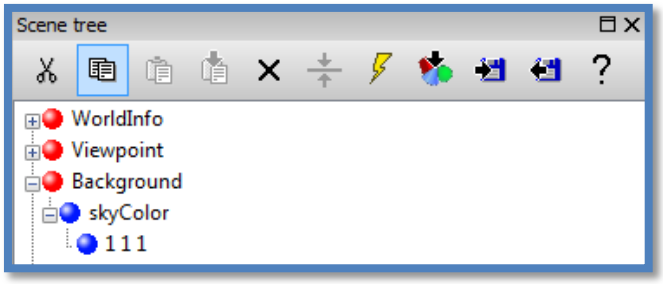


Figura 50. Nodo Background.

7.3.2.1.4 Nodo PointLight

El nodo ***"Pointlight"*** determina la luminosidad, el color, la intensidad, la ubicación de las luces y el radio (ver figura 51). Para darle valores al "foco" es necesario conocer los siguientes campos:

- **Campo intensidad del ambiente (*"ambientIntensity"*):** este campo permite controlar la intensidad del ambiente, obtener una luz más brillante o más oscura dependiendo de lo que se desee.
- **Campo atenuación (*"attenuation"*):** indica la posición en que la luz se dirige en el campo, se posiciona con las coordenadas (X, Y y Z).
- **Campo color (*"color"*):** permite indicar el color de la luz. Por defecto en este proyecto trabajamos con la luz blanca cuyos valores son 1, 1, 1 en formato RGB.
- **Campo intensidad (*"intensity"*):** permite aumentar la intensidad a la imagen, sin embargo, no es recomendable aumentar la intensidad porque los colores se empiezan a perder por la intensidad de la luz.
- **Campo localización (*"location"*):** ubica la posición del foco en el campo; con las coordenadas X, Y y Z se determina el origen de la luz.
- **Campo encendido (*"on"*):** este campo da la posibilidad de apagar o encender el foco. El programa se establece como True, encendido.
- **Campo de sombra (*"castShadows"*):** se obtiene sombra de los objetos que están en el campo, por defecto se establece como False.

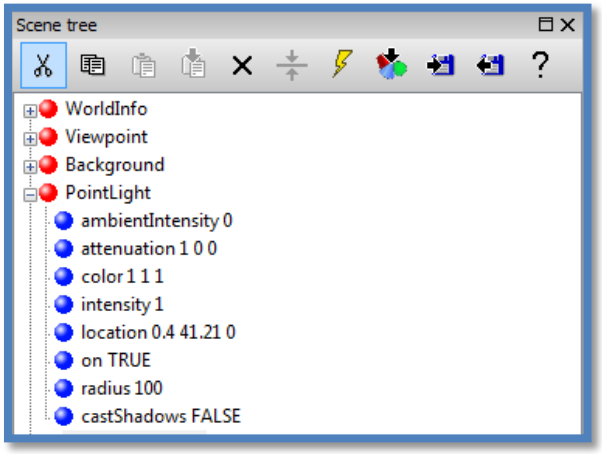


Figura 51. Nodo PointLight.

7.3.2.2 Text Editor

Una vez creados todos los objetos del mundo y modelado también nuestro robot, debemos programar las acciones que se llevarán a cabo. Lo que nos permite Webots es asociar un programa (controlador) a un robot, de tal forma que podemos recibir información de los sensores que hayamos colocado en nuestro robot y mandar información a los actuadores que también le hayamos incorporado. Estos controladores pueden estar escritos en C, C++ o Java. En este proyecto se ha decidido trabajar en C.

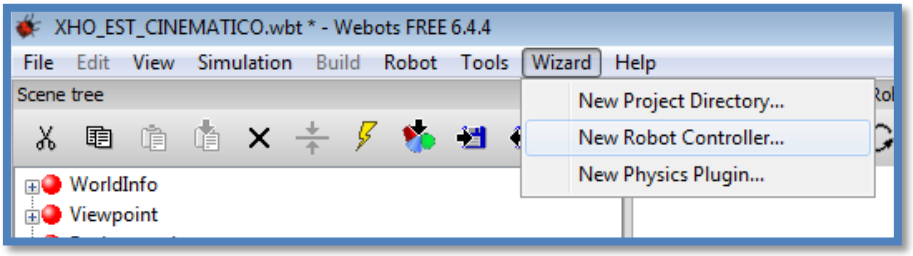


Figura 52. Crear nuevo controlador.

Cuando queremos crear un controlador, debemos hacer click en Wizard>New robot "controller" (ver figura 52). Entonces elegimos el lenguaje de programación (en nuestro caso elegiremos lenguaje C) desarrollarlo y seguidamente el nombre que le vamos a dar a ese nuevo controlador "Robot_XHO" (ver figura 53). Automáticamente Webots creará una carpeta con su nombre dentro de la carpeta "controllers" del directorio de trabajo.

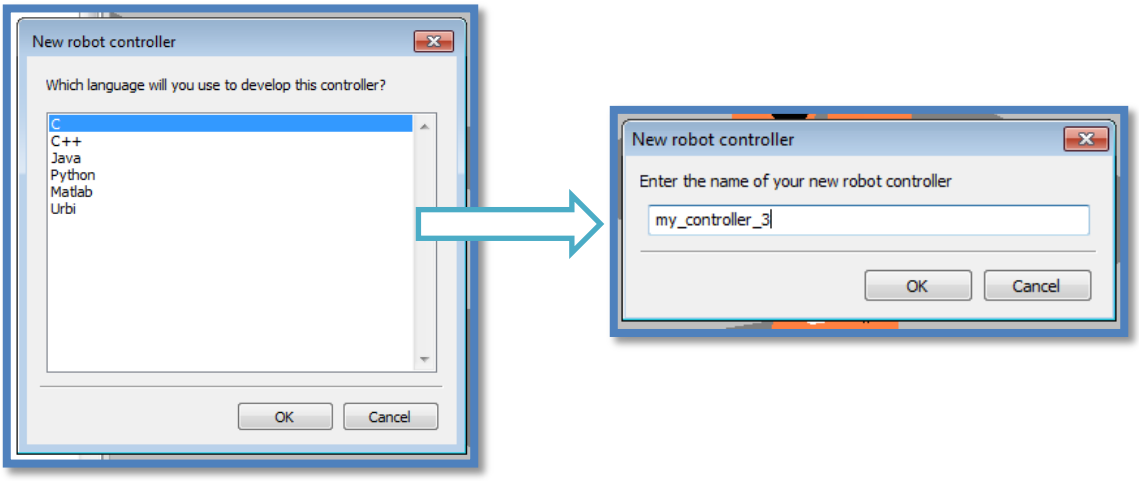


Figura 53. Elegir un lenguaje de programación y asignar el nombre del controlador.

De esta forma creamos el controlador y aparecerá una nueva ventana "Text Editor", en esta ventana permite acceder al código del programa para editar y compilar el controlador.

Esta ventana de edición de texto podemos mostrarla desde Windows>Text editor (ver figura 54), en la ventana principal.

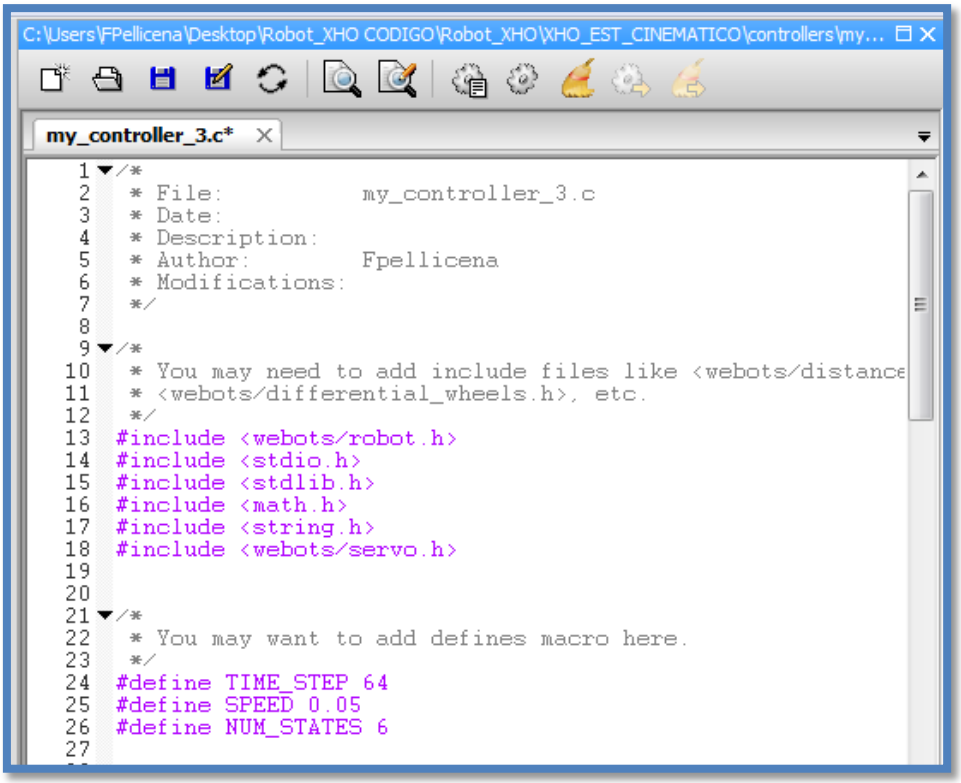


Figura 54. La ventana del Text Editor.

7.3.2.3 Console

La ventana ***"Console"*** permite desplegar resultados que se están ejecutando en el robot nos va a permitir mostrar todos los datos que necesitemos conocer en el momento de ejecución. para tener una idea de los valores que tiene al interactuar con el medio ambiente y poder utilizar de una mejor manera los actuadores.

Como todas las demás, podemos mostrarla desde Windows>Log. Todo lo volcado a esta ventana durante la ejecución se guarda en un fichero webots.log en nuestro directorio de trabajo. De esta manera podemos seguir la ejecución y conocer los datos que necesitemos en cualquier momento (ver figura 55).

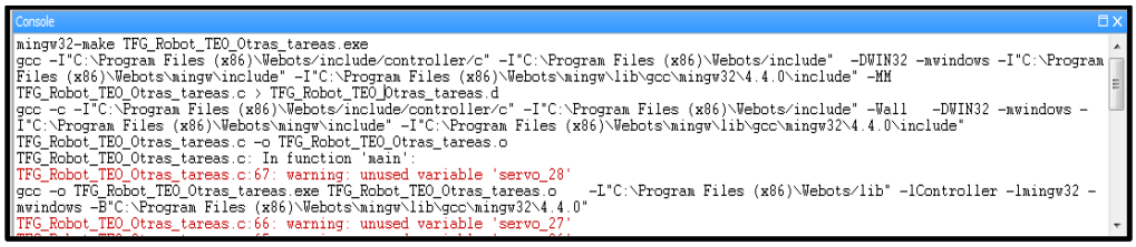


Figura 55. La ventana de Console.

7.3.2.4 3D Window

La ventana "3D window" muestra los componentes que se establecieron en el "Scene Tree" de una forma visual, y cuando el código finalmente es ejecutado se puede "observar" todo lo que hace el robot en este mundo virtual (ver figura 56).

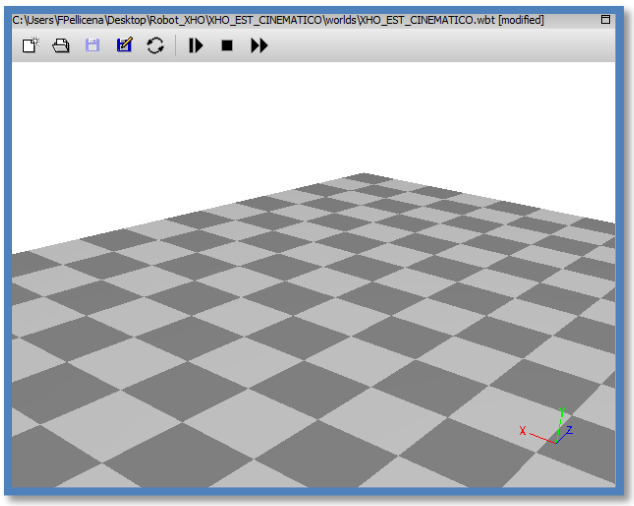


Figura 56. La ventana 3D.

7.3.3 VRML

En Webots, los objetos se encuentran compuestos por uno o varios nodos. Estos nodos son descritos utilizando el lenguaje VRML, un nodo es la estructura mínima indivisible de un fichero VRML y tiene como misión la de definir las características de un objeto o bien las relaciones entre distintos objetos.

VRML (Virtual Reality Modeling Language) es el lenguaje de programación que se utiliza en Webots para modelar el mundo virtual donde se simulará el comportamiento de los robots. Este lenguaje posibilita la descripción de objetos 3D de formas sólidas situadas y orientadas de determinada forma. Para crear estos mundos de realidad virtual se utilizan ficheros de texto, cuya extensión será siempre .wrl, los cuales pueden ser creados mediante cualquier editor o procesador de textos. Además, existe la posibilidad de utilizar programas de diseño gráfico, los cuales generan automáticamente ficheros en formato VRML, que en nuestro caso es la herramienta Webots.

Los nodos a su vez contienen campos que describen propiedades. Todo campo posee un tipo determinado y no se puede inicializar con valores de otro tipo. De este modo, cada tipo de nodo tiene una serie de valores predeterminados para todos sus campos, de forma que cuando se utilice en una escena sólo se deben indicar aquellos campos que se quieran modificar.

Los campos pueden ser simples, o bien indicando a vectores u otros nodos. En la figura 57 se presenta un ejemplo de programa VRML en Webots, el cual contiene un nodo del tipo “Transform” (transformación) que define el sistema de coordenadas de sus hijos con respecto al sistema de coordenadas del padre. Este nodo contiene tres campos: uno de translación (“translation”) con respecto al nodo padre, uno de rotación (“rotation”) con respecto a sus ejes, y uno de escala (“scale”). Como hijo del nodo “Transform” se tiene el nodo “Shape” (forma), que genera formas tridimensionales en el mundo virtual. Este nodo contiene dos campos: uno de nombre “appearance” (apariencia) el cual define el color y texturas de dicha forma, y otro de nombre “geometry” (geometría), que puede ser una figura geométrica simple (caja, cono, cilindro, etc.) o una extrusión.

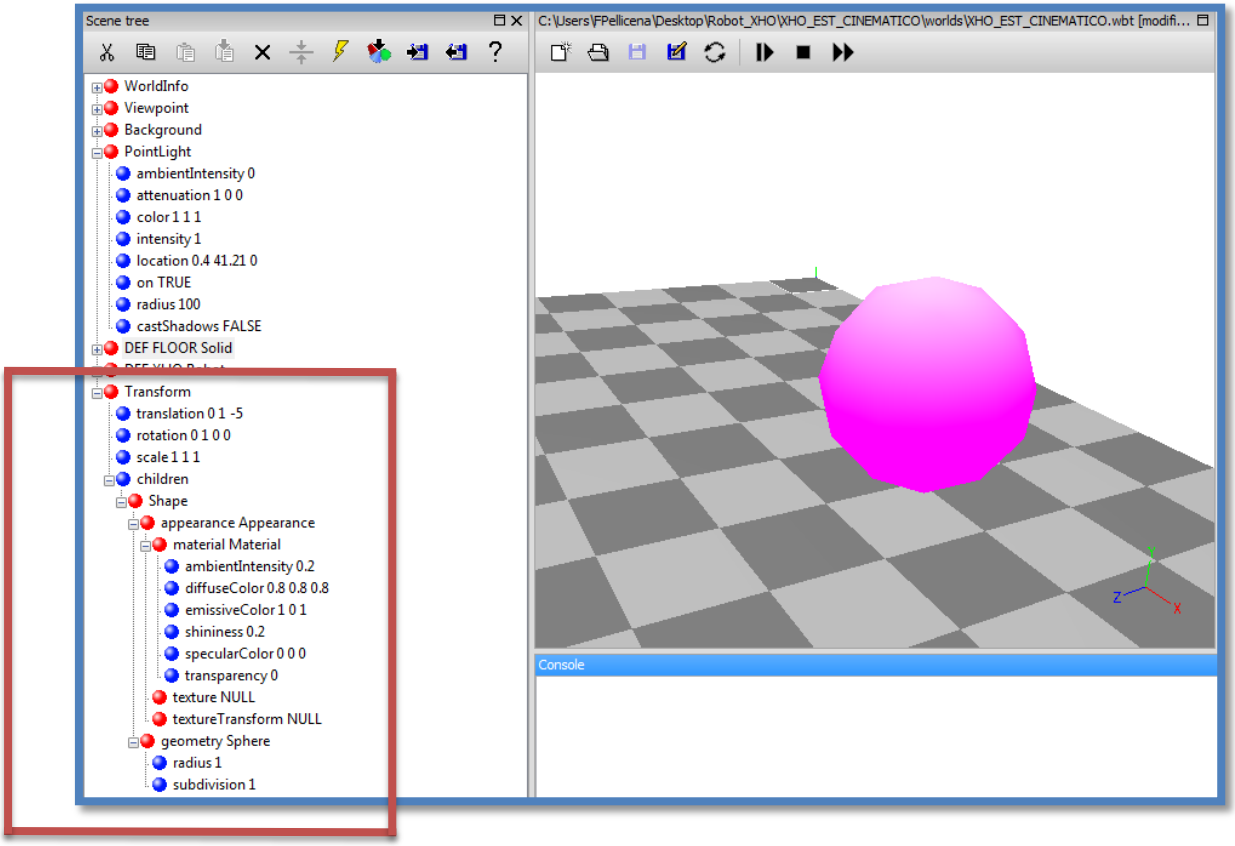


Figura 57. Ejemplo de VRML, creación de una esfera.

7.3.4 Algunos nodos importantes

La construcción del modelo es necesario conocer los siguientes nodos: nodo "Robot", nodo "Servo", nodo "Transform", nodo "Shape", nodo "Solid".

7.3.4.1 Nodo Robot

El nodo "***Robot"*** puede ser utilizado como base para la construcción de un robot, por ejemplo, un robot articulado, un robot humanoide, un robot con ruedas, etc. (ver figura 58). Para construir un robot es necesario conocer los siguientes campos:

- **Controlador ("*controller*"):** Un controlador es un archivo binario, el cual es usado para controlar a un robot que se ha descrito en un archivo "world". Los controladores son almacenados en subdirectorios de Webots en el directorio "controllers". Los controladores pueden ser de diferentes tipos:
  - Archivos binarios de java (.class)
  - Archivos de C (.c y .cpp)
  - Archivos de C++ (.c y .cpp)

  En este proyecto utilizamos lenguaje C.

- ***ControllerArgs:*** Es una cadena que contiene los argumentos (separados por espacios) que se pasan a la función "main ()" de programación de C / C + + o el "main ()" de programación en Java.
- **Sincronización (*"synchronization"*):** si el valor es "TRUE" (valor predeterminado), el simulador está sincronizado con el controlador, si el valor es "FALSE", el simulador se ejecuta sin controlador. La función "wb_robot_get_synchronization()" se utiliza para leer el valor de un programa de controlador.
- **Batería (*"battery"*):** Este campo debe contener tres valores: la primera corresponde al nivel de energía actual del robot en julios (J), la segunda es la energía máxima en julios, y la tercera es la velocidad de recarga de energía en vatios ([W] = [J] / [s]). El simulador actualiza el valor primero, mientras que los otros dos permanecen constantes. Importante: cuando el valor actual de la energía llega a cero, el robot se para todo el movimiento.
- **Consumición de CPU (*"CPUConsumption"*):** El consumo de energía de la CPU (unidad central de procesamiento) del robot en vatios.
- **"SelfCollision":** Se establece como "TRUE", le permitirá la detección de colisiones en el robot. Esto es útil para los complejos robots articulados para que el controlador no impiden las colisiones internas. Sin embargo, la activación de colisión pueden disminuir la velocidad de simulación. Cuando "selfCollision" es "FALSE" (por defecto), Webots no intenta detectar las posibles colisiones internas en un robot. En este caso, el controlador puede impedir que las diferentes partes del cuerpo de un robot cruzándose entre sí.

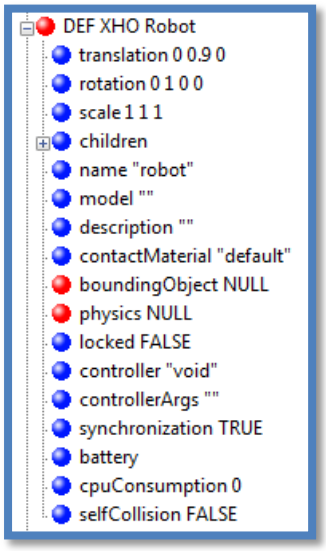


Figura 58. Nodo de Robot en Webots.

7.3.4.2 Nodo Servo

Un nodo "***servo"*** se utiliza para añadir una junta (1 grado de libertad (GDL)) en una simulación mecánica. El movimiento de servo puede ser de tipo "rotacional" o "lineal". Un "servo rotacional" se utiliza para simular un movimiento de rotación, como un motor eléctrico o una bisagra. Un "servo lineal" se utiliza para simular un movimiento de deslizamiento, como un motor lineal, un pistón, un hidráulico / cilindro neumático, un resorte o un amortiguador (ver figura 59). Para construir un robot y sus servos es necesario conocer los siguientes campos:

- **Tipo ("*type"*):** Este campo especifica el tipo de movimiento de servo, puede ser "rotacional" o "lineal" (ver figura 60).
- **Máxima velocidad ("*maxVelocity"*):** En este campo se especifica el límite de la velocidad del servo. La velocidad se puede cambiar en tiempo de ejecución con la función "wb_servo_set_velocity()". El valor debe ser siempre positivo (el valor predeterminado es 10).
- **Máxima fuerza ("*Maxforce"*):** En este campo se especifica el límite superior de la fuerza del servo motor. Es una fuerza que está disponible en el motor para llevar a cabo los movimientos requeridos. Esta fuerza se puede cambiar en tiempo de ejecución con la función "wb_servo_set_motor_force()".El valor debe ser siempre positivo (el valor predeterminado es 10).
- ***ControlP***: En este campo se especifica el valor inicial del parámetro P. Cuando el valor de P es mayor, se produce un error muy pequeño, y por lo tanto, se obtiene un sistema más sensible. Sin embargo, si el valor de P es demasiado alto, el sistema puede llegar a ser inestable. Si el valor de P es pequeños, se necesitará más pasos/tiempo de simulación para alcanzar la posición de destino, pero el sistema es más estable. Este valor se puede cambiar en tiempo de ejecución con la función "wb_servo_set_control_p()".
- **Aceleración ("*aceleration"*):** Define la aceleración por defecto de la Pcontrolador. Si el valor es -1 significa que la aceleración no está limitada por el "P_controlador".
- **Posición ("*position"*):** Representa la posición actual del servo, en radianes o en metros. Si es un servo rotacional, en radianes. Si es un servo lineal, en metros. Para conocer la posición actual se utiliza la función "wb_servo_set_position()".
- ***"MinPosition"*** y ***"MaxPosition"*** especifican la mínima posición y la máxima posición que puede alcanzar el servo motor.

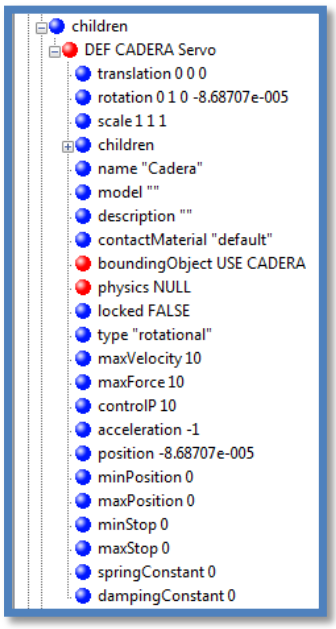


Figura 59. Nodo Servo en Webots

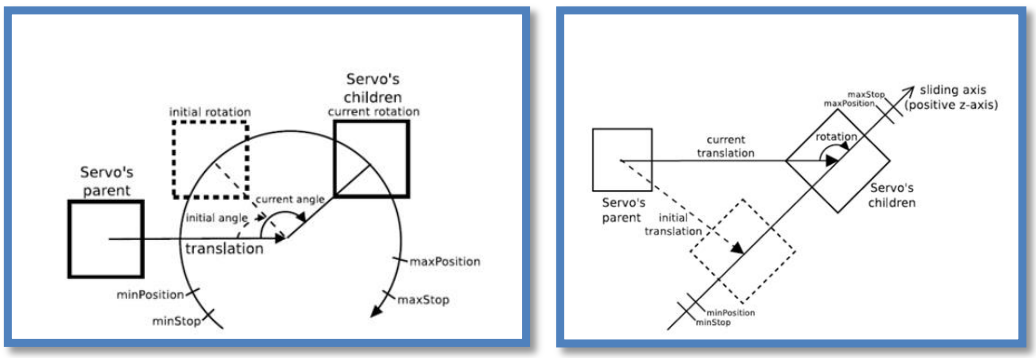


Figura 60. Movimiento de tipo rotacional (izda,) y lineal (dcha) del servo.

7.3.4.3 Nodo Transform

El nodo "***Transform"*** es un nodo de agrupación que define el sistema de coordenadas de nodo hijo con respecto al sistema de coordenadas de nodo padre (ver figura 61).

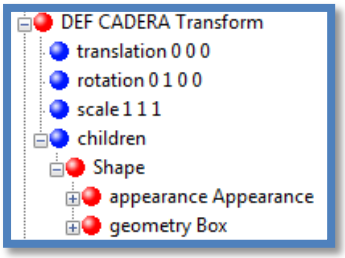


Figura 61. Nodo Transform en Webots

7.3.4.4 Nodo Shape

El nodo "***Shape"*** tiene dos campos, la apariencia ("***appearance"***) y la geometría ("***geometry"***), que se utilizan para crear objetos en el mundo (ver figura 62).

7.3.4.4.1 Appearance

El campo "***Appearance"*** contiene un nodo Apariencia que especifica los atributos visuales (por ejemplo, material y textura) que se aplicará a la geometría.

7.3.4.4.2 Geometry

El campo de "***Geometry"*** contiene un nodo de geometría: Caja, Cono, Cilindro, Extrusión, IndexedFaceSet, IndexedLineSet, plano o una esfera.

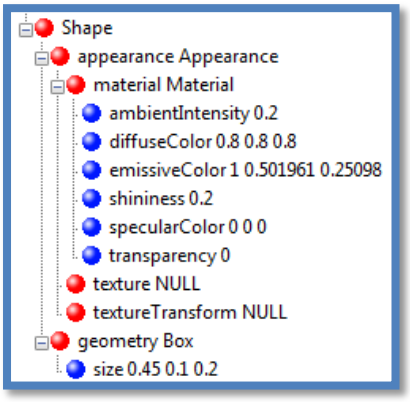


Figura 62. Nodo Shape en Webots

7.3.4.5 Nodo Solid

Un nodo Solid representa un objeto con propiedades físicas, tales como dimensiones, materiales y la masa (ver figura 63). Robot es una subclase de Solid. En la ventana 3D, los nodos sólidos se pueden manipular (arrastrar, levantar, girar, etc .) con el ratón. Es necesario conocer los siguientes campos:

- **Nombre (*"name"*):** Es el nombre del sólido. Se utilizado la función "wb_robot_get_device()" para obtener el nombre.
- ***"BoundingObject"*:** Especifica las primitivas geométricas utilizadas para la detección de colisiones. Si el campo "***boundingObject"*** es NULL, **entonces no se realiza la detección de colisiones y ese objeto puede pasar a través de cualquier otro objeto ("transparente"),** por ejemplo, el suelo, los obstáculos y otros robots. Teniendo en cuenta que si el campo "boundingObject" es NULL, el campo de la "Physics" también debe ser NULL.
- ***Physics:*** Este campo es un campo opcional, ya que un nodo "Physics" que se utiliza para modelar las propiedades físicas de este sólido. Un nodo "Physics" deben añadirse cuando conocemos la gravedad, la inercia, fricción y fuerzas, etc. Si el campo de la física es NULL entonces Webots simula este objeto en modo cinemático. Teniendo en cuenta que si este campo no es NULL, entonces el campo "boundingObject" debe ser especificado.
- **Bloqueado (*"locked"*):** Si es TRUE, el objeto sólido no se puede mover con el ratón.

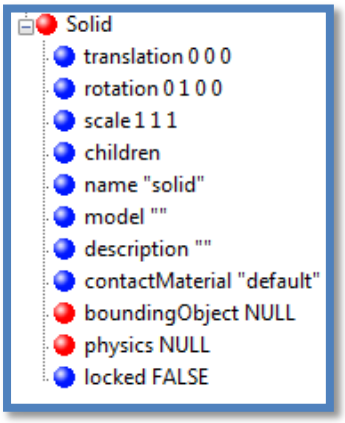


Figura 63. Nodo Solid en Webots

7.4 Anexo D. Programación II.

En este Anexo se recoge todos cálculos de todas las etapas estudiadas, para la obtención de los ángulos del modelo cinemático.

- **Etapa 0:** Posición inicial, posición de reposos en la que encontramos el robot.

- **Etapa 1:** Posición en la cual la pierna derecha se flexiona, mientras que la pierna izquierda permanece en reposo. A continuación, en la figura 19 puede verse el sistema estudiado mediante resolución gráfica ayudado de unas ecuaciones que completan la resolución analíticamente.

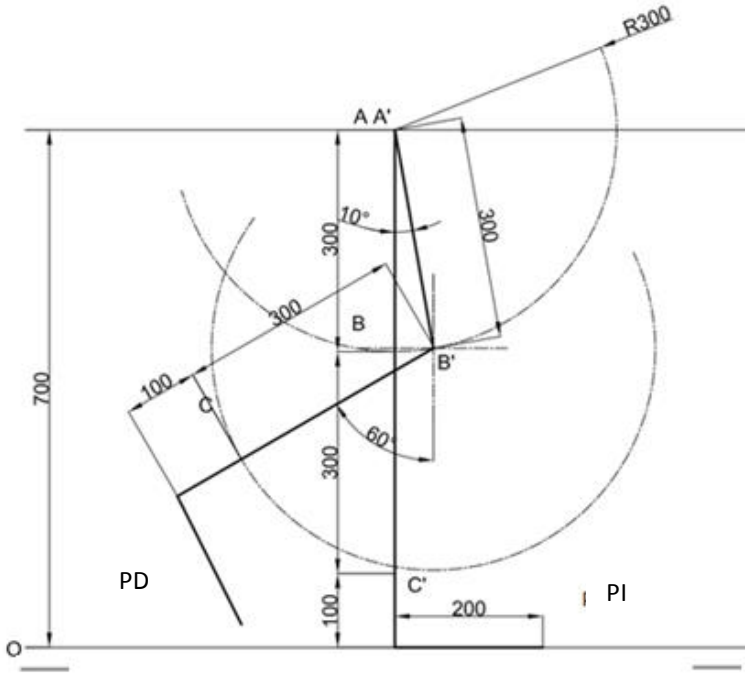


Figura 19. Estudio gráfico de la etapa 1, en AutoCAD.

En este estado solo modificarán su posición los Servos A' y B' correspondientes a la pierna derecha.

$$300\cos\theta A' + 300\cos\theta B' + d(OC') = 700$$

Para= θA´= 10° y d(OC´)=250 → θB´= 60°

- **Etapa 2:** En esta etapa, la pierna izquierda se flexiona esta vez por delante del cuerpo, mientras que la pierna derecha empieza a desplegar el talón del suelo

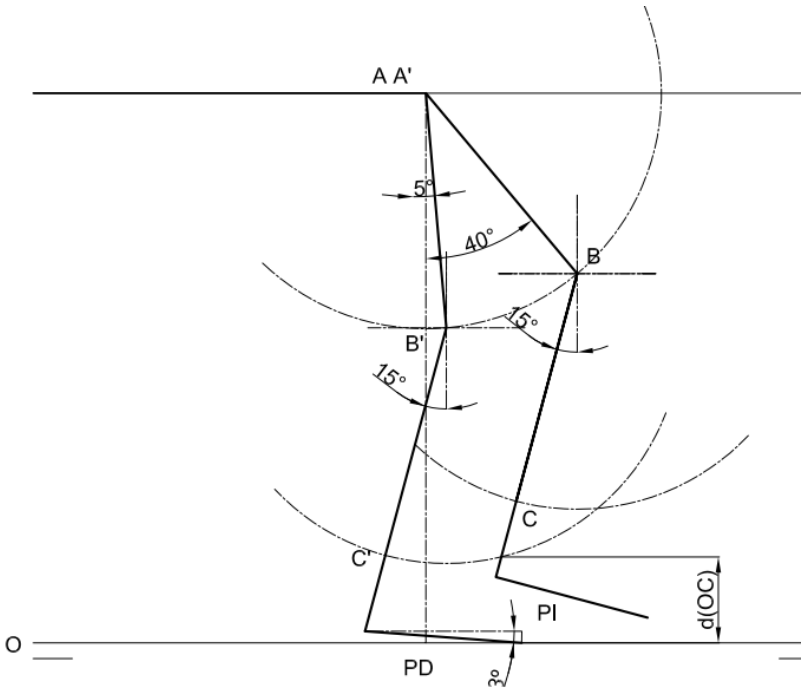


Figura 62. Estudio gráfico de la etapa 2, en AutoCAD.

La pierna derecha no es estudiada ya que vamos a tomar los ángulos obtenidos en la resolución gráfica, por estar en el aire y ser menos importante, en esta etapa. El interés de esta etapa reside en la pierna izquierda por estar apoyada y tener que calcular los ángulos para estar en contacto y mantener el Servo "A" de la pierna a una distancia constante de 700, ya que al igual que la cadera este punto no varía en el eje "y".

Pierna izquierda: θA= 40° y θB= 15° (el signo de los ángulos, lo tendremos en cuenta en el siguiente paso, introducir valores en el código)

Pierna izquierda;

$$300\cos\theta A + 300\cos\theta B + d(OC) = 700$$

Con los ángulos obtenidos de la resolución gráfica θA=5° y θB=15° obtenemos d(OC)=111,36mm este valor implica que el pie se encuentra apoyado sólo sobre la punta del mismo. Por lo tanto deberemos de estudiar el valor del Angulo θC, este valor sí que es de gran interés. El cálculo se hace en base a la figura 19, detalle de la figura 20.

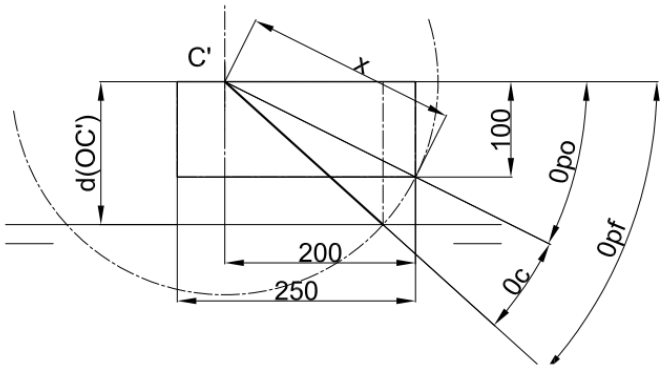


Figura 20. Detalle del pie derecho de la etapa 2, en AutoCAD.

$$\theta C = \sin^{-1}\left(\frac{d(OC)}{x}\right) - \tan^{-1}\left(\frac{100}{200}\right)$$

Donde, aplicando el teorema de Pitágoras:

$$x = \sqrt{100^2 + 200^2}$$

Obteniendo, θC=3,3°

- **Etapa 3:** La pierna izquierda se encuentra próxima a hacer contacto con el suelo con el talón del pie, mientras que la pierna derecha está a punto de despegarse del suelo.

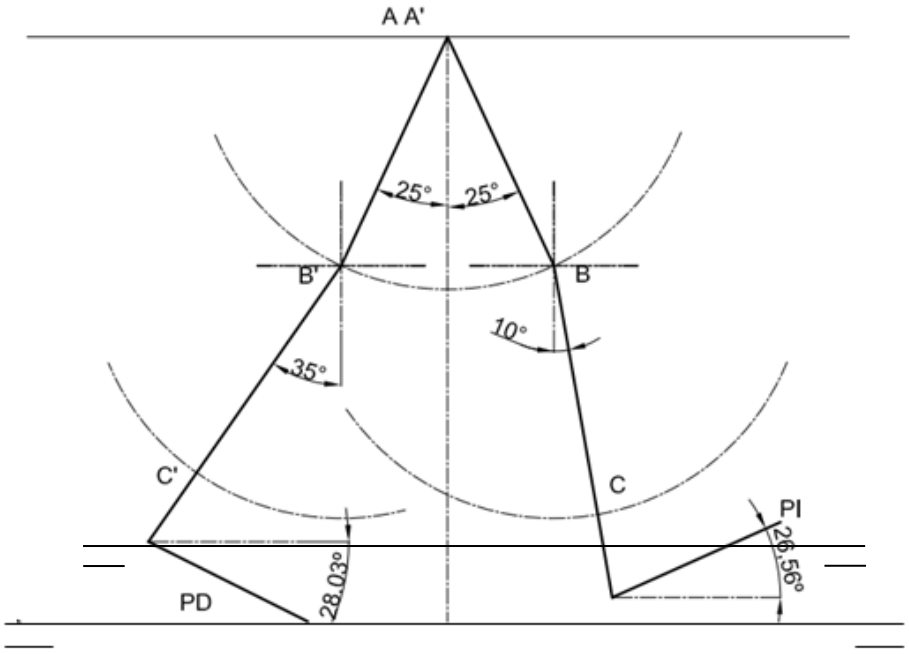


Figura 21. Estudio gráfico de la etapa 3, en AutoCAD.

En esta etapa, las dos piernas son de interés, ya que ambas se encuentran en contacto con el suelo. En situaciones límite, la pierna izquierda, mantiene contacto con la punta del pie, mientras que la pierna derecha con el talón.

En primer lugar estudiamos la pierna izquierda:

$$300\cos\theta A + 300\cos\theta B + d(OC) = 700$$

Con θA=25° y queriendo aumentar d(OC) de 111,36mm hasta 180mm, para provocar la flexión de la pierna izquierda. Obtenemos θB=35°, mientras que θC=28,03° tomando x=180mm.

$$\theta C = \sin^{-1}\left(\frac{d(OC)}{x}\right) - \tan^{-1}\left(\frac{100}{200}\right)$$

Por otra parte, en el estudio de la pierna derecha suponiendo θA=25° y queriendo obtener un avance de 180 mm.

$$300\sin\theta A' + 300\sin\theta B' = 180mm$$

Se obtiene θB=10°, mientras que θC= 26,56° al obtener la siguiente ecuación del detalle que aparece en la Figura 22.

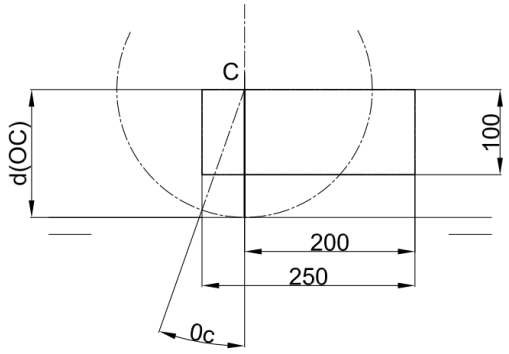


Figura 22. Detalle del pie izquierdo de la etapa 3, en AutoCAD.

$$\theta C = -\tan^{-1}\left(\frac{50}{100}\right)$$

Los ángulos de las siguientes etapas, son los mismos calculados hasta el momento pero intercambiando las piernas.

- **Etapa 4:** Es similar a la etapa 2, pero esta vez es la pierna derecha la que se encuentra en la posición de reposo, mientras que la izquierda realiza la pequeña flexión.

- **Etapa 5:** Similar a la etapa 3, pero cambiando las funciones de las piernas.

- **Etapa 6:** Similar a la etapa 4, pero, cambiando las funciones de las piernas. De esta etapa se vuelve a la etapa 1 y así se completa un ciclo.

De este modo, en la Tabla 1 recogemos todos los ángulos que deben tomar los distintos servos para cada estado, tal y como se ha escrito el código, este es el valor que habrá que introducir.

Los ángulos han sido calculados respecto al sistema de coordenadas globales, y así es como lo exige el programa. En el programa se introducen en radianes (SI) tal y como aparecen a continuación.

	Estado 1	Estado 2	Estado 3	Estado 4	Estado 5	Estado 6
PI_A	0	-0,087	0,43	-0,174	-0,698	-0,436
PI_B	0	0,261	0,61	1,04	0,261	-0,17
PI_C	0	0,057	0,489	0	0	-0,463
PD_A	-0,174	-0,698	-0,436	0	-0,087	0,43
PD_B	1,04	0,261º	-0,17	0	0,261	0,61
PD_C	0	0	-0,463	0	0,057	0,489

Tabla 1. Ángulos obtenidos para cada servo de cada pierna en cada etapa estudiada.

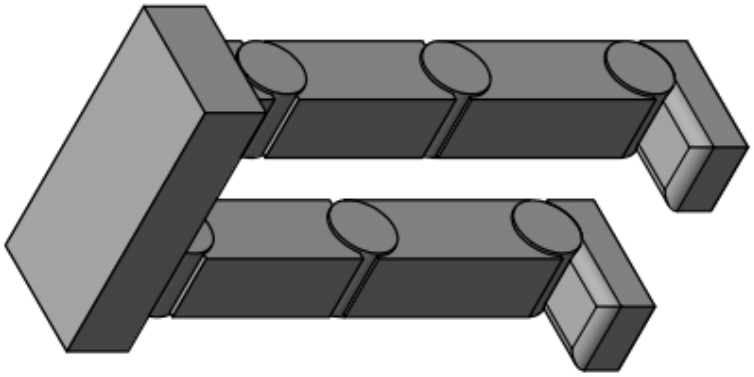
7.5 Anexo E. Código completo implentado en Webots.

```
/*
 * File:         my_controller_3.c
 * Date:
 * Description:
 * Author:       Fpellicena
 * Modifications:
 */

/*
 * You may need to add include files like <webots/distance_sensor.h> or
 * <webots/differential_wheels.h>, etc.
 */
#include <webots/robot.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <webots/servo.h>


/*
 * You may want to add defines macro here.
 */
#define TIME_STEP 64
#define SPEED 0.05
#define NUM_STATES 6

/*
 * You should put some helper functions here
 */
 int max_pulse(int p1, int p2, int p3, int p4, int p5, int p6)
 {
  int parray[] = {p1, p2, p3, p4, p5, p6};

  int i = 0;
  int mp = parray[i];
  for(i = 1; i < 6; i++)
  {
    if(mp < parray[i]){
      mp = parray[i];
    }
  }

  return mp;
 }

/*
```

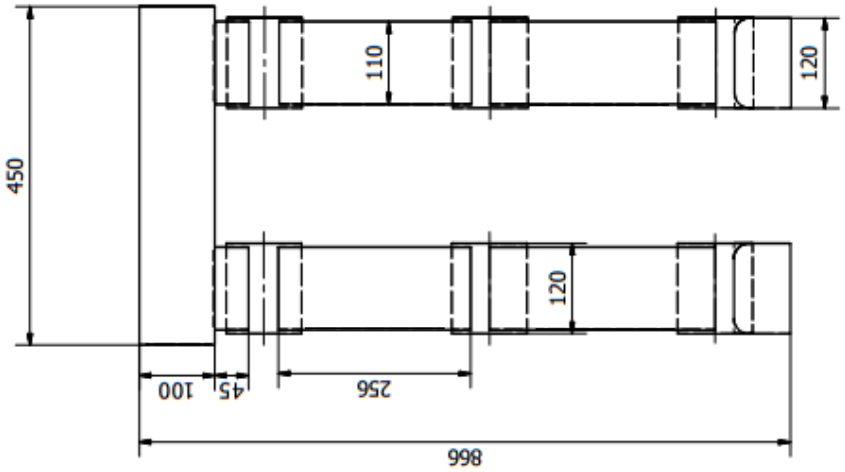
```
 * This is the main program.
 * The arguments of the main function can be specified by the
 * "controllerArgs" field of the Robot node
 */
int main(int argc, char **argv)
{
 /* necessary to initialize webots stuff */
 wb_robot_init();

 /*
  * You should declare here WbDeviceTag variables for storing
  * robot devices like this:
  *  WbDeviceTag my_sensor = wb_robot_get_device("my_sensor");
  *  WbDeviceTag my_actuator = wb_robot_get_device("my_actuator");
  */
  int i = 0, t = 0, state = 0;
 double pulse_ratio = 500.0/1.571;

 double pi_servo_a_current_position = 0.0;
 double pi_servo_b_current_position = 0.0;
 double pi_servo_c_current_position = 0.0;
 double pd_servo_a_current_position = 0.0;
 double pd_servo_b_current_position = 0.0;
 double pd_servo_c_current_position = 0.0;

 double pi_servo_a_positions[NUM_STATES] = {0, -0.087, 0.43, -0.174, -0.698, -0.436};
 double pi_servo_b_positions[NUM_STATES] = {0, 0.261, 0.61, 1.04, 0.261, -0.17};
 double pi_servo_c_positions[NUM_STATES] = {0, 0.057, 0.489, 0, 0, -0.463};
 double pd_servo_a_positions[NUM_STATES] = {-0.174, -0.698, -0.436, 0, -0.087, 0.43};
 double pd_servo_b_positions[NUM_STATES] = {1.04, 0.261, -0.17, 0, 0.261, 0.61};
 double pd_servo_c_positions[NUM_STATES] = {0, 0, -0.463, 0.057, -0.057, 0.489};

 int ppia;
 int ppib;
 int ppic;
 int ppda;
 int ppdb;
 int ppdc;

 int npulse;

 WbDeviceTag pi_servo_a = wb_robot_get_device ("PI_SERVO_A");
 WbDeviceTag pi_servo_b = wb_robot_get_device ("PI_SERVO_B");
 WbDeviceTag pi_servo_c = wb_robot_get_device ("PI_SERVO_C");
 WbDeviceTag pd_servo_a = wb_robot_get_device ("PD_SERVO_A");
 WbDeviceTag pd_servo_b = wb_robot_get_device ("PD_SERVO_B");
 WbDeviceTag pd_servo_c = wb_robot_get_device ("PD_SERVO_C");

 /* main loop */
 do {

  wb_servo_set_velocity(pi_servo_a, SPEED);
```

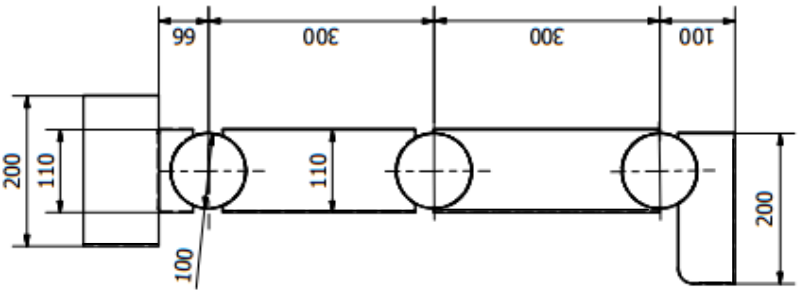
```
    wb_servo_set_velocity(pi_servo_b, SPEED);
    wb_servo_set_velocity(pi_servo_c, SPEED);
    wb_servo_set_velocity(pd_servo_a, SPEED);
    wb_servo_set_velocity(pd_servo_b, SPEED);
    wb_servo_set_velocity(pd_servo_c, SPEED);

    for(state = 0; state < NUM_STATES; state++)
    {
      ppia = (int)(fabs(pi_servo_a_positions[state] - pi_servo_a_current_position) * pulse_ratio);
      ppib = (int)(fabs(pi_servo_b_positions[state] - pi_servo_b_current_position) * pulse_ratio);
      ppic = (int)(fabs(pi_servo_c_positions[state] - pi_servo_c_current_position) * pulse_ratio);
      ppda = (int)(fabs(pd_servo_a_positions[state] - pd_servo_a_current_position) *
pulse_ratio);
      ppdb = (int)(fabs(pd_servo_b_positions[state] - pd_servo_b_current_position) *
pulse_ratio);
      ppdc = (int)(fabs(pd_servo_c_positions[state] - pd_servo_c_current_position) *
pulse_ratio);

      npulse =max_pulse(ppia,ppib,ppic,ppda,ppdb,ppdc);

      //Estado state
      for (t=0; t<npulse; t++)
      {
        //Pierna izquierda
        wb_servo_set_position(pi_servo_a, pi_servo_a_positions[state]);
        wb_servo_set_position(pi_servo_b, pi_servo_b_positions[state]);
        wb_servo_set_position(pi_servo_c, pi_servo_c_positions[state]);
        //Pierna derecha
        wb_servo_set_position(pd_servo_a, pd_servo_a_positions[state]);
        wb_servo_set_position(pd_servo_b, pd_servo_b_positions[state]);
        wb_servo_set_position(pd_servo_c, pd_servo_c_positions[state]);

        wb_robot_step(TIME_STEP);
      }

      pi_servo_a_current_position = pi_servo_a_positions[state];
      pi_servo_b_current_position = pi_servo_b_positions[state];
      pi_servo_c_current_position = pi_servo_c_positions[state];
      pd_servo_a_current_position = pd_servo_a_positions[state];
      pd_servo_b_current_position = pd_servo_b_positions[state];
      pd_servo_c_current_position = pd_servo_c_positions[state];
    }
    i++;
  } while (i<=10);

  /* Enter here exit cleanup code */

  /* Necessary to cleanup webots stuff */
  wb_robot_cleanup();

  return 0;
}
```

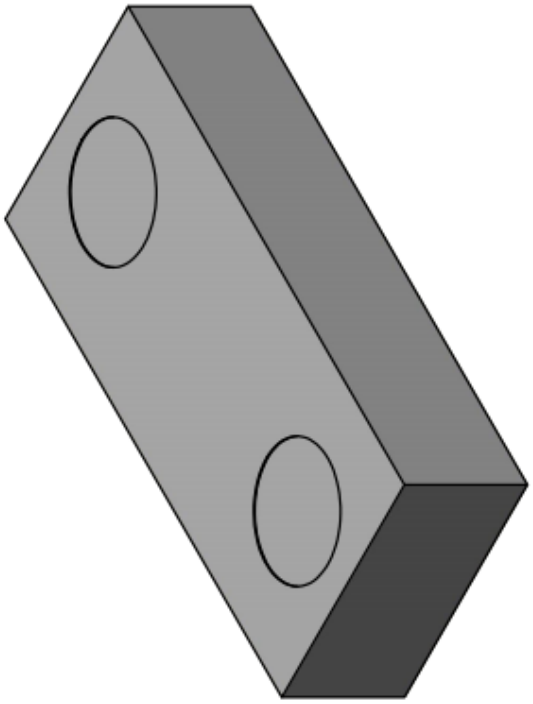
7.6 Anexo F. Planos.



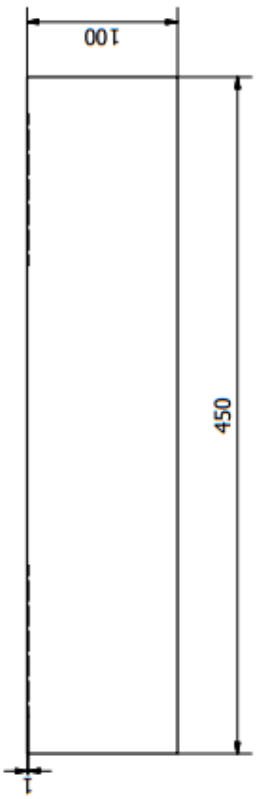
Diseño de F. PELLICENA	Revisado por	Aprobado por	Fecha
Escuela de Ingeniería y Arquitectura Universidad Zaragoza			Fecha 13/06/2016
Ensamblaje			Edición
			Hoja 1 / 1

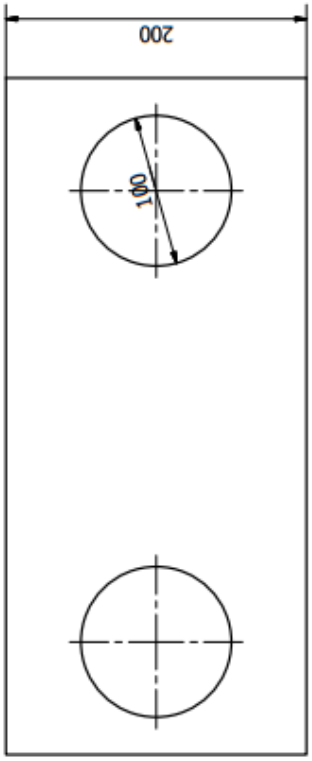


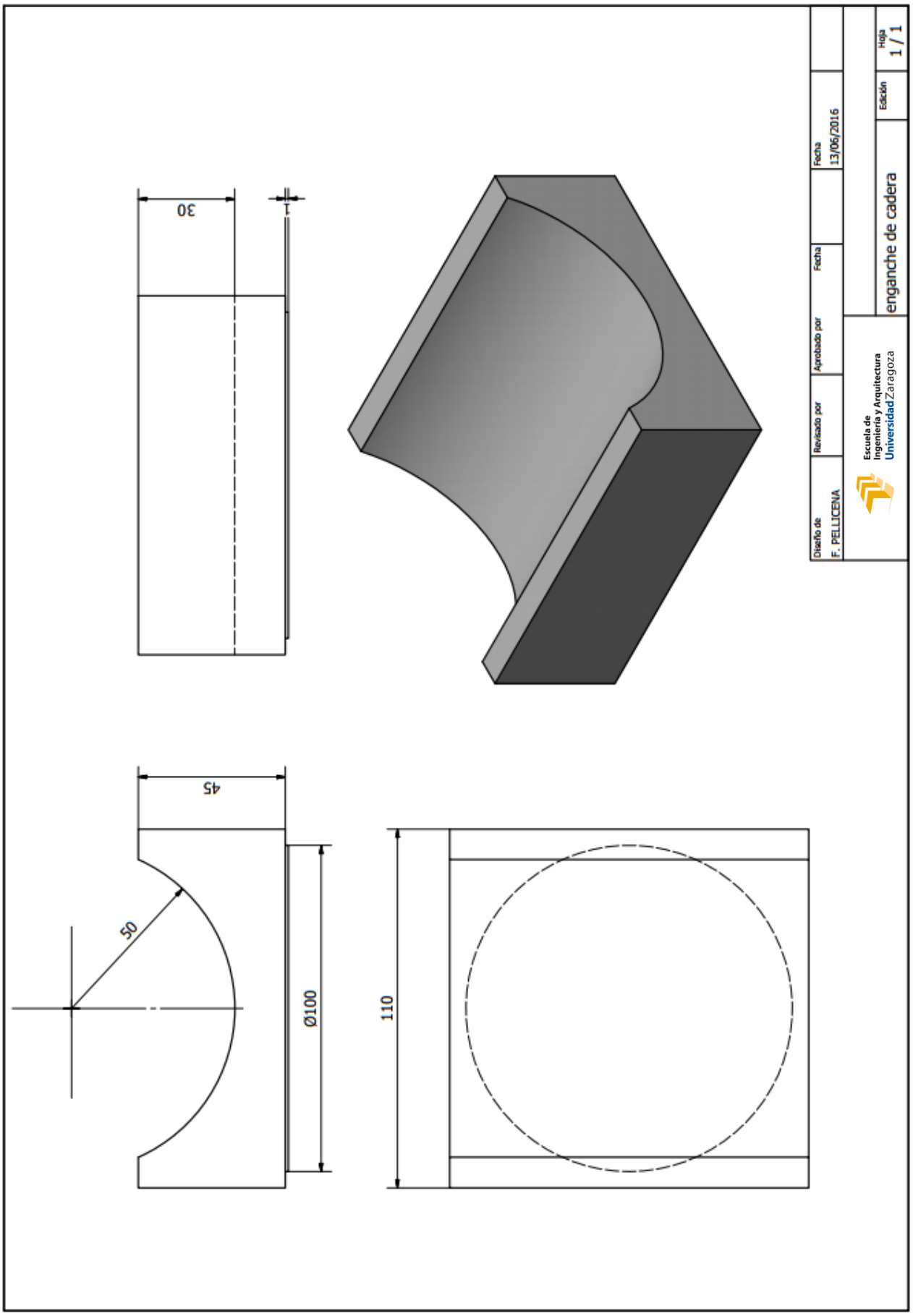


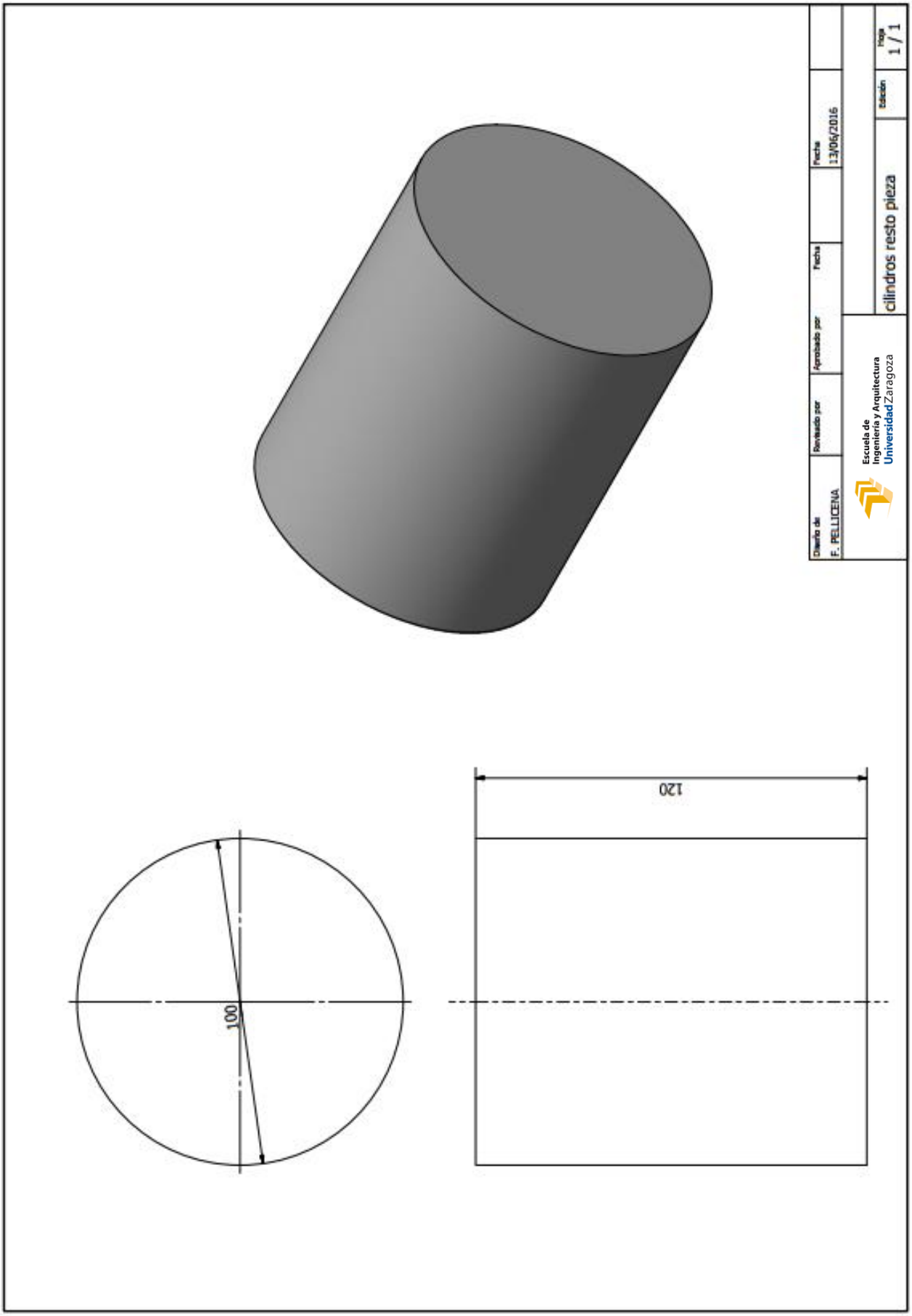


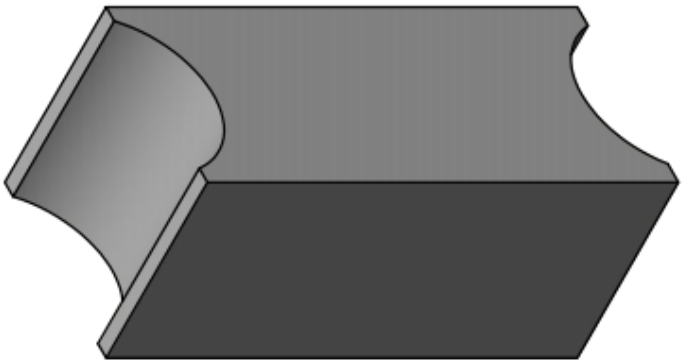
Diseño de F. PELLICENA	Revisado por	Aprobado por	Fecha
Escuela de Ingeniería y Arquitectura Universidad Zaragoza			Fecha 13/06/2016
	Cadera		Edición
			Hoja 1 / 1

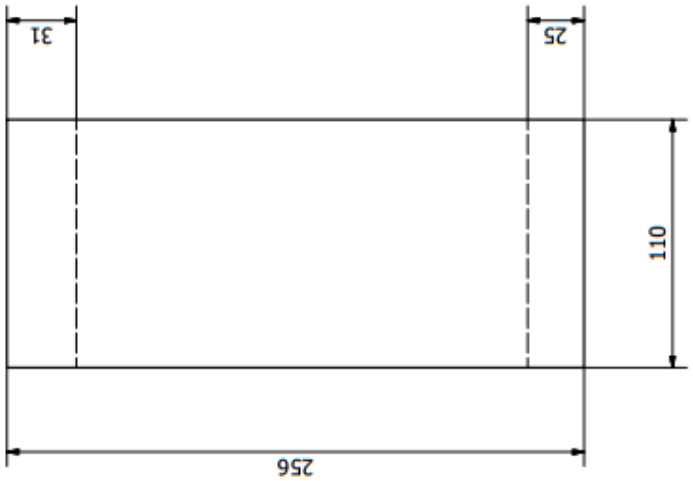


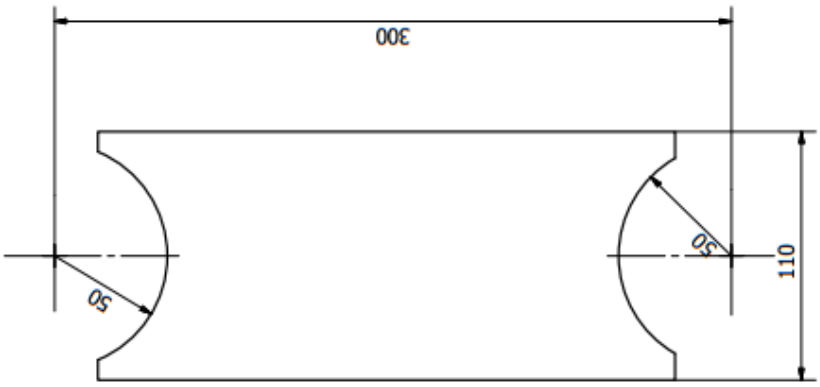



30
1
45
50
Ø100
110
enganche de cadera
Fecha 13/06/2016
Edición
Hoja 1 / 1
Aprobado por
Fecha
Revisado por
Diseño de F. PELLICENA
Escuela de Ingeniería y Arquitectura Universidad Zaragoza

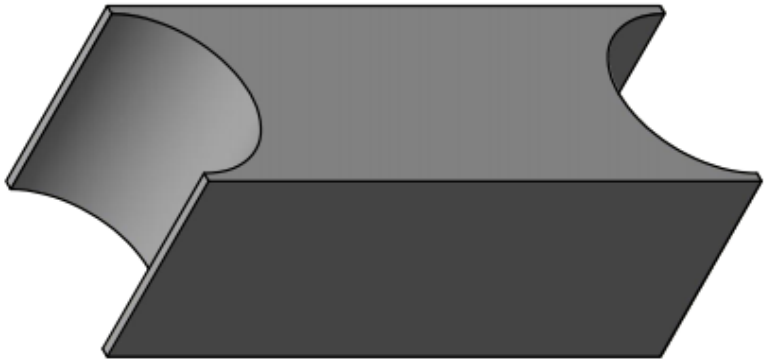

Hoja 1 / 1
Edición
Fecha 13/06/2016
cilindros resto pieza
Escuela de Ingeniería y Arquitectura Universidad Zaragoza
Diseño de F. PELLICENA
120
100

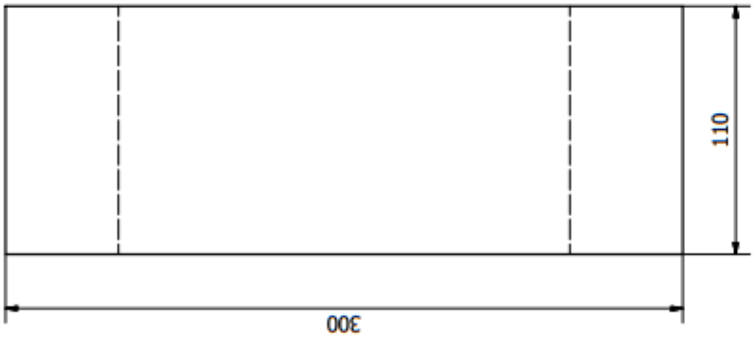




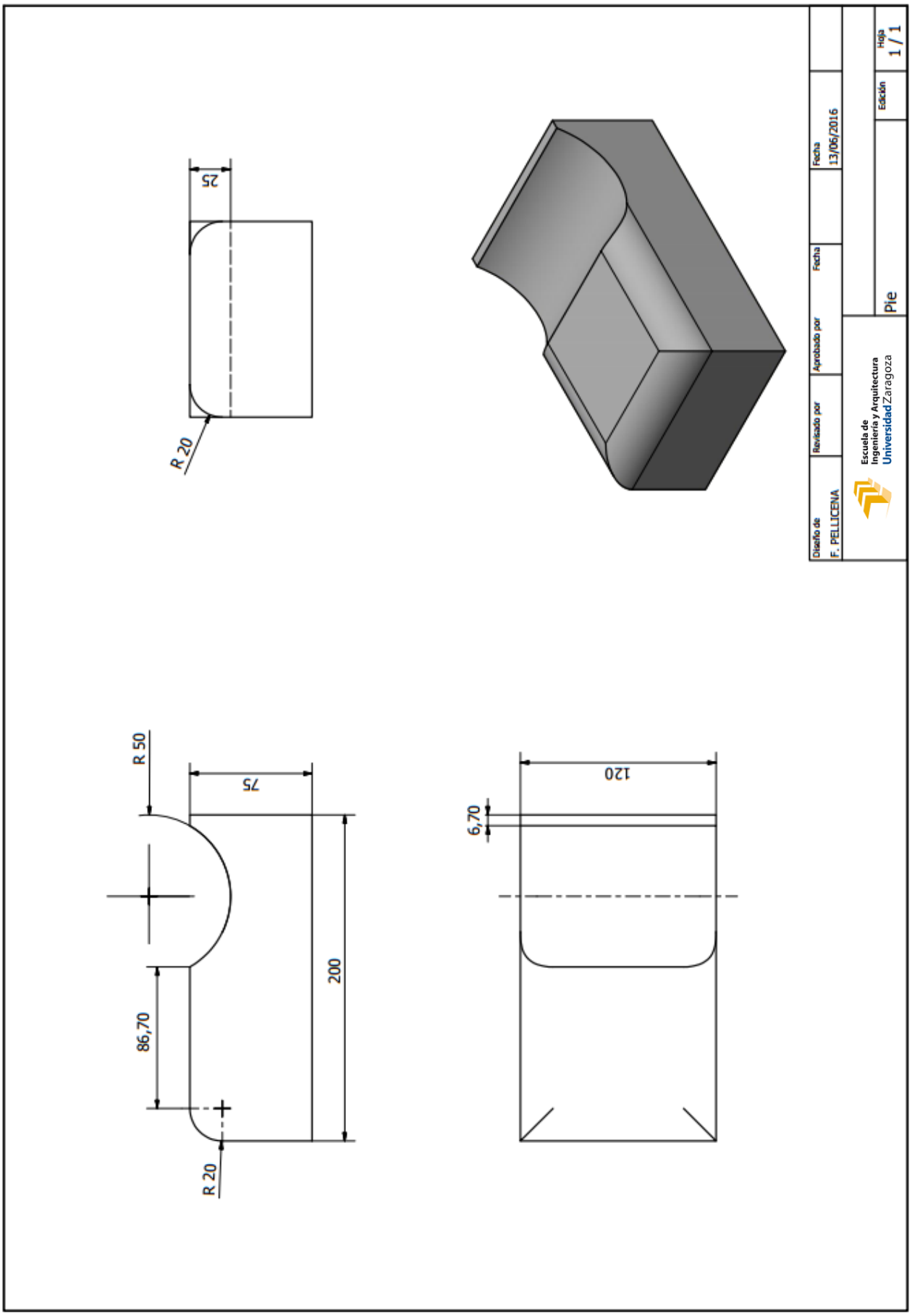


Diseño de F. PELLICENA	Revisado por	Aprobado por	Fecha 13/06/2016
Escuela de Ingeniería y Arquitectura Universidad Zaragoza			
pierna superior			Edición
			Hoja 1 / 1





Diseño de F.PELLICENA	Revisado por	Aprobado por	Fecha 13/06/2016
Escuela de Ingeniería y Arquitectura Universidad Zaragoza			
pierna inferior			Edición
			Hoja 1 / 1


25
R 20
Pie
Hoja 1 / 1
Edición
Fecha 13/06/2016
Aprobado por
Fecha
Revisado por
Diseño de F. PELLICENA
Escuela de Ingeniería y Arquitectura Universidad Zaragoza
R 50
75
200
86,70
R 20
120
6,70

7.7 Anexo G. Versión XHO 2.0, modelo dinámico.

A continuación, se presenta una versión mejorada del modelo XHO en el software Webots estudiado en este trabajo, este modelo, bautizado XHO 2.0 no tiene codigo implementado, pero se espera de él que realice movimientos más complejos, ya que presenta mayor número de grados de libertad, concretamente presentrá 4 en el tronco y 8 GdL en cada pierna (4 zona cadera, 1 zona rodilla, 3 zona tobillo) esto hace que el nuevo modelo tenga un total de 20 grados de libertad, lo que aumenta notablemente la dificultad de controlarlo.

El objetivo principal de este incremento de grados de libertad, no es solo adquirir mayor complejidad en los movimientos, sino que es, que este numero de grados de libertad permitan resolver el problema del equilibrio estudiado en el modelo dinámico.

Por esta razón, podemos ver que se le ha añadido un tronco, el cual poseerá una masa para ayudar a la resolución del problema de equilibrio.

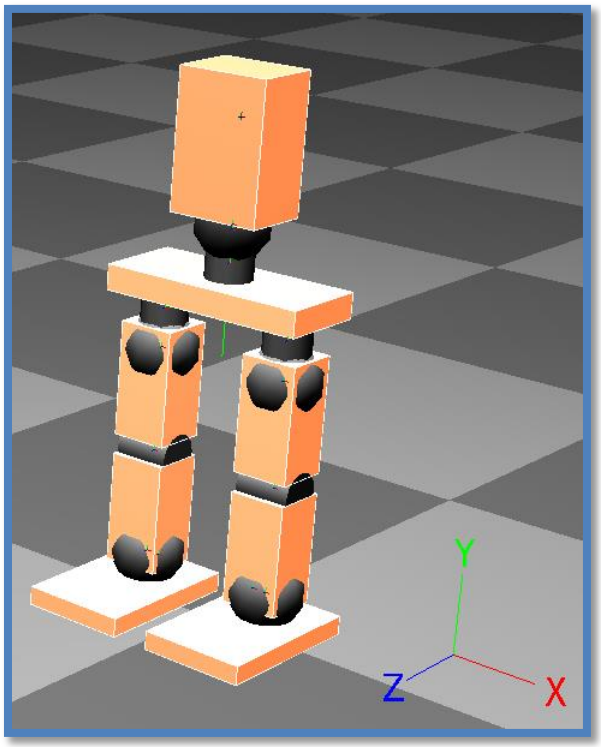


Figura 64. Modelo XHO 2.0, construido en Webots

8. Bibliografía

- "Construcción de un Robot Bípedo Basado en Caminado Dinámico". Autor:Ing. Cesar Humberto Guzmán Valdivia, Ing. en Mecatrónica por la Universidad Politécnica de Zacatecas. (Tesis de maestría en ciencias).

- YABIRO: PROTOTIPODE ROBOT BÍPEDOAUTÓNOMO. Autores: Miguel Albero, Francisco Blanes, Ginés Benet, José Simó, Pascual Pérez. Departamento de Informática, Sistemas y Computadoras. Universidad Politécnica de Valencia.

- Análisis Cinemático de un Bípedo con fases: Pie de soporte-Pie en movimiento. Autores: Enrique González Núñez, Alejandro Aceves López, M. Iván Ramírez Sosa Morán. IEEE 5º Congreso Internacional en Innovación y Desarrollo Tecnológico.

- Generación de trayectorias para marcha semiestática de un robot bípedo: diseño y pruebas experimentales. Autores: Rodríguez-Ángeles, Cruz-Villar, y Vite-Téllez.

- Generalized Biped Walking Control. - http://www.cs.ubc.ca/~van/papers/2010-TOG-gbwc/paper.pdf

- SimRobot- http://www.informatik.uni-bremen.de/simrobot/index_e.htm

- Gazebo- http://gazebosim.org/

- OpenHRP3- http://www.openrtp.jp/openhrp3/en/about.html

- Marilou Robotics Studio- http://www.anykode.com/index.php

- Microsoft Robotics Developer Studio 4 - http://es.wikipedia.org/wiki/Microsoft_Robotics_Studio

- Webots User Guide (release 6.4.4) copyright (c) 2011 Cyberbotics Ltd. All rights reserved - http://www.cyberbotics.com

- Webots Reference Manual (release 6.4.4) copyright (c) 2011 Cyberbotics Ltd. All rights reserved.- http://www.cyberbotics.com