Locomoción bípeda del robot humanoide Nao

Proyecto Final de Carrera

Memoria

Autor
Samuel Fernández Iglesias

Director
Josep M. Fuertes i Armengol

Noviembre 2009

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Universitat Politècnica
de Catalunya

Resumen

La *locomoción bípeda de robots humanoides*, es decir, aquellos que tienen morfología humana y las articulaciones necesarias para realizar movimientos parecidos a los humanos, es un desafío para el desarrollo tecnológico. Esta faceta de la ingeniería agrupa muchas ramas reunidas para un mismo objetivo: mecánica, automática, electrónica, informática, biología...

Este proyecto de investigación se propone como objetivo principal *desarrollar un algoritmo de control para conseguir que el robot humanoide Nao camine*, mejorando su algoritmo propio. Además, se creará una interfaz gráfica de usuario para el control de Nao y las funciones que se desarrollen.

Como fuente de inspiración y para ver el estado del arte se hará un análisis de otros algoritmos de locomoción existentes en robots, tales como:

- Generación de trayectorias imponiendo estabilidad.
- Aprovechamiento de dinámicas internas.
- Máquinas de estados.
- La propia locomoción humana.

Figura 1: *Robot Nao*

El algoritmo propio que se desarrollará, *Sammy's Walk*, *aprovechará las oscilaciones propias del sistema para generar la locomoción y así tener una dinámica de locomoción más eficiente*. Esto conllevará hacer estudios de frecuencias de oscilación del robot y atractores en el espacio de fases.

Este planteamiento, distante de la mayoría de algoritmos de locomoción basados en generación de trayectorias, es novedoso en su aplicación para robots con muchos grados de libertad activos, y puede sentar una base para investigaciones futuras.

El resultado de aprovechar estas dinámicas se traducirá en un ahorro energético, y un estilo de marcha parecido al humano.

Índice general

Índice de figuras

Índice de tablas

Índice de vídeos

El siguiente listado muestra todos vídeos que se citan a lo largo de la memoria. Éstos están incluidos en el CD adjunto y se han codificado con el códec libre XVID para mayor portabilidad. Además se indica un enlace para su versión on-line, al que se puede acceder directamente desde el documento electrónico de la memoria. En http://www.youtube.com/view_play_list?p=BC6A4C9D88DF0431 se encuentra la lista de reproducción con todos los vídeos recopilados.

Capítulo 1

Introducción

El presente proyecto de investigación plantea el estudio de locomoción bípeda de robots humanoides, es decir, aquellos que tienen morfología humana y las articulaciones necesarias para realizar movimientos parecidos a los humanos. Las investigaciones en este campo ya tienen un amplio desarrollo, y se han logrado resultados muy interesantes.

El objetivo central de este proyecto es *diseñar un algoritmo de locomoción bípeda para que el robot Nao camine, mejorando el propio de Nao.* En concreto, se pretende diseñar este algoritmo a través de inducir ciclos límites en el movimiento del robot y aprovechar las dinámicas propias del sistema para provocar su avance. A su vez, se plantean los siguientes subobjetivos, que conforman la estructura de este proyecto:

- Como fuente de inspiración y para ver el estado del arte, investigación de algoritmos de locomoción ya existentes y de la locomoción humana.
- Puesta en marcha de Nao, su interfaz con el ordenador, programación y simulación en realidad virtual.
- Estudio de ciclos límites en la oscilación del robot, y diseño de un algoritmo que los aproveche para inducir un movimiento de avance, esto es, hacer que camine.
- Estabilización de las oscilaciones mediante atractores en el espacio de fases.
- Análisis de prestaciones y resultados.

Como objetivo paralelo se desarrollará una interfaz de usuario para poner en uso las funciones principales de Nao y sus funciones de locomoción.

El enfoque estará orientado al diseño de los algoritmos de control, experimentación y análisis de resultados más que en una formulación dinámica-mecánica del robot como sistema multisólido.

La metodología que se seguirá será principalmente experimental. Como punto de partida se formularán hipótesis a partir de modelos simplificados. También se usarán ideas de la amplia bibliografía investigada, aunque ésta muchas veces no se podrá implementar en Nao o no obtendrá los resultados esperados.

Por tanto, para alcanzar el objetivo deseado, éste es, lograr que Nao camine, será necesario un desarrollo propio y novedoso. El diseño del algoritmo se hará desarrollando diferentes metodologías, sin ser ni mucho menos una búsqueda intuitiva al azar, sino más bien una experimentación basada en estudios sobre robots e introspección en la locomoción humana.

Es de esperar que al ser una implementación novedosa se encuentren problemáticas de diversa índole, que se deberán ir solucionando con ingenio y los recursos disponibles.

Capítulo 2

Robots bípedos y algoritmos de control

La locomoción de robots con piernas, humanoides o similares, no es una novedad para el desarrollo tecnológico. Los primeros trabajos se iniciaron hace 40 años. Por tanto, antes de comenzar la aproximación propia a esta cuestión, es un deber analizar algunas de las soluciones y tecnologías que ya se han encontrado e implementado.

Este capítulo tiene como objetivo ver el estado del arte de la locomoción de robots bípedos, y a su vez servir de fuente de inspiración para el posterior diseño de un algoritmo propio.

Del vasto trabajo de investigación que se ha realizado en este campo, se estudiarán diversos algoritmos de control representativos, así como algunos ejemplos de robots reales y simulaciones interesantes. Sólo se hará un análisis descriptivo de los aspectos más importantes, sin entrar en demostraciones ni formulaciones complejas del robot como sistema dinámico multisólido. Para ampliar la información se puede consultar la bibliografía referenciada.

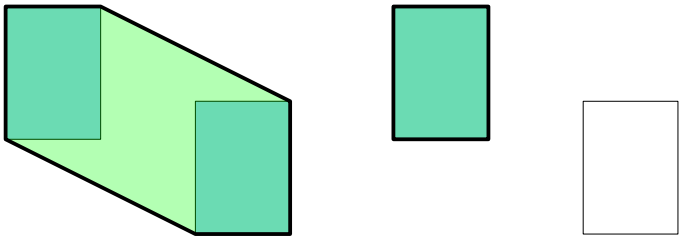
2.1. *Zero-Moment Point* (ZMP)

Esta técnica de control fue introducida por primera vez hace más de 35 años [24], y es actualmente una de las más extendidas. Establece un criterio de estabilidad dinámica para el robot que permite generar patrones de locomoción. Este concepto fue aplicado con éxito por primera vez en 1984 en *Waseda University, Laboratory of Ichiro Kato*, en el robot *WL-10RD*, el primero equilibrado dinámicamente, y desde entonces en otros múltiples robots.

ZMP (*Zero Moment Point*, punto de momento nulo) se puede definir como *el punto p_{ZMP} en el suelo tal que el momento neto de las fuerzas externas no tiene componente sobre los ejes horizontales*. Cuando p_{ZMP} existe dentro del polígono de soporte[1], el contacto entre el suelo y el pie es estable. Cuanto más cercano esté p al centro de la superficie de soporte, más robustez se conseguirá. Cuando ZMP está fuera del polígono de soporte, el robot se inclina rotando sobre alguno de los bordes de dicho polígono. El criterio de que ZMP exista dentro del polígono de soporte es condición necesaria y suficiente para garantizar la estabilidad dinámica del robot.

De forma intuitiva, *la condición ZMP asegura que el movimiento del cuerpo será tal que el pie estará plano en el suelo* y por tanto no caerá.

[1] Área conexa formada con los puntos de contacto sobre el suelo (ver Figura 2.1). En caso de soporte sobre un único pie, el polígono de soporte es este mismo. En caso de soporte con los dos pies, el polígono de soporte abarca la región de los pies y el parte del área entre ambos.



(a) *Soporte doble (dos pies)* **(b)** *Soporte simple (un pie)*

Figura 2.1: *Polígono de soporte según fase de locomoción*

2.1.1. Formulación matemática

A continuación se hará una demostración sencilla del criterio ZMP, así como la condición que permite generar trayectorias de locomoción estables.

La clave está en suponer que el robot está apoyado sobre un único pie. Se impone como condición de equilibrio que el pie no esté en condiciones de volcada inminente, es decir, que la reacción del suelo esté aplicada en el interior del pie, no en uno de sus bordes.

Si se aísla el sistema "pie", sobre él actúa la reacción del suelo (la fuerza $\mathbf{R}$ y el momento $\mathbf{M}$), y la de la articulación del tobillo (la fuerza $\mathbf{F_T}$ y el momento $\mathbf{M_T}$) (ver Figura 2.2). La reacción del suelo sobre el "pie", considerado un sólido rídigo, se puede reducir a una fuerza y un momento aplicados en un punto determinado. En general, habrá tres componentes de fuerza R_x, R_y, R_z, y un tres componentes de momento M_x, M_y, M_z. Igualmente en la articulación del tobillo habrá tres componentes de fuerza F_{Tx}, F_{Tx}, F_{Tz} y de momento M_{Tx}, M_{Tx}, M_{Tz}.

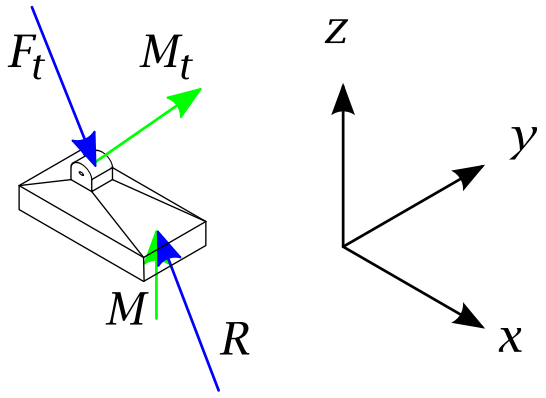


Figura 2.2: *Fuerzas y momentos sobre el pie*

Dadas las características del enlace pie-suelo, y suponiendo que no hay deslizamiento, se cumplirá que las componentes horizontales de $\mathbf{F_T}$ (F_{Tx}, F_{Ty}) siempre se verán compensadas por las componentes horizontales de $\mathbf{R}$ (R_x, R_y), esto es, la fricción. Lo mismo pasará con la componente vertical del momento M_{Tz} y M_z.

Por tanto, el problema se reduce a estudiar la componente vertical de las fuerzas (F_{Tz} y R_z) y las componentes horizontales de los momentos (M_{Tx}, M_{Ty} y M_x, M_y). Dado que la fuerza R_z es unidireccional, siempre se puede encontrar un punto p tal que la reacción del suelo R_z, M_x, M_y se reduzca únicamente a una fuerza, R_z, siendo las componentes horizontales del momento compensadas por la ubicación de la fuerza. Este punto es el *Zero-Moment Point ZMP*.

Cabe destacar que esta definición es muy parecida a la de centro de presión (CdP). CdP se define como el *punto tal que el efecto de la fuerza y momento de la reacción del suelo equivale a únicamente una fuerza aplicada en ese punto*. A continuación se explicará la diferencia entre ambos conceptos.

Si el pie está en equilibrio, es decir, no está en condiciones de volcada inminente, ZMP se encontrará dentro del polígono de soporte, y será coincidente con CdP por definición. En condiciones de volcada inminente, ZMP y CdP están ubicados sobre un borde del polígono de soporte. Cuando el momento externo de la articulación continúe aumentando, se pasará a condiciones de volcada, fuera del equilibrio. CdP continuará estando en el borde del polígono de soporte (no puede existir fuera de éste). Sin embargo, existe un momento que está haciendo girar el pie. ZMP existirá fuera del polígono de soporte en un punto tal que el momento neto de todos los enlaces sea nulo.

También existe una interpretación que es la que permite utilizar este concepto en el diseño de trayectorias: CdP se define en función de la interacción de las fuerzas de reacción del suelo, ZMP se puede definir en función de *una trayectoria computada*. Para calcular la posición de ZMP a partir de una trayectoria en primer lugar hay que imponer equilibrio estático de momentos en el pie, y analizar las componentes horizontales:

$$\sum_i \mathbf{F_i} = 0, \quad \sum_i \mathbf{M_{P_i}} = 0 \tag{2.1}$$

$$\mathbf{R} + \mathbf{F_T} + \mathbf{m_{pie}g} = 0 \Rightarrow \mathbf{R} = -\mathbf{F_T} - \mathbf{m_{pie}g} \tag{2.2}$$

$$\overrightarrow{\mathbf{OP}}_{\mathbf{ZMP}} \wedge \mathbf{R} + \overrightarrow{\mathbf{OG}} \wedge \mathbf{m_{pie}g} + \mathbf{M_T} + \mathbf{M} + \overrightarrow{\mathbf{OT}} \wedge \mathbf{F_T} = 0 \tag{2.3}$$

Si se estudian únicamente las componentes horizontales, $\mathbf{M}$ no aparece en la ecuación al tener únicamente componente vertical. Sustituyendo la ecuación 2.2 en la ecuación 2.3, y restringiéndola a las componentes horizontales se obtiene:

$$(\overrightarrow{\mathbf{OP}}_{\mathbf{ZMP}} \wedge (-\mathbf{F_T} - \mathbf{m_{pie}g}))^{\mathbf{H}} + \overrightarrow{\mathbf{OG}} \wedge \mathbf{m_{pie}g} + \mathbf{M_T^H} + (\overrightarrow{\mathbf{OT}} \wedge \mathbf{F_T})^{\mathbf{H}} = 0 \tag{2.4}$$

Para una trayectoria computada, de la cual se conozcan las variables articulares, velocidades y aceleraciones, a través de una formulación dinámica inversa (Newton-Euler por ejemplo) se pueden determinar los momentos y fuerzas en cada articulación, y en concreto para el tobillo del pie soporte. La ecuación generalizada para calcularlas es:

$$\boldsymbol{\tau} = \mathbf{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \mathbf{g}(\boldsymbol{q}) + \boldsymbol{\tau_{ext}} \tag{2.5}$$

siendo $\boldsymbol{\tau}$ el vector de fuerzas generalizado, $\mathbf{H}$ la matriz de inercia, $\mathbf{C}$ incluye los términos de coriolis, $\mathbf{g}$ los términos de gravedad y $\boldsymbol{\tau_{ext}}$ las fuerzas externas. Como resultados de esta ecuación se pueden encontrar en concreto los valores de $\mathbf{M_T}$ y $\mathbf{F_T}$. Sustituyéndolos en la ecuación 2.4 se puede determinar la posición del punto p_{ZMP}. Si este está dentro del polígono de soporte, la trayectoria es estable. Si está fuera, no se podrá cumplir el equilibrio, existirá un momento neto y por tanto el pie volcará sobre uno de sus bordes. En este caso será necesario redefinir la trayectoria inicial de tal forma que se consiga cumplir la condición ZMP. Esta condición expuesta para el soporte con un solo pie se puede extrapolar para el soporte doble.

La condición de ZMP impone que el pie esté estáticamente equilibrado. Pero como para su cómputo se utilizan las dinámicas del resto del cuerpo, se dice que esta condición es de *estabilidad dinámica, ya que las trayectorias de la locomoción son estables.*

2.1.2. Algoritmo

El algoritmo que se presentará a continuación permite generar trayectorias que cumplan la condición ZMP. El procedimiento consta de los siguientes pasos [4]:

1. Se definen una serie de posiciones de los pasos deseados.

2. Para encontrar las trayectorias de un paso se sigue un proceso de optimización. Para pasar de trayectorias a configuraciones articulares es necesario usar la cinemática inversa, y para pasar de movimientos articulares a fuerzas la dinámica inversa. Las restricciones que debe cumplir la optimización deben ser:

   - Condición ZMP (equilibrio dinámico).
   - Cumplimiento de las posiciones deseadas para cada paso.
   - Mínimos y máximos movimientos articulares.
   - Criterios estilísticos: ya que el modelo a resolver es de dimensión muy elevada, es necesario definir más criterios. Éstos pueden ser del estilo trayectoria del pie cicloidal, altura de las caderas dentro de un rango...
   - La función a optimizar puede incluir aspectos energéticos (minimizar la energía aportada por los actuadores en un ciclo), de robustez (ZMP lo más cercano posible al centro del polígono de soporte), y otros dependientes de cada caso concreto.

Además, la posición del ZMP puede constituir una variable de control efectiva para asegurar estabilidad.

Sobre esta base se pueden implementar mejoras, como por ejemplo calcular las trayectorias y el ZMP *on-line*, o usar información de otros medios para calcular trayectorias futuras. El robot HRP-2 implementa este y otros criterios [23]. Aunque quizá el ejemplo más representativo del algoritmo ZMP es el robot ASIMO de Honda [8].

2.1.3. ASIMO

ASIMO (acrónimo de *Advanced Step in Innovative Mobility* – paso avanzado en movilidad innovadora), es un robot humanoide creado por la empresa Honda [8], la última y novedosa versión se presentó en el año 2005 (ver Figura 2.3). Quizá sea uno de los ejemplos más representativos de los robots caminadores bípedos basados en el algoritmo ZMP. El vídeo [2.1.3 ASIMO] contiene una demostración del robot caminando en sus diferentes modos.

(a) *Evolución de ASIMO (1986 a 2005)* **(b)** *Versión 2005*

Figura 2.3: *Robot ASIMO*

El funcionamiento se basa en generar un patrón de movimiento ideal para las articulaciones que cumpla la condición ZMP, y mover las articulaciones según éste. Cuando hay un desajuste entre el patrón ideal y la realidad (ya sea por perturbaciones o por el terreno) entran en funcionamiento las siguientes tres estrategias de control para prevenir una caída, y consecuentemente lograr estabilizar la locomoción:

- **Control de la reacción del suelo:** se controla la fuerza ejercida en los pies para detectar las irregularidades en el terreno, las perturbaciones que provocan y así poder corregirlas.
- **Control del ZMP objetivo:** Se hacen las correcciones necesarias sobre las articulaciones de tal forma que se sitúe lo más rápido posible en la posición ZMP ideal.
- **Control de la posición del siguiente paso:** Se hacen los ajustes necesarios a la trayectoria planificada de tal manera que el siguiente paso sea en el sitio óptimo para compensar la perturbación.

La última versión de ASIMO mide 130 cm, pesa 54 kg, tiene 34 grados de libertad y una batería que dura aproximadamente 25 min, aparte de todo un arsenal de sensores inerciales, de presión y de fuerza. Camina de forma normal a 2,7 km/h, y puede alcanzar corriendo (con fase aérea) los 6 km/h. Puede correr girando y evitar deslizamiento de los pies. Sin duda, es uno de los robots más avanzados que se han diseñado en laboratorios de investigación.

2.1.4. Ventajas e inconvenientes

Las ventajas de aplicación del concepto ZMP y los algoritmos derivados podrían ser resumidas en:

- En sí mismo ZMP constituye una metodología sistemática para la generación de trayectorias para la locomoción.
- Se asegura que las *trayectorias* de locomoción sean *dinámicamente estables*.
- Constituye una variable de control *on-line*. Aunque las trayectorias se planeen generalmente *off-line*, lógicamente existen perturbaciones. Se puede establecer un control articular tal que en caso de que ZMP se desvíe de su posición calculada regrese a ésta.
- En caso que la perturbación no se consiga compensar y ZMP salga del polígono de soporte, el cálculo del grado de separación de ZMP permite tener una medida de la perturbación. Ésta se puede corregir entonces modificando la trayectoria de tal manera que el nuevo paso se sitúe de tal forma que el nuevo polígono de soporte incluya ZMP, volviendo a recuperar la estabilidad.

Los siguientes puntos se plantean como inconvenientes al ZMP, siempre recordando que los mejores robots que caminan usan este método:

- La resolución del algoritmo es de alta complejidad. Con el crecimiento de capacidad de cálculo de los ordenadores no supondrá un problema en el futuro.
- El uso de cinemática inversa también comporta problemas. Debido a que se necesita gran precisión para tener una trayectoria estable, los errores numéricos pueden tener implicaciones muy perjudiciales de desequilibrado.
- El uso cinemática y dinámica inversas puede tener problemas por singularidades, ya que cerca de ellas se producen errores numéricos muy importantes. Los más comunes son divisiones por cero que conllevan explosiones numéricas, y por tanto comportamientos bruscos del robot.
- Dependiendo de la configuración (articulación en su límite, soporte sobre un pie, sobre dos, sobre un borde de un pie...) cambian los grados de libertad del sistema, y por tanto deben usarse matrices de dinámica diferentes. Esto eleva la complejidad muchísimo. La otra opción sería evitar configuraciones cercanas a singularidades, pero esto limitaría la movilidad "natural" del robot.

Por último, sería interesante reflexionar en lo siguiente: ZMP busca que las *trayectorias* de locomoción sean *dinámicamente estables*, ¿es realmente necesario? ¿No se podría buscar un sistema de locomoción tal que el ciclo límite de los sucesivos pasos fuera estable, aunque las *trayectorias fueran inestables*? Efectivamente sí se puede y de hecho, tal como se verá, los humanos no cumplen la condición ZMP en todas las fases de la marcha. Llegados a este punto resulta interesante analizar trabajos realizados sobre el denominado caminar pasivo (*Passive Walking*), y los trabajos que se derivan aplicados a caminadores activos.

2.2. *Passive & Dynamic Walking*

2.2.1. *Passive Walking*

El concepto de locomoción pasiva *(Passive Walking)*, es decir, sin actuadores, fue introducido por primera vez en 1988 por Tad McGeer [15][18][16][17], quien estudió y construyó un caminador pasivo (ver vídeo [2.2.1 Passive Walker]).

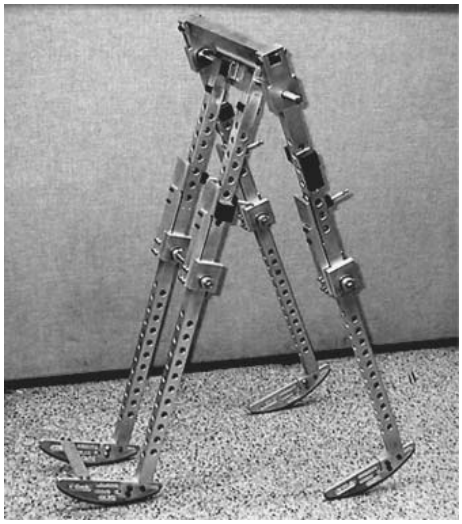


Figura 2.4: *Copia del mecanismo inventado por McGeer*

La virtud de este mecanismo es que *no necesita energía externa ni ningún tipo de control para caminar* por una pendiente, el movimiento viene "propulsado" por la energía gravitacional de la bajada. El sistema actúa como dos péndulos acoplados. La pierna soporte es un péndulo invertido, y la pierna oscilante es un péndulo normal unido a la pierna soporte. Con la distribución de masas y longitudes adecuada, se consigue un ciclo límite estable, una trayectoria nominal que se repite. Inicialmente el mecanismo no tenía rodillas, pero una mejora posterior implementó esta articulación manteniendo la estabilidad. El movimiento lateral se inhibe al contar con cuatro piernas.

McGeer hizo un estudio exhaustivo de la mecánica y dinámica del caminador. Demostró las condiciones necesarias para asegurar la estabilidad del ciclo límite. Cabe destacar una diferencia sustancial respecto al algoritmo ZMP: el caminador pasivo describe *trayectorias inestables* (por definición, se comporta como un péndulo invertido inestable), manteniendo el ciclo límite estable.

También es de gran interés el hecho que los movimientos de este simple mecanismo, que no tienen actuación ni control, tengan una gran semejanza al de los humanos. Esto sugiere la idea de que los humanos *aprovechamos las dinámicas propias* de nuestro cuerpo para optimizar el consumo energético de la locomoción.

Este descubrimiento abrió una nueva puerta de investigación: robots que aprovechen los mismos conceptos de la locomoción pasiva aplicándolos para conseguir locomoción sobre llanos y subidas, lógicamente añadiendo elementos actuadores y de control. Esto se conoce como *dynamic walking* (caminata dinámica).

2.2.2. *Dynamic Walking*

Steve Collins y sus compañeros hicieron interesantes estudios de robots humanoides que usan los conceptos de la locomoción pasiva [6]. Desarrollaron tres robots diferentes, sustituyendo la "propulsión" gravitacional por actuadores simples.

El bípedo "Cornell" (Figura 2.5a) está basado en el mecanismo pasivo, y propulsado por motores eléctricos y muelles que ayudan al movimiento de los tobillos cuando se levantan los pies. Tiene cinco grados de libertad (dos tobillos, dos rodillas y la cadera), los brazos están unidos mecánicamente a la pierna opuesta. El bípedo "Delft" (Figura 2.5b) tiene una morfología similar, pero con actuación neumática. Y el último (Figura 2.5c) fue desarrollado en el MIT *(Massachusetts Institute of Technology)*. Tiene la peculiaridad de estar controlado mediante un aprendizaje con refuerzo *(reinforcement learning)*, que converge automáticamente a la estrategia de control óptima. Se pueden ver en funcionamiento en los vídeos [2.2.2 Bípedo Cornell], [2.2.2 Bípedo Delft], [2.2.2 Bípedo del MIT].

(a) *Bípedo "Cornell"* **(b)** *Bípedo "Delft"* **(c)** *Bípedo del MIT*

Figura 2.5: *Tres caminadores actuados basados en el mecanismo de locomoción pasivo*

El bípedo "Cornell" está específicamente diseñado para minimizar la energía de propulsión. Mediante una política en los actuadores adecuada, se consigue que siempre aporten trabajo positivo y no se usen como disipadores. Este hecho está en contraposición con el problema que presentan muchos otros robots: en el momento en el que el pie impacta en el suelo se disipa energía debido a que el robot está frenando, afectando así al rendimiento, ya que luego hay que recuperarla. Aunque los robots del MIT i "Delft" no fueron diseñados específicamente para usar poca energía, ambos tienen las ventajas inherentes de los caminadores pasivos en lo que respecta a bajo consumo.

Resulta interesante comparar la eficiencia de estos robots con otros y con los humanos. Para compararlos se usa un concepto de coste energético mecánico específico adimensional c_{mt}, propuesto por [6]:

$$c_{mt} = \frac{P_m[\mathrm{W}]}{g[\mathrm{N/kg}] \cdot v[\mathrm{m/s}] \cdot m[\mathrm{kg}]} \tag{2.6}$$

siendo P_m la potencia de locomoción consumida en *watts* (sólo la de los actuadores que se invierte en la locomoción, no se cuenta la del procesador ni otros elementos como sensores, etc.), g la gravedad, v la velocidad en m/s y m la masa en kg. La Tabla 2.1 muestra una estimación comparativa de estos parámetros. Es notorio que el robot ASIMO, un representante de los robots basados en control de los ángulos articulares, tiene un consumo específico *más de un orden de magnitud mayor* que los humanos, que a su vez está en el mismo orden de magnitud que los robots mencionados en este apartado.

Robot	*Bípedo "Cornell"*	*Bípedo "Delft"*	*Bípedo MIT*	*ASIMO*	*Humano*
$\mathbf{c_{mt}}$	0,055	0,08	0,02	1,5	0,05

Tabla 2.1: *Coste energético específico de la locomoción*

En contraste con otros robots actuados, los robots bípedos "Delft" y "Cornell" usan esquemas de control primitivos. Los únicos sensores que tienen son de contacto con el suelo, y los comandos a los motores son señales de on/off.

Algunas características interesantes (e imprescindibles) de los robots basados en el caminador pasivo son:

- La forma de la "suela" de los pies está muy estudiada. Es una forma curvada que permite que el movimiento de avance sea suave. Gran parte del movimiento talón–punta del pie tan "humano" que tienen es gracias a esta característica. El robot del MIT también tiene forma curvada en el plano lateral, facilitando su oscilación y asegurando estabilidad intrínseca en este plano.

- Presentan elementos pasivos, por ejemplo muelles. Estos elementos con las características adecuadas permiten almacenar energía en una fase de la locomoción para liberarla en otra fase. De esta forma se consigue un ahorro energético muy importante, además de un mejor comportamiento especialmente en el impacto del pie en el suelo. También pueden incluir amortiguadores mecánicos que proporcionan estabilidad en el momento de impacto con el suelo.

El resultado de estos estudios de robots bípedos basados en caminadores pasivos se puede resumir en los siguientes puntos:

- Controladores simples.

- No usan actuadores de alta potencia.

- No usan controladores de alta frecuencia, que no serían comparables al comportamiento humano.

- Tienen eficiencias y movimientos parecidos a los humanos.

2.3. BigDog

BigDog[22] es un robot con cuatro piernas, y una morfología parecida a la de un perro (de ahí su nombre *perro grande*), capaz de moverse en entornos reales muy diversos. Ha sido desarrollado por *Boston Dynamics* (www.bostondynamics.com), una ingeniería especializada en la construcción de robots dinámicos y *software* de simulación. Surgió del *Massachusetts Institute of Technology*, y en primer lugar se centraron en desarrollar un robot con forma animal que pudiera caminar.

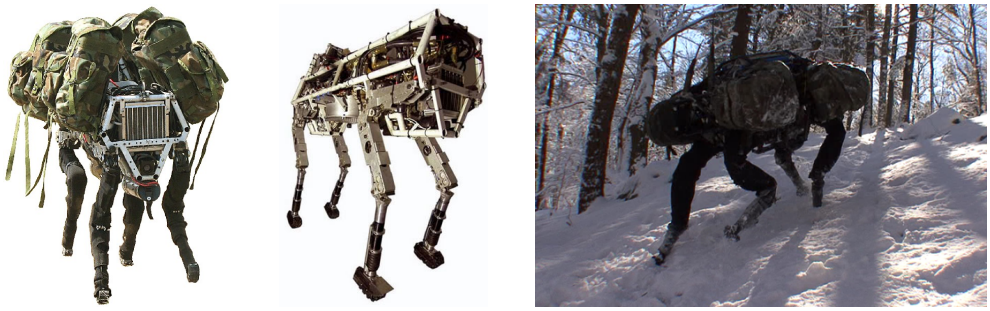


Figura 2.6: *BigDog*

Aunque este robot no es humanoide, ya que tiene cuatro piernas, su comportamiento es muy natural, siendo muy estable ante perturbaciones externas importantes y variaciones de terreno. En el vídeo [2.3 BigDog] se puede comprobar la bondad del algoritmo de control de este robot.

El robot pesa 109 kg, mide alrededor de un metro de alto, 1,1m de largo y 0,3m de ancho.

2.3.1. Descripción y morfología

BigDog tiene varios sistemas a bordo, que le suministran potencia, actuación, sensado, control y comunicaciones. Utiliza como fuente de energía un motor de combustión interna de 2 tiempos de 15 CV que va conectado a una bomba hidráulica, la cual proporciona aceite a alta presión a través de los sistemas correspondientes.

Los actuadores son cilindros hidráulicos de baja fricción controlados por servoválvulas de alta calidad. Cada actuador tiene sensores de posición y fuerza. Cada una de las cuatro piernas dispone de cuatro actuadores hidráulicos, y además un quinto grado de libertad pasivo (ver Figura 2.7). La computadora controla el comportamiento del BigDog, sus sensores y las comunicaciones.

BigDog tiene unos 50 sensores. Los de inercia miden la inclinación y aceleración del cuerpo, mientras que los de las articulaciones miden el movimiento y fuerza de los actuadores. Con toda esta información es posible tener una estimación de cómo se está moviendo en el espacio.

El sistema de control realiza tareas tanto de bajo como de alto nivel: control de fuerzas y posiciones de las articulaciones y coordinación global de la locomoción. Permite controlar la interacción con el suelo para mantenerse erguido y continuar la marcha.

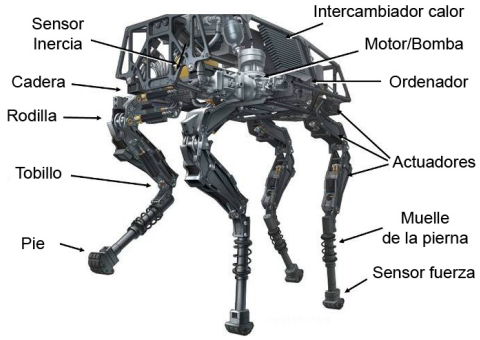


Figura 2.7: *Descripción BigDog*

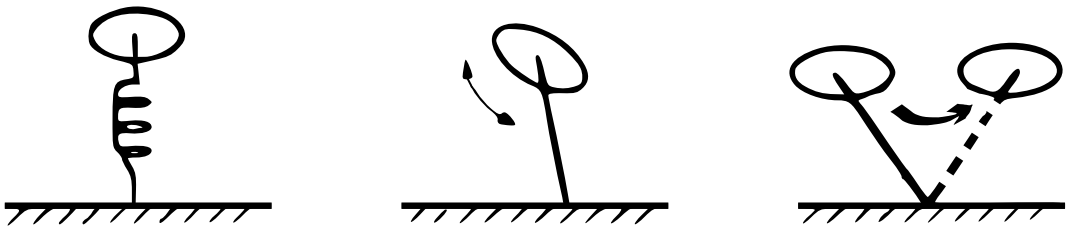
El robot tiene una gran variedad de estilos de locomoción. Puede permanecer en pie, avanzar levantando sólo una pierna cada vez (velocidad $0{,}2m/s$), caminar levantando 2 piernas opuestas por vez (igual que un perro, $1,6m/s$), caminar corriendo con fase aérea (velocidad normal $2m/s$, máxima $3,1m/s$).

2.3.2. Control

BigDog camina al trote mientras asegura su estabilidad dinámicamente. El sistema de control es simple comparado con otros. Usa una estimación de la velocidad y aceleración lateral para estabilizarlo, prediciendo dónde colocar el siguiente paso de tal forma que se estabiliza el movimiento.

Se basa en los siguientes principios:

- Soportar el cuerpo con un movimiento vertical basado en rebotes (ver Figura 2.8a).
- Controlar la forma de caminar a través de los momentos aplicados en las caderas durante la fase de soporte de la pierna (ver Figura 2.8b).
- Colocar los pies en posiciones clave a cada paso usando principios de simetría para que el robot se balancee de forma estable mientras camina (ver Figura 2.8c).



(a) *Rebotar sobre las piernas con muelle*

(b) *Mantener postura vertical con las piernas soporte*

(c) *Movimientos simétricos*

Figura 2.8: *Principios del control de la locomoción*

Adicionalmente a los principios expuestos, el sistema de control realiza las siguientes tareas (en la Figura 2.9 se puede ver el diagrama de control):

- Estimar la evolución del terreno usando la información histórica.
- Ajustar la postura para optimizar la fuerza de cada pierna.
- Usar el control de tracción para detectar, evitar y recuperarse de resbalones.
- Evitar colisiones entre piernas.

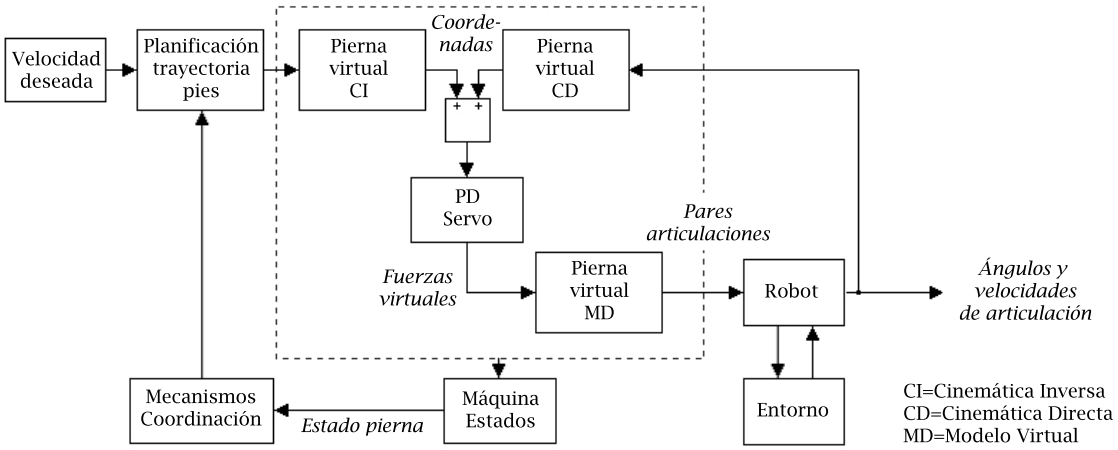


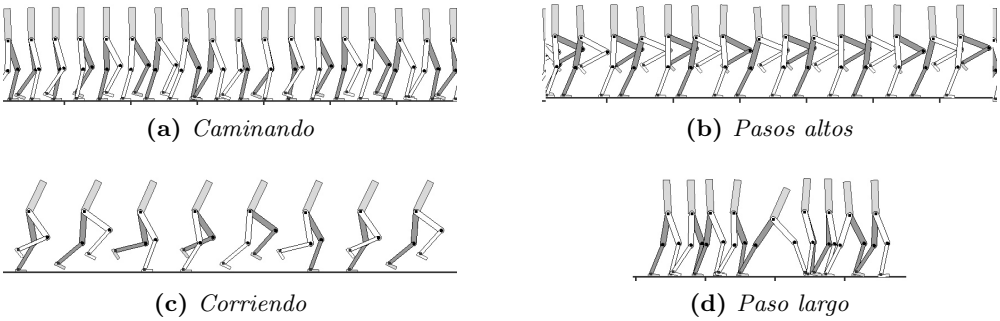
Figura 2.9: *Diagrama del lazo de control*

El algoritmo de coordinación entre piernas inicia las transiciones de estado para producir una locomoción estable. A través de cinemática inversa, y comparándola con la real se obtienen las fuerzas que se deben ejercer. Éstas se envían a articulaciones para que su interacción con el suelo sea tal que las fuerzas y momentos de la reacción estabilicen el robot mientras camina.

2.4. *SIMBICON: Simple Biped Locomotion Control*

SIMBICON [27] es un algoritmo de control para robots bípedos que usa técnicas de *feedback* orientadas a conseguir el equilibrio a partir del estudio de las dinámicas involucradas. Permite una gran cantidad de estilos de locomoción en tiempo real (hacia delante, hacia atrás, de lado, girando, corriendo, saltando, gateando...

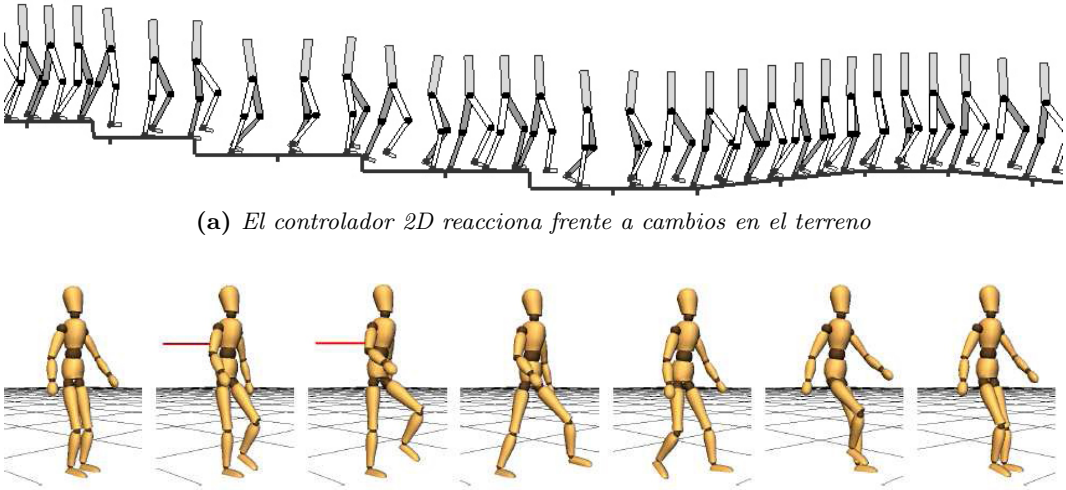
Este algoritmo funciona bien tanto en 2D como en 3D, y con parámetros adecuados se puede conseguir un estilo de caminar muy parecido al humano y muy estable frente a perturbaciones externas o cambios de terreno. Cabe destacar que sólo se ha probado con simulaciones físicas (con parámetros reales), pero no en robots bípedos. Se puede ver una demostración en el vídeo [2.4 SIMBICON].



(a) *Caminando* **(b)** *Pasos altos*

(c) *Corriendo* **(d)** *Paso largo*

Figura 2.10: *Simulaciones de un personaje en 2D con el algoritmo* SIMBICON



(a) *El controlador 2D reacciona frente a cambios en el terreno*

(b) *El controlador 3D reacciona a una fuerza externa en diagonal*

Figura 2.11: *Simulación en tiempo real de un personaje que camina con el algoritmo* SIMBICON

2.4.1. Algoritmo

La idea del algoritmo es usar una máquina de estados finita, o grafo con posturas de control. Cada estado consiste en una postura del cuerpo que representa los ángulos objetivo de cada articulación respecto al eslabón anterior, para todas las articulaciones. Cada articulación individual usa un controlador PD (proporcional-derivativo) para dirigirse a su ángulo objetivo. Las transiciones entre estados ocurren después de un tiempo fijo o, para otros estados, después de que el pie establezca contacto con el suelo. Los estilos de locomoción para caminar se modelan con cuatro estados, y para correr sólo dos estados.

El control por posturas por sí solo no tiene equilibrio intrínseco, sin embargo, unas modificaciones permiten obtener una locomoción robusta. En primer lugar, el torso y la articulación cadera-fémur de la pierna oscilante tienen ángulos objetivo respecto al *sistema de referencia del suelo*, no respecto a su articulación previa. En segundo lugar, se añade un término de *feedback* para corregir el ángulo objetivo de la cadera-fémur en función de la posición del centro de masas y su velocidad. De esta forma se consigue un balanceo robusto modificando el siguiente punto de soporte.

Máquina de estados

Cada estado tiene sus ángulos objetivo. Para una locomoción simétrica, los pares de estados derecho-izquierdo deben ser simétricos, es decir, los estados 0 y 2, 1 y 3. La transición entre los estados 0 → 1 ocurre después de un tiempo, y de 1 → 2 después del contacto del pie con el suelo (ver Figura 2.12).

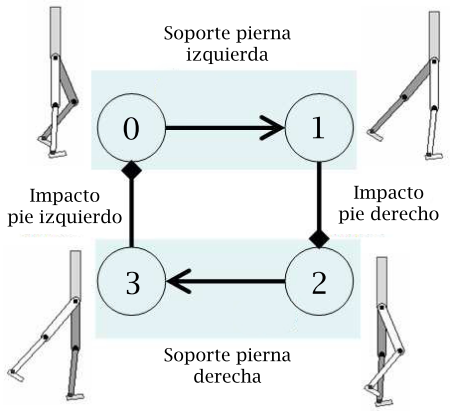


Figura 2.12: *Máquina de estados*

En cualquier estado dado, los pares aplicados a las articulaciones se calculan con un control proporcional-derivativo (PD) $\tau = k_p(\theta_d - \theta) - k_d\dot{\theta}$ para llevar cada articulación al ángulo deseado θ_d. Estos ángulos son un *objetivo*, no se alcanzan realmente en un funcionamiento normal.

Control de la inclinación del torso y la cadera en movimiento

La cadera de la pierna de apoyo y la pierna oscilante se controlan de forma diferente, tal como se ilustra en la Figura 2.13a. En primer lugar se controla la inclinación del torso respecto al sistema de referencia del suelo, esto se consige usando un controlador virtual PD que trabaja en este sistema de referencia que calcula el par neto τ_A, tal como se muestra en la figura. En segundo lugar se desacopla el control del ángulo de la cadera de la pierna oscilante respecto a la inclinación actual del torso, controlando la cadera de la pierna en movimiento respecto al sistema de referencia del suelo. El par τ_B es calculado también con un controlador PD virtual que trabaja en la referencia del suelo. Finalmente cabe notar que el par τ_B es interno al sistema y, de hecho, la reacción (par inverso) se está aplicando sobre el torso. Es necesario que el par deseado sobre el torso τ_{torso} sea el par neto que se aplica, $-\tau_A - \tau_B$. Para solucionar este aspecto se calcula el par neto del torso como $\tau_A = \tau_{torso} - \tau_B$.

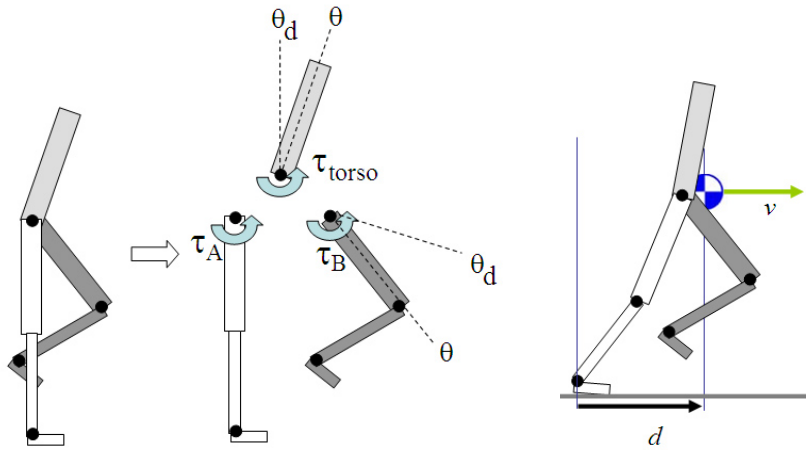
Realimentación para conseguir equilibrio

El último componente de la estrategia de control es aplicar una realimentación al posicionamiento del pie en movimiento. Se emplea una ley de la forma

$$\theta_d = \theta_{d0} + c_d d + c_v v \tag{2.7}$$

a la cadera de la pierna en movimiento, siendo θ_d el ángulo objetivo del controlador PD, θ_{d0} el ángulo objetivo por defecto de la máquina de estados, d la distancia horizontal del pie de soporte hasta el centro de masa (CDM), y v la velocidad del CDM, tal como se ve en la Figura 2.13b. El punto medio de las caderas se puede usar como una solución simple y efectiva para estimar la posición y velocidad del CDM.

El parámetro c_d es importante para suministrar equilibrio en locomociones a baja velocidad o simplemente dando pasos en el sitio. El parámetro c_v regula la velocidad en estado estacionario.



(a) *Relación entre los pares del torso, cadera de pierna de apoyo y cadera de la pierna en movimiento*

(b) *Posición y velocidad del centro de masas*

Figura 2.13: *Elementos de la estrategia de realimentación para el equilibrio*

2.4.2. Resultados

El algoritmo *SIMBICON* proporciona una serie de ventajas: gran facilidad para crear estilos de locomoción, resulta muy estable frente a perturbaciones, y su simplicidad y facilidad de programación. Puede ser una opción muy interesante para el desarrollo de la locomoción de un robot humanoide. En la Figura 2.10 se muestran los dibujos de diversas simulaciones.

2.5. Conclusiones

Después de este análisis se puede ver el estado del arte en lo que respecta a la locomoción bípeda de robots humanoides. Los resultados más exitosos se han conseguido con el algoritmo ZMP, un procedimiento sistemático que permite asegurar la estabilidad de las locomociones. Este algoritmo es siempre una referencia para el desarrollo de otros.

También existen otras múltiples aproximaciones, como *dynamic walking*. Esta última está bastante desarrollada, y permite a través de un diseño conjunto de la estructura física

y el control reducir muchísimo el consumo energético de la locomoción. Una aproximación inspirada en este planteamiento puede proporcionar propiedades muy interesantes al robot.

Aparte de los algoritmos usados, existen muchos otros, como por ejemplo el uso de modelos virtuales [21], aprendizaje con redes neuronales [14], aprendizaje por refuerzo [19] [12], técnicas de optimización [25]... Aquí se han mostrado algunos de los algoritmos más usados para locomoción bípeda, y otros interesantes para tener una base y fuente de inspiración para la implementación particular que se desarrollará en este proyecto. Si embargo, falta analizar un aspecto muy importante: la locomoción bípeda humana. Este punto se tratará en el siguiente capítulo.

Capítulo 3

Análisis de la locomoción bípeda humana

La locomoción humana es compleja y un análisis exhaustivo sería inviable para este proyecto. A continuación se analizarán aquellos aspectos biológicos-cinemáticos-mecánicos relevantes para el diseño del algoritmo de locomoción del robot. Este análisis se centrará en las piernas, ya que son la clave para caminar.

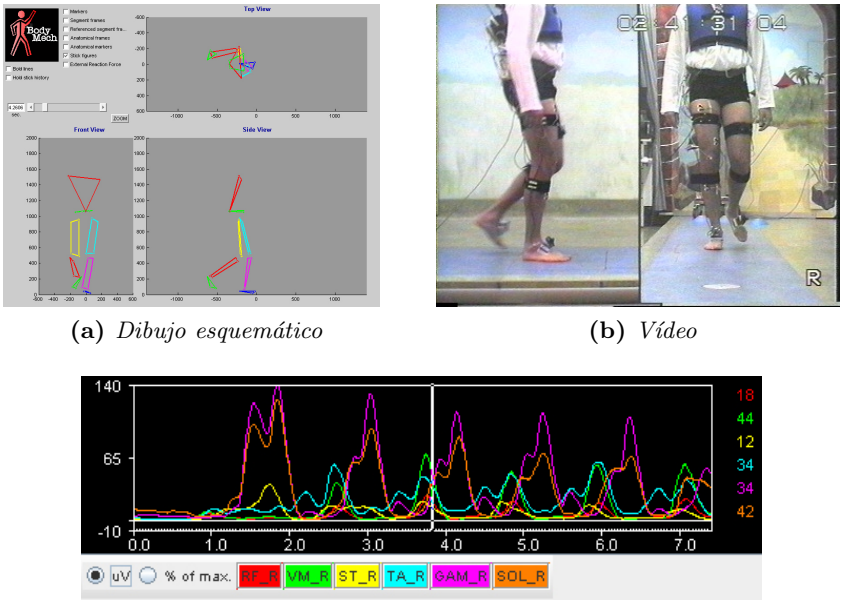
Se puede afirmar que la forma de caminar de los humanos está muy optimizada para la configuración de dos piernas. Ya que se pretende implementar el control en un robot *humanoide*, también es un deber estudiar el modelo "original".

Se ha utilizado como fuente de datos la información proporcionada en [13], y también por la página web www.dynamicwalking.org. En ella se puede hallar una captura de información con múltiples sensores y cámaras de precisión para analizar el caminar humano. Parte de los datos expuestos se han tomado de este estudio, y se mostrarán de una forma gráfica para su mejor comprensión.

Los datos que proporcionan constan de (ver Figura 3.1):

- Diferentes estilos de caminar: Normal, lento, muy lento, muy muy lento, rápido y estilo ASIMO.
- Captura en vídeo de la secuencia.
- Información de la actividad muscular.
- Captura de la posición XYZ de cada articulación.
- Ángulos, momentos y potencia de cada articulación, obtenidos por cinemática/dinámica inversa.
- Dibujo esquemático de la persona.

El aspecto más relevante para el posterior desarrollo de un algoritmo de locomoción en el robot es la cinemática. En concreto se analizarán el estilo general de locomoción y los movimientos articulares. Otros aspectos como la dinámica, potencia u otros aspectos salen de los objetivos de este proyecto. Hay múltiples e interesantes estudios al respecto que se pueden consultar para más información en la web de www.dynamicwalking.org.



(a) *Dibujo esquemático* **(b)** *Vídeo*

(c) *Actividad muscular*

Figura 3.1: *Análisis del caminar humano*

3.1. Descripción cualitativa

3.1.1. Fases de la marcha

La marcha normal, un acto esencial que se aprende por instinto, es de una complejidad extraordinaria. Los movimientos son muy rápidos, ya que un ciclo completo dura del orden de 1s. Se caracteriza por el hecho de que el cuerpo jamás abandona totalmente el contacto con el suelo, contrariamente a lo que ocurre en la carrera. Para mayor claridad se explican las fases de la marcha referenciadas al porcentaje del ciclo de un solo miembro, desde el contacto del talón en el suelo hasta el siguiente contacto.

De forma esquemática los diversos tiempos de la marcha son:

1. *Ataque del talón en el suelo (0-15 %)* (Figura 3.2a) – La rodilla está en extensión completa, la pelvis oblicua hacia adelante, el tobillo en posición neutra.

2. *Pie plano en el suelo (15-40 %)* (Figura 3.2b) – El sujeto se halla en equilibrio sobre un único pie. La rodilla está flexionada de un 15 a un 25 % con el fin de evitar un ascenso del centro de gravedad.

3. *Despuegue del talón (40-50 %)* (Figura 3.2c) – Apoyo reducido sólo al antepié. Este elemento es estable a partir del cual los demás se mueven.

4. *Despegue de los dedos (50-60 %)* (Figura 3.2d) – Es el doble apoyo.

5. *Avance del miembro inferior oscilante (60-75 %)* (Figura 3.2e) – Flexión de la rodilla rápida e importante (40-50º), flexión del tobillo, el miembro posee su longitud mínima.

6. *Extensión total (75-100 %)* (Figura 3.2f) - El miembro oscilante pasa a gran velocidad y se coloca en la posición de mayor longitud posible con el fin de alcanzar el suelo lo más delante posible del cuerpo.

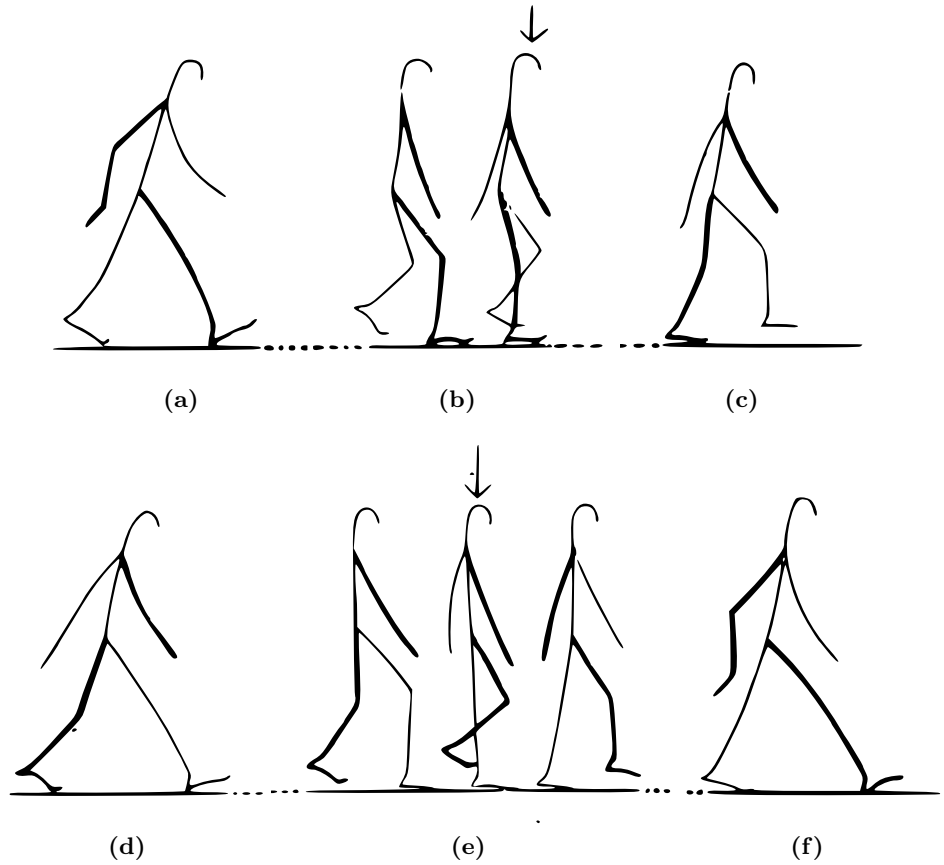


Figura 3.2: *Fases de la marcha*

Es de notorio interés que en la marcha normal sólo se tiene un pie totalmente apoyado durante un 50 % del tiempo. Es decir, que la condición ZMP que el pie completo esté totalmente apoyado sólo se cumple durante medio ciclo. Se puede comprobar fácilmente intentando caminar con los pies siempre planos que se requiere mucho más esfuerzo y es más dificultoso.

3.1.2. Desarrollo del pie durante la marcha

1. De 0 al 15 % (Figura 3.3a) – El abordaje al suelo se realiza por el talón.
2. Del 15 al 40 % (Figura 3.3b) – El pie permanece plano. El contacto se realiza a través del borde externo.
3. Del 40 al 50 % (Figura 3.3c) – Despegue del talón. Todo el peso reposa sólo sobre el antepié.
4. Del 50 al 60 % (Figura 3.3d) – Despegue de los dedos.

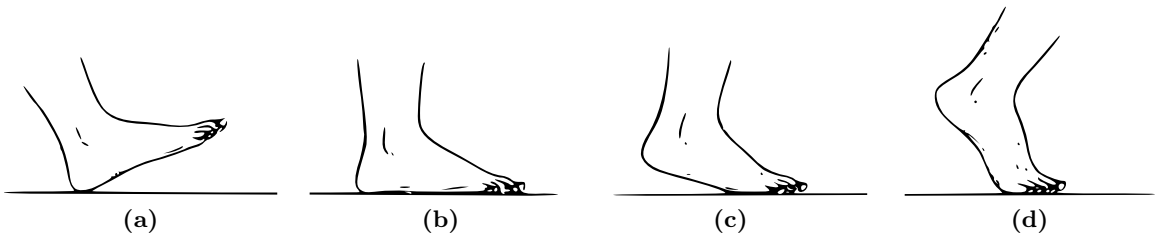


Figura 3.3: *Desarrollo del pie durante a marcha*

Este sería el esquema del desarrollo del paso para un pie estrictamente normal. En caso que se amputaran todos los dedos la última fase del paso no se puede realizar, y el paso se debe acortar.

Se puede ver que desde el punto de vista de la pierna, el pie está basculando. Esta idea es la que se aprovechaba en los caminadores pasivos, por eso tienen movimientos tan parecidos a los humanos.

3.1.3. Características del paso

El *paso* es la distancia que separa dos apoyos sucesivos, y se mide de talón a talón. Se llama *ángulo del paso* el formado por la línea de marcha y el eje del pie, que cruzan por detrás. Su valor normal es de 15º. La *amplitud del paso* es la distancia que separa el talón de la línea de marcha, para una velocidad media es de 5 a 6cm (ver Figura 3.4).

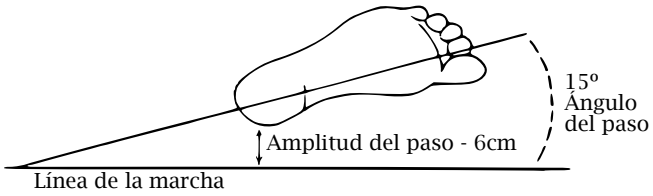


Figura 3.4: *Características del paso*

La *cadencia* es el número de pasos hechos en un minuto. La *velocidad de marcha* es igual al producto de la cadencia por la longitud de paso. En un hombre adulto de talla media (170cm de altura) la cadencia más eficiente[1] varía entre 110 y 130 pasos por minuto. Esto es un período de ciclo[2] entre 0,92 y 1,09 s. La longitud de paso oscila entre 75 y 85 cm, y la velocidad será de 5 a 6,5 km/h.

La marcha normal es la que tiene un consumo energético menor, con una velocidad alrededor de 4,5 a 5 km/h.

3.1.4. Brazos y movimiento de torsión transversal

Con el fin de separar lo menos posible el centro de gravedad de su eje de progresión, y disminuir la oscilación transversal del tronco, es preciso que los hombros y brazos realicen el giro en sentido inverso (ver Figura 3.5).

[1] Entendida como la que tiene la relación velocidad/(consumo energético) mayor.
[2] Recordar que en un ciclo completo se realizan *dos* pasos.

Se puede comprobar que si se intenta caminar con los brazos fijos pegados al tronco resulta mucho más dificultoso, el torso oscila mucho en el eje transversal, aunque se puede proseguir con la marcha.

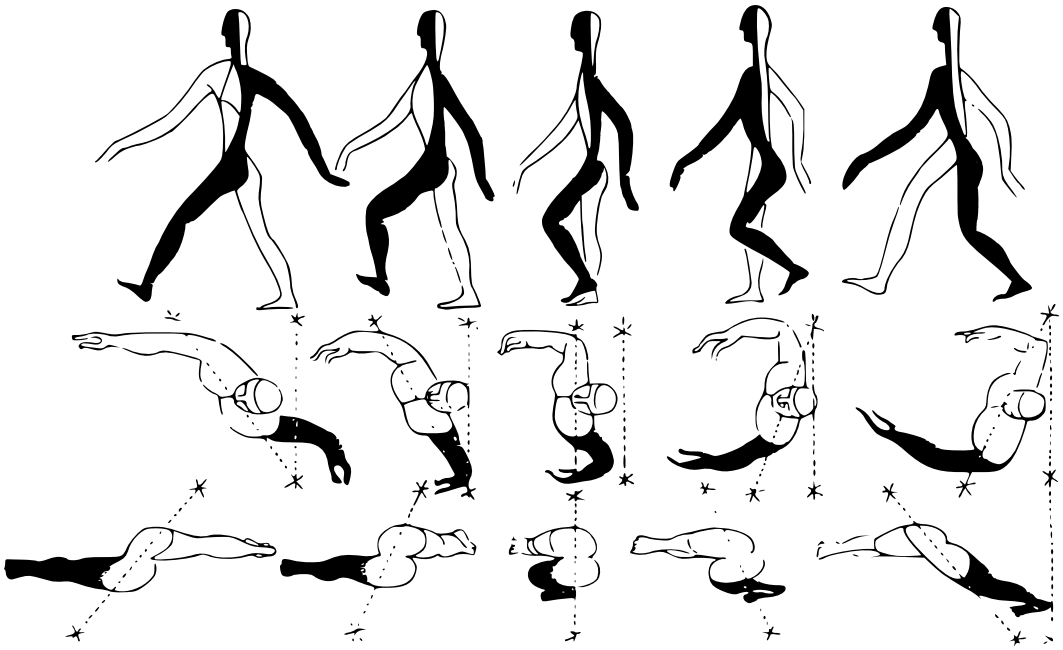


Figura 3.5: *Movimiento de brazos y hombros al caminar*

3.2. Cinemática

3.2.1. Sistema de referencia

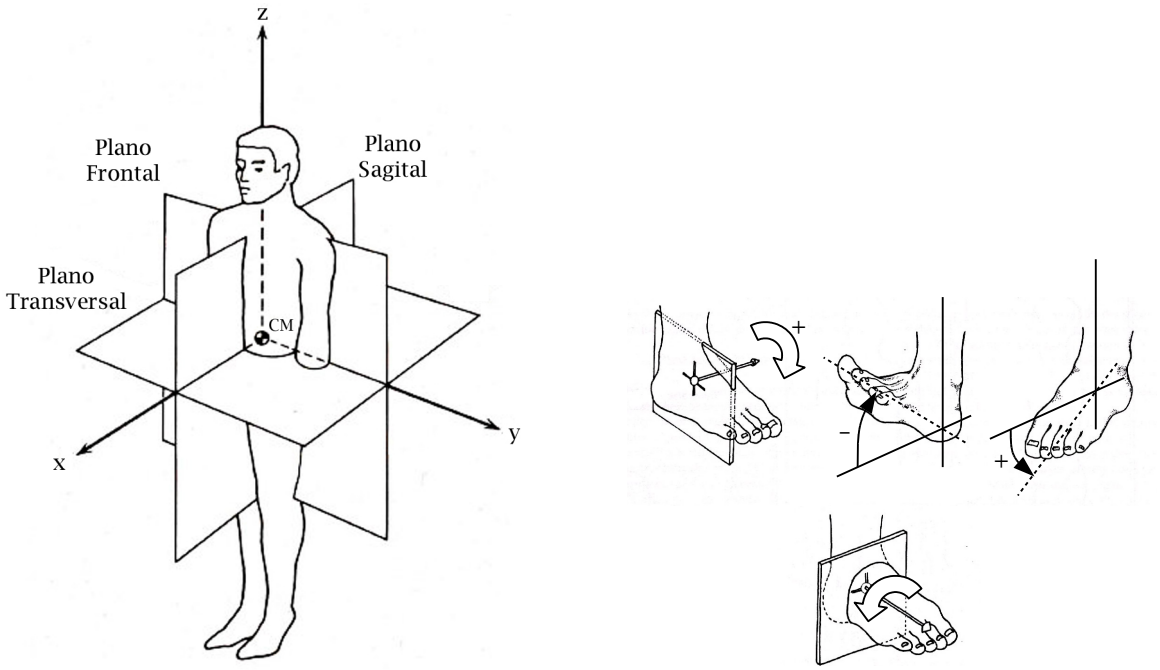
En primer lugar se definirán planos de referencia y ejes respecto al cuerpo (Figura 3.6). Se considerarán los movimientos en cada plano independientes para esta primera aproximación. Los ejes de rotación locales de cada articulación están definidos con el mismo sentido que estos ejes locales para la posición indicada en el dibujo. De hecho, por coherencia se han tomado los mismos ejes que se definirán en el robot Nao.

Ya que se analizará la locomoción en línea recta, se estudiarán solamente los planos frontal (movimientos de aducción/abducción), y sagital (movimientos de flexión/extensión). Los movimientos en el plano transversal (movimientos de torsión) son menos relevantes en línea recta a bajas velocidades.

3.2.2. Movimientos articulares

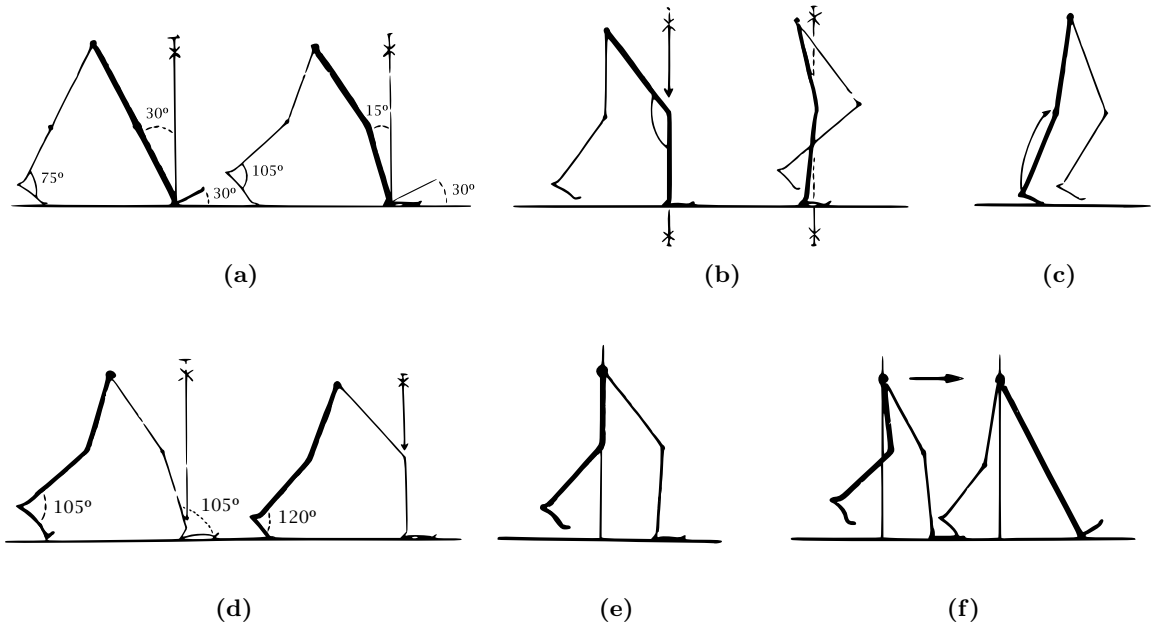
Los comandos que se enviarán al robot para controlarlo serán las configuraciones angulares de cada articulación. Es por eso que resulta interesante analizar la evolución temporal de los ángulos de articulación cuándo una persona está caminando.

En la Figura 3.7 se pueden hallar instantáneas para cada fase del paso.



(a) *Planos referencia* **(b)** *Ejemplo articulación tibia-pie*

Figura 3.6: *Sistema de referencia*



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figura 3.7: *Instantáneas de las fases de la marcha [13]*

En la web mencionada anteriormente se encuentra información sobre la evolución temporal de las variables articulares. Todo el análisis que se realizará a continuación será con un estilo de caminar a velocidad normal ($v = 1,27m/s = 4,6km/h$), porque se considera que éste es una buena referencia.

En la Figura 3.9 se representan los ángulos articulares mientras se está caminando de forma normal, una vez ya se ha iniciado el movimiento. El sentido está definido de la articulación superior a la inferior, según los ejes indicados. Se muestra un único ciclo, de período 1,08 s. El eje de tiempo está adaptado para poder seguirlo en el vídeo [3.2.2 Persona caminando a velocidad normal].

Los movimientos de flexión/extensión son los previsibles. En los de abd/aducción se puede destacar el hecho que mientras un pie está apoyado, la configuración articular intenta ser tal que el centro de masa del cuerpo esté sobre la vertical del pie (ver Figura 3.8).

Como lección se puede extraer que los movimientos no son extremadamente complejos, como era de esperar. Esto implica que planteamientos como *SIMBICON* basados en un grafo de posturas pueden representar bien el comportamiento humano.

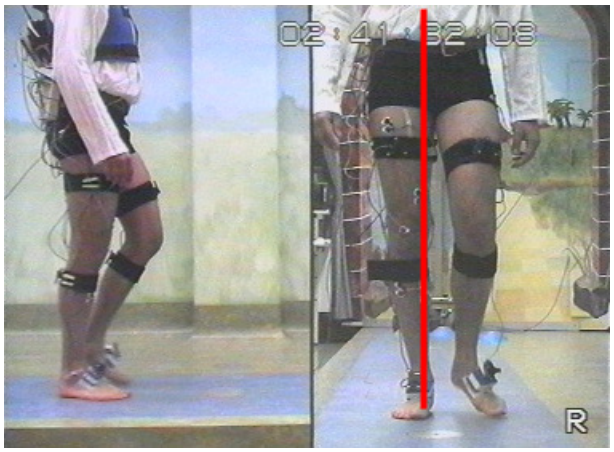
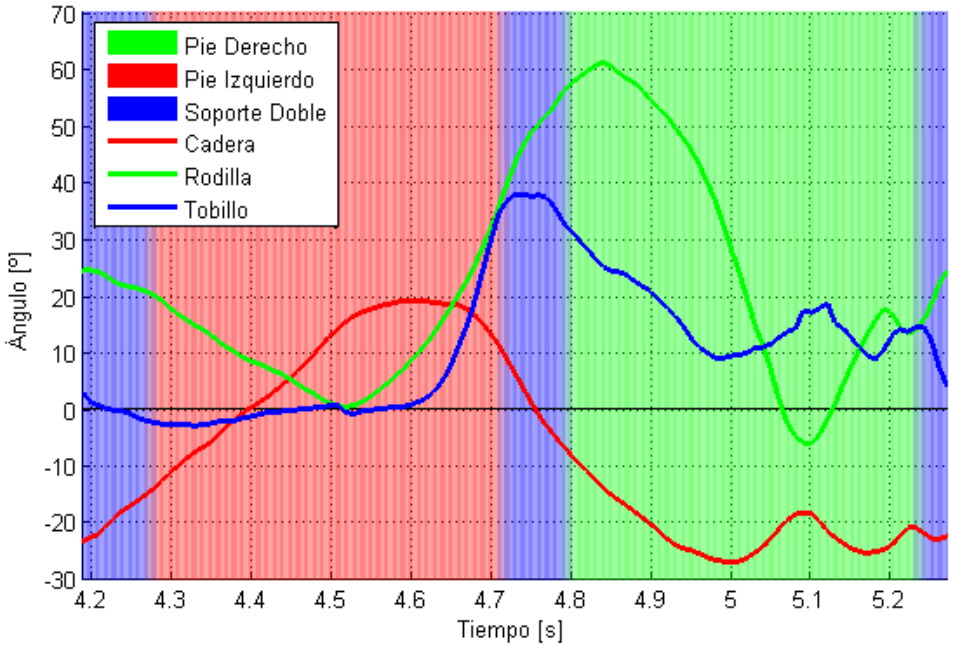


Figura 3.8: *Instantánea en* $t = 5s$

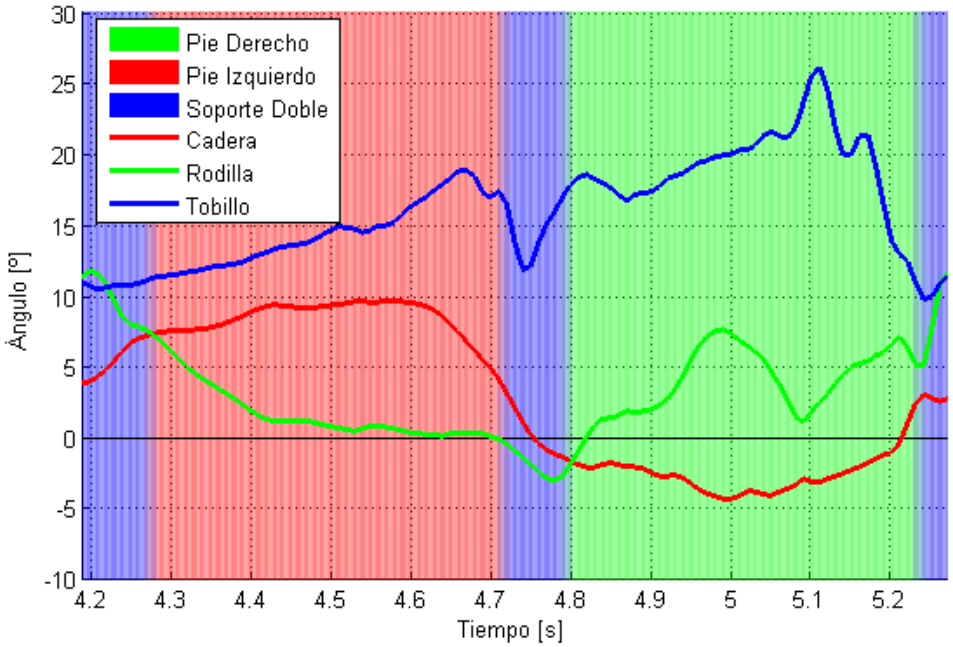
3.3. Conclusiones

El análisis hecho aquí, y especialmente la evolución temporal de los ángulos articulares para cada fase, se tomará como base para el algoritmo de locomoción de desarrollo propio. Posteriormente se hará un análisis de prestaciones para compararlo con el comportamiento humano.

Es de notar el hecho que en la locomoción humana resulta un punto clave los dedos de los pies. Ya que en el robot no se dispone de esta articulación (tal como se verá posteriormente), es de esperar que sea necesario usar una marcha de pasos cortos.



(a) *Ángulos eje Y de flexión/extensión*



(b) *Ángulos eje X de abd/aducción*

Figura 3.9: *Ángulos articulares de la pierna derecha y fase de paso*

Capítulo 4

Nao

El robot humanoide usado en este proyecto es el *Nao Academics Edition v3*, diseñado y fabricado por la compañía francesa *Aldebaran Robotics* (`www.aldebaran-robotics.com`, ver Figura 4.1). Ha sustituido a *Aibo* de *Sony* en la competición internacional de robótica *Robocup*. Nao es un robot de tecnología puntera, programable y controlable usando Linux, Windows y Mac, proporcionando una interfaz de comunicación muy flexible. Permite realizar movimientos precisos y coordinados. Además, lleva incorporadas una serie de funciones de alto nivel para facilitar su uso. Por estas características se ha escogido como plataforma para el desarrollo de este trabajo. El vídeo [4 Presentación de Nao] muestra sus grandes posibilidades.

En este capítulo se hará una descripción general del robot, a modo de presentación. En el apéndice A se amplía esta información, y se explican detalles de cómo usarlo. Los datos técnicos se han obtenido principalmente de la documentación suministrada con el robot [3]. Las explicaciones, programación y scripts son fruto de desarrollo e investigación propios.

Figura 4.1: *Nao Academics Edition*

4.1. Características generales

Nao mide $58cm$ de altura, pesa $4{,}3kg$ y la carcasa está fabricada en plástico. Usa una batería de litio de $U_n = 21,6V$, con una autonomía aproximada de 45 *min*. El consumo en

uso normal es de 30W, y en actividad 70W. Permite conexión con cargador de 24V.

Tiene un total de 25 grados de libertad (ver Figura 4.2 y Tabla 4.1):

- 2 en la cabeza.
- 5 en cada brazo+1 en cada mano.
- 1 en la pelvis.
- 5 en cada pierna.

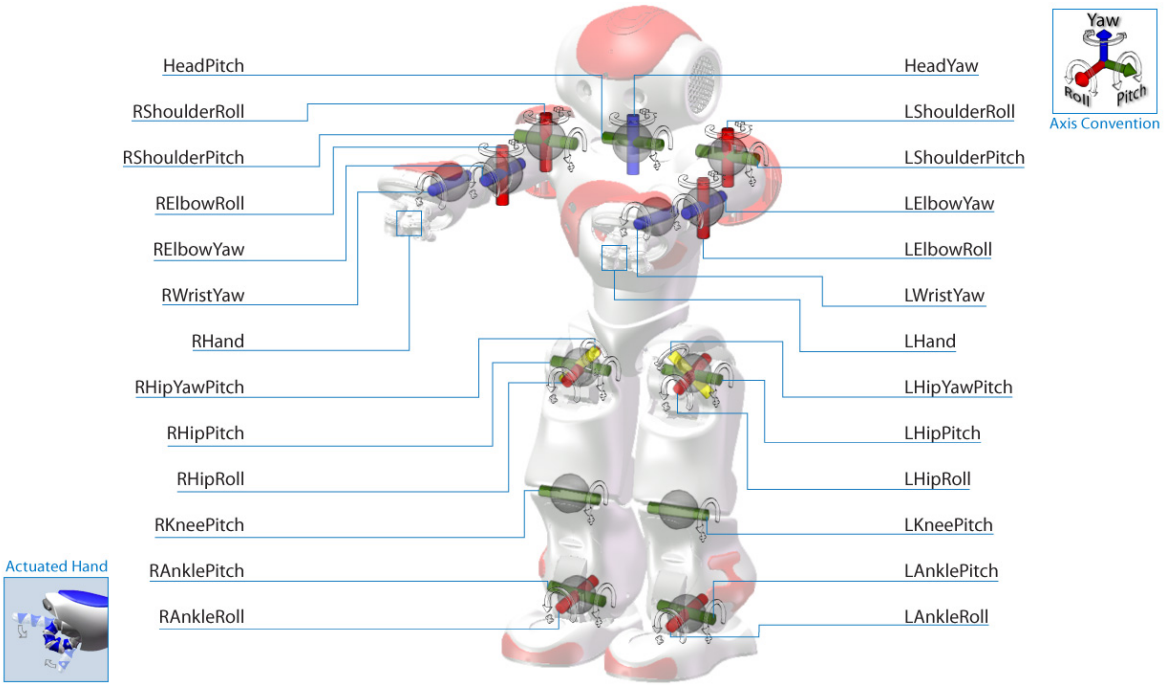


Figura 4.2: *Articulaciones del robot*

Cabeza	
HeadPith	Flexión/Extensión cuello
HeadYaw	Rotación cuello

Pelvis	
HipYawPith	Articulación pélvica a 45º

Brazos (R/L Derecha/Izquierda)	
R/LShoulderPitch	Flexión/Extensión hombro
R/LShoulderRoll	Abd/Aducción hombro
R/LElbowYaw	Rotación codo
R/LElbowRoll	Flexión/Extensión codo
R/LWristYaw	Rotación muñeca
R/LHand	Abrir/Cerrar mano

Piernas (R/L Derecha/Izquierda)	
R/LHipPith	Flexión/Extensión cadera
R/LHipRoll	Abd/Aducción cadera
R/LAnklePith	Flexión/Extensión tobillo
R/LAnkleRoll	Abd/Aducción tobillo
R/LKneePith	Flexión/Extensión rodilla

Tabla 4.1: *Lista de articulacines de Nao*

También incorpora 2 altavoces de 36*mm* de diámetro situados en las orejas, 4 micrófonos y 2 cámaras VGA 640x480, 30 fps. Respecto a los sensores, tiene 4 de ultrasonidos situados en el pecho, 4 sensores de fuerza en cada pie, sensores de tacto en la parte frontal de cada pie, sensores inerciales (acelerómetro de 3 ejes y giróscopo de 2 ejes), y en cada articulación sensores de posición con una resolución de 0,1º. Para hacerlo más vistoso, incorpora LEDs de colores en los ojos, orejas, pecho y pies. La conexión puede hacerse vía Wi-Fi IEE 802.11g (inalámbrica) o Ethernet (con cable).

Los actuadores son motores *Coreless MAXON DC*. Hay de dos tipos diferentes y con dos tipos de reducciones diferentes según las necesidades de cada articulación. El control de los motores se realiza mediante microcontroladores PIC que llevan un algoritmo tipo PID, de parámetros configurables. El modo de interactuar es mediante el envío de consignas de posición.

Nao lleva su propio PC incrustado. Tiene una CPU x86 AMD GEODE 500MHz, con 256 MB SDRAM y 1 GB de memoria flash. El sistema operativo de Nao es un Embedded Linux (32 bit x86), basado en la distribución *OpenEmbedded*. Sobre este sistema operativo corre un programa llamado *NaoQi*, que es el encargado de controlar el robot mediante diversos módulos específicos.

Para ampliar esta información se puede consultar la documentación del robot.

4.2. NaoQi: programación de Nao

Nao se suministra con un entorno de desarrollo llamado NaoQi. Éste es un programa que implementa una gran cantidad de funciones de alto nivel y se encarga de comunicarse con los dispositivos de bajo nivel. También maneja las comunicaciones. Entre algunas de las funciones de alto nivel que implementa se encuentra un algoritmo de locomoción bípeda tipo ZMP.

Para ejecutar órdenes sobre Nao es necesario hacerlo a través de NaoQi. Se puede hacer de forma remota (desde el PC) usando lenguajes de programación como C++, Python o URBI. También se pueden compilar programas para ejecutarse desde Nao, programados en C++. Esta sección se amplía en el Apéndice A.

4.3. Simulador: *Webots*

Webots es un entorno de desarrollo (ver Figura 4.3) para modelar, programar y simular robots móviles. Existen versiones para Windows y Linux. Con Webots se puede crear un mundo virtual común con varios robots que interactúen entre ellos y realizar una simulación física precisa gracias al motor matemático ODE *(Open Dynamics Engine)*. También permite grabar en AVI o MPG las simulaciones. Se puede realizar la programación con diversos lenguajes: C/C++, Java, Python, URBI, u otros. En el caso de algunos robots, el programa usado en Webots se puede trasladar directamente al robot físico. Con Nao no es posible, a no ser que se programe a través de NaoQi.

Webots se distribuye preparado para ejecutar simulaciones con Nao, ya que es la pla-

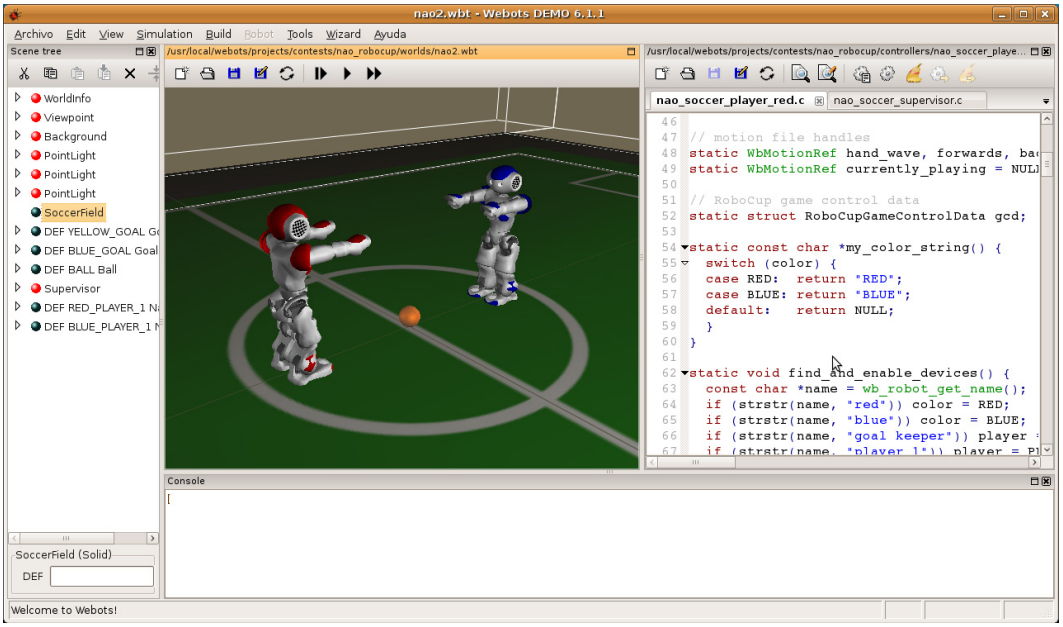


Figura 4.3: *Webots 6.1.1*

taforma estándar para la liga simulada de RoboCup (www.robocup.org). Con Nao viene un modelo para Webots que se puede controlar desde un NaoQi ejecutado en el ordenador (no con la versión demostración de Webots). También se puede programar en C o C++ un ejecutable que controle el robot durante la simulación.

Existe también otro simulador de Nao, Microsoft Robotics Studio (ver Figura 4.4). Éste se puede ejecutar únicamente bajo Windows. Igual que Webots tiene un simulador físico de un mundo virtual. Se programa en Visual Programming Language, un estilo de programación mediante bloques.

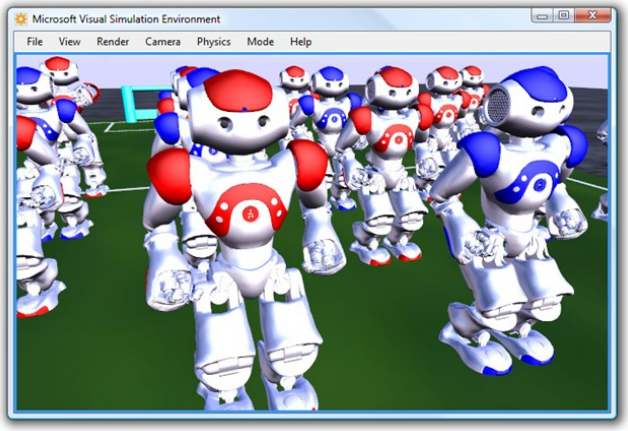


Figura 4.4: *Microsoft Robotics Studio*

Se dispone de un modelo de Nao. El problema que tiene éste es que no se puede acceder a la información de los sensores, por lo que se desestima esta vía. Además, las simulaciones

que se han probado dan resultados poco realistas en cuanto a movimientos, y especialmente el contacto con el suelo. Webots se muestra más correcto con la simulación de Nao.

4.4. Conclusiones

Nao se muestra como un robot con características muy avanzadas. Desde la estructura física, actuadores y sensores hasta la arquitectura de software se nota un diseño muy cuidado. Las funcionalidades que incorpora de control articular, su morfología humanoide bípeda y los múltiples sensores que tiene hacen que sea una plataforma idónea para el desarrollo de este proyecto, y el estudio de la locomoción bípeda en robots humanoides.

Capítulo 5

Diseño del algoritmo de locomoción para el robot Nao

Después de llegar a una clara compresión de las claves de la locomoción bípeda vistas a través de ejemplos de robots humanoides y el propio cuerpo humano, y de ver el potencial de Nao como plataforma de investigación, ha llegado el momento de desarrollar la parte clave de este proyecto: el diseño de un algoritmo propio para locomoción de Nao. Esta sección se centrará en el diseño conceptual, sin entrar en detalles de programación que oscurecerían el desarrollo lógico. La programación se mostrará en el Apéndice B.1. Finalmente se hará un análisis de prestaciones.

Conjuntamente con la investigación que se expondrá en este apartado se ha desarrollado una interfaz gráfica para poder usar todas las funciones que se mencionan, y también otras propias de Nao. Aquí mostrarán algunas capturas de pantallas, y la descripción detallada se puede encontrar en el Apéndice B.2.

Las pruebas se harán sobre el robot directamente, con las precauciones debidas. Las suelas del robot son de plástico, y sobre un suelo cerámico resbalan. Por este motivo se harán las pruebas sobre una superficie de plástico que de adherencia al robot.

Sin embargo, tal como se mencionó en la sección anterior, Nao ya tiene implementadas funciones para caminar, un algoritmo de locomoción tipo ZMP. Ya que el modelo está programado por el fabricante, se presupone la corrección de este. El desarrollo se centrará en el estudio de la funcionalidad del algoritmo ZMP implementado y cómo mejorarlo. No se hará un análisis matemático de la mecánica del robot como sistema multisólido ni la formulación del algoritmo ZMP, ya que éste ya lo ha realizado la empresa fabricante de Nao y viene dado, y además está fuera de los objetivos de este proyecto. Se analizará el algoritmo y sus prestaciones, y se le harán las modificaciones necesarias para que funcione correctamente.

Además, se investigará en el desarrollo de un nuevo algoritmo, por varios motivos. En primer lugar, aunque el algoritmo ZMP es muy potente, tiene unos costes energéticos muy elevados. Los robots basados en los *passive walkers* tienen un consumo mucho menor, ya que aprovechan las dinámicas propias del sistema. Sin embargo, estos robots se han diseñado específicamente con este fin. En la investigación bibliográfica hecha hasta la fecha, no se ha encontrado ningún robot "estándar" (es decir, no especialmente diseñado con este objetivo, que no tenga los pies con formas especiales, y con los actuadores situados en las posiciones habituales) que use los principios de los caminadores pasivos. Sería muy interesante verificar la validez de esta concepción en concreto con Nao.

5.1. *ALWalk*: Algoritmo de locomoción implementado en Nao

NaoQi, el *software* de Nao, está organizado en módulos, o bloques funcionales (información ampliada en el Apéndice A.3.1). Uno de ellos es ALMotion, que incorpora las funciones para mover el robot. La forma de moverlo es mediante consignas de posición para las articulaciones.

AlMotion permite, entre otras cosas, resolver el modelo cinemático, mover puntos determinados en el espacio cartesiano o controlar la rigidez de las articulaciones. Además, también tiene funciones de alto nivel, como por ejemplo mantener el equilibrio o caminar. El submódulo que tiene estas funciones se llama ALWalk, y se describe a continuación.

5.1.1. Descripción de las funciones de locomoción bípeda

Una vez se hace la conexión al módulo ALWalk, se tiene acceso a las siguientes funciones (se muestran en lenguaje C++):

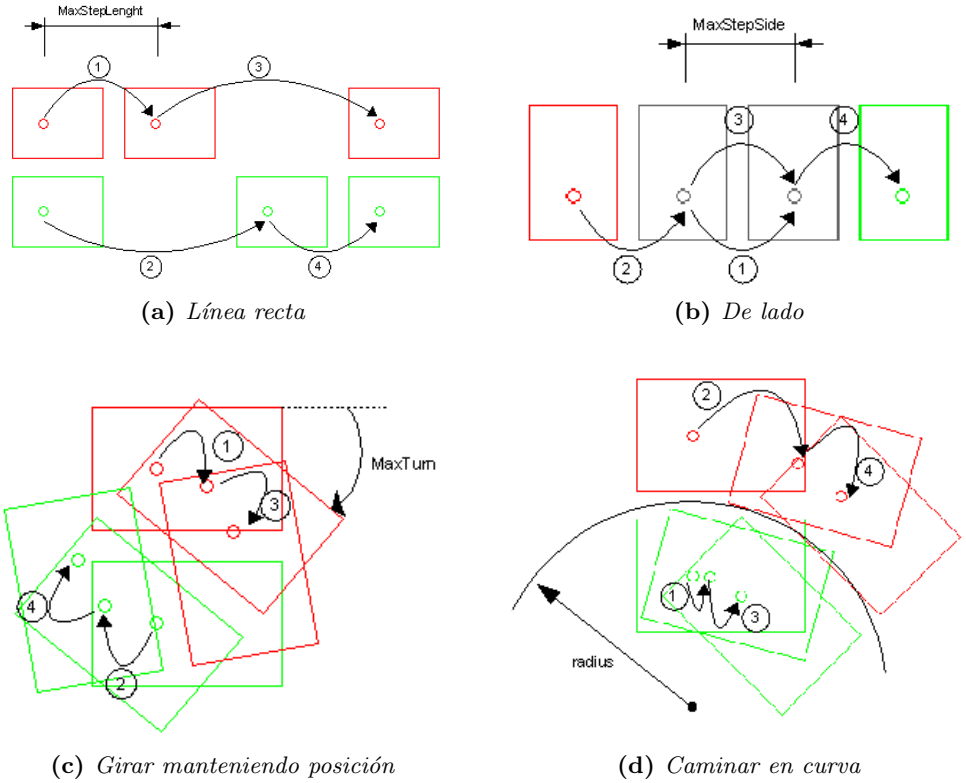
```
1  // pDistance: distancia a caminar en metros
2  // pNumSamplesPerStep: número de ciclos de 20ms por paso
3
4  //Caminar en línea recta
5  void walkStraight(float pDistance, int pNumSamplesPerStep)
6
7  //Caminar de lado
8  void walkSideways(float pDistance, int pNumSamplesPerStep)
9
10 //Girar manteniedo posición
11 void turn(float pAngle, int pNumSamplesPerStep)
12
13 //Caminar describiendo una curva
14 // pAngle: ángulo en radianes de la circumferencia
15 // pRadius: radio del círculo
16 void walkArc(float pAngle, float pRadius, int
       pNumSamplesPerStep)
```

La idea del algoritmo implementado es transformar una posición del ZMP objetivo en un objetivo de posición del centro de masa CdM, haciendo un seguimiento de este. La posición ZMP se crea *on-line* a través de patrones de posición de los pies para cada tipo de paso. Los diferentes patrones son los que se muestran en la Figura 5.1. La trayectoria de los pies se realiza con forma de cicloide (ver Figura 5.2). El cómputo de la trayectoria de CdM se realiza con un modelo simplificado de un péndulo invertido. Como es un modelo simplificado, puede dar lugar a inestabilidades.

Además, se pueden configurar parámetros característicos de la locomoción. Éstos son:

pMaxStepLength Longitud máxima de un paso en metros (de $0,0$ a $0,09m$, por defecto $0,055m$).

pMaxStepHeight Altura máxima de un paso en metros (de $0,0$ a $0,08m$, por defecto $0,015m$). Representa la altura de la cicloide.



(a) *Línea recta* **(b)** *De lado*

(c) *Girar manteniendo posición* **(d)** *Caminar en curva*

Figura 5.1: *Patrones de locomoción*

pMaxStepSide Longitud máxima de un paso lateral en metros (de $0,0$ a $0,06m$).

pMaxStepTurn Máximo cambio de orientación según eje vertical en radianes (de $0,0$ a $1,0rad$).

pZmpOffsetX Offset de la posición objetivo del ZMP hacia adelante (de $0,0$ a $0,1m$, por defecto $0,015m$). Este parámetro y el siguiente sirven para controlar como las posiciones de cada paso se convierten en una ruta objetivo del ZMP. **pZmpOffsetX** es una distancia positiva desde el talón de un paso, que hace que el CdM permanezca durante más tiempo sobre el pie.(ver Figura 5.3).

pZmpOffsetY Offset de la posición objetivo del ZMP hacia un lado (de $-0,05$ a $0,05m$, por defecto $0,015m$). Se puede usar para reducir el ancho del movimiento lateral del CdM.

Asimismo, también se definen cuatro parámetros extra para facilitar la creación de patrones de locomoción estables: **pLHipRollBacklashCompensator** (0º a 15º, por defecto 4,5º), **pRHipRollBacklashCompensator** (-15º a 15º, por defecto $-4,5$º), **pHipHeight** (0,15 a $0,244m$) y **pTorsoYOrientation** (-20º a 20º, por defecto 5º). **pLHipRollBacklashCompensator** y **pRHipRollBacklashCompensator** son dos parámetros para compensar la deformación plástica que se produce en la cadera durante la caminata. Se envía una señal trapezoidal a la articulación *HipRoll* durante la fase de soporte sobre un único pie para mantener el torso recto lateralmente. **pHipHeight** define la altura de las caderas del robot durante

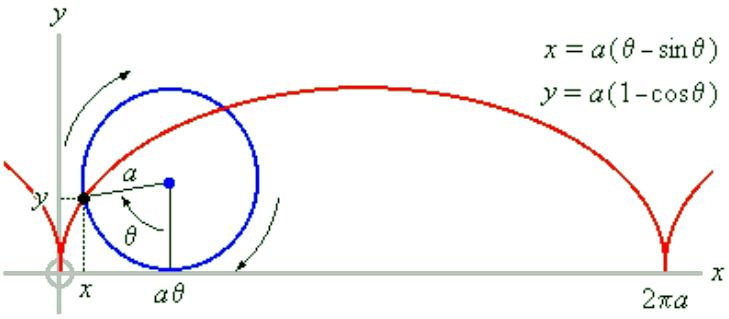


Figura 5.2: *Cicloide usada para trayectorias de los pies*

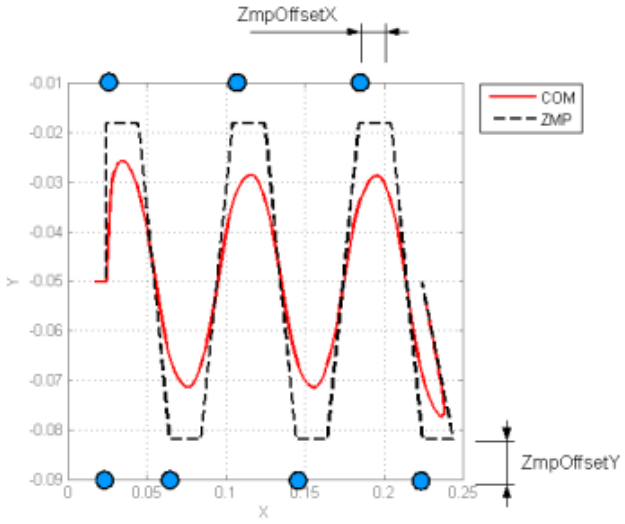


Figura 5.3: *Configuración de parámetros de Offset del ZMP*

la marcha. Cuánto más alto, más inestable será la locomoción, y más cerca se estará de la singularidad, lo que puede llevar a provocar movimientos bruscos. **pTorsoYOrientation** define la orientación del torso frontalmente.

En la documentación de Nao [3] se dan explicaciones detalladas de cómo configurar estos parámetros para estabilizar la locomoción, así como una configuración por defecto.

5.1.2. Pruebas de funcionamiento

Una vez configurado, hacer que Nao camine debería ser tan fácil como ejecutar la siguiente órden de Python:

```
motion.walk(1, 60)  #Camina 1m, con un paso de duración 60x20ms=1,2s
```

Los resultados se muestran en el vídeo [5.1.2 ALWalk velocidad baja]. También se hicieron pruebas para una velocidad de paso mayor, se muestra en el vídeo [5.1.2 ALWalk velocidad alta]. Como se puede ver, el robot cae estrepitosamente por los siguientes motivos, algunos de ellos inherentes al algoritmo ZMP:

- El sistema está en lazo abierto, por tanto es inviable compensar ningún tipo de perturbación. El robot únicamente reproduce una secuencia de movimientos, no tiene información del entorno.
- El algoritmo ZMP se basa en un seguimiento con una alta ganancia de una trayectoria precalculada. Esto impide que el sistema absorba ningún tipo de perturbación. En este caso se tiene 100 % de *stiffness*[1] para todas las articulaciones (se llamará *full-stiffness*).
- Seguramente existan diferencias entre el modelo físico usado y el robot real, por lo que las trayectorias calculadas no necesariamente sean tan robustas como se esperaría.
- Las articulaciones no son perfectas. Existen huelgos mecánicos y otros fenómenos no modelados que provocan inestabilidades.

El sistema está en lazo abierto y por ahora Nao no dispone de medios para estabilizarlo. Los errores de modelado son difíciles de compensar dada la complejidad y alta dimensión del sistema. Sin embargo sí que se puede hacer que se absorban un cierto grado las perturbaciones, el siguiente apartado explicará la solución que se ha obtenido.

5.1.3. Modificación del algoritmo ZMP

El problema principal reside en que el seguimiento de las trayectorias se realiza con ganancias muy altas, no se absorben las perturbaciones y por tanto se desequilibra. De hecho, cada articulación está configurada para suministrar el par máximo (100 %) en caso de ser necesario. Este comportamiento no se corresponde al humano, en que las articulaciones se puede considerar que no hacen un seguimiento con altas ganancias. Hay múltiples investigaciones al respecto, pero [9] es de especial interés al centrarse específicamente en la aplicación a Nao.

Una solución a esta problemática es limitar el par máximo aplicado a cada articulación. Esto se puede realizar directamente en Nao gracias al parámetro *stiffness* (rigidez) de cada articulación. La *stiffness* representa qué fracción en tanto por uno del par máximo se aplicará. Es una saturación de la salida del controlador del motor que permite un seguimiento de la trayectoria más suave, y a la vez absorber las perturbaciones y estabilizar el sistema.

Después de múltiples ensayos experimentales iterativos, se llegó a una configuración de *stiffness* que permitía estabilizar la locomoción, que se muestra en la Tabla 5.1. Respecto a los parámetros de configuración extra se establecieron los que se muestran en la Tabla 5.2. En la Figura 5.4 se muesta una captura de la interfaz que se ha creado para controlar *ALWalk* con los parámetros citados.

El procedimiento de ajuste ha sido el siguiente:

- Para disminuir la inestabilidad se ha reducido la altura de las caderas al mínimo.
- Las inestabilidades se deben a que los pies no se apoyan bien sobre el suelo, y lo desequilibran. Para solucionar este problema se bajó mucho la *stiffness* para que se pudieran absorber las perturbaciones, y así evitar un comportamiento rígido.

[1] Rigidez. Se usarán indistintamente los dos términos, ya que en la bibliografía el término inglés *stiffness* está ampliamente usado.

Articulación	*Stiffness*
Brazos	0,1
HipYawPitch	0,6
R/LHipPitch	0,7
R/LHipRoll	0,3
R/LKneePitch	0,3
R/LAnkleRoll	0,3
R/LAnklePitch	0,2

Tabla 5.1: *Configuración de Stiffness para cada articulación*

Parámetro	Valor	Parámetro	Valor
pMaxStepLenght	0,04	*pMaxStepHeight*	0,025
pMaxStepSide	0,02	*pMaxStepTurn*	0,3
pTorsoYOrientation	0,0	*pHipHeight*	0,0
pZmpOffsetX	0,0	*pZmpOffsetY*	0,0
pRHipRollBacklashCompensator	0,0	*pLHipRollBacklashCompensator*	0,0

Tabla 5.2: *Configuración de parámetros extra*

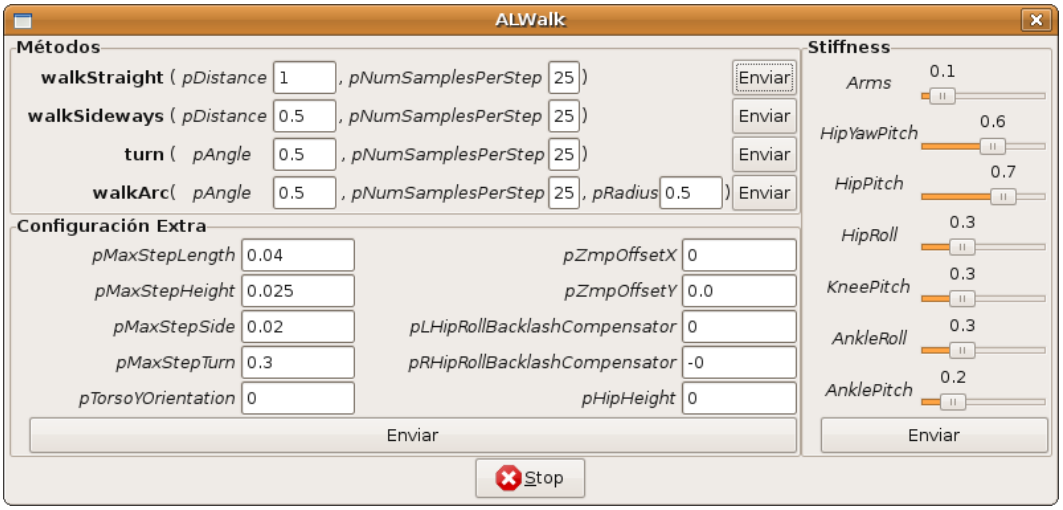


Figura 5.4: *Pantalla de la GUI para la configuración de ALWalk*

- También se bajó la *stiffness* de las rodillas y del movimiento lateral de las caderas para hacer un movimiento más suave, como si fueran "muelles".
- La *stiffness* del movimiento frontal de caderas no se puede bajar, ya que es la encargada de mantener erguido el cuerpo.
- Como se ha reducido mucho la fuerza en los pies y rodillas, el robot tiende a caerse. Para estabilizar el movimiento hay que asegurar que el robot esté siempre "encima de los pies", para asegurar la estabilidad. Esto se consigue haciendo pasos cortos.
- Para que los pies no toquen el suelo siempre hay que imponer una cierta altura en el paso.

- La inclinación del torso tiene que ser aquella que haga que el robot no bascule hacia adelante ni hacia atrás, sino que se mantenga en la posición vertical.
- Se observa que los movimientos lentos son perjudiciales, dan tiempo a que el sistema se inestabilice. Si se hace el movimiento más rápido no sólo se consigue aumentar la velocidad de translación, sino la estabilidad. Esta idea se usa por ejemplo en el robot *BigDog*.

Estos parámetros dieron como resultado una locomoción estable y robusta, tal como se puede ver en el vídeo [5.1.3 ALWalk *low-stifness*, velocidad alta]. Además se produjo una mejora sustancial en las prestaciones de la locomoción, que se analizarán en el siguiente capítulo.

La aproximación mencionada en este apartado es una modificación del algoritmo que ya viene incorporado con Nao, no es en sí mismo un algoritmo nuevo. La formulación ZMP es ampliamente conocida e investigada, con los problemas mencionados anteriormente. Así, se decide iniciar otra vía de investigación basada en el aprovechamiento de las dinámicas propias del sistema, para poder comparar los resultados con los ya encontrados.

5.2. *Sammy's Walk*: Diseño de un algoritmo de locomoción bípeda

La naturalidad del estilo de caminar de los *passive walkers* resulta sorprendente. Además, el hecho de que los robots tengan un consumo tan bajo los convierte en un modelo a seguir. El algoritmo que se propone a continuación tiene el objetivo de producir una locomoción estable para Nao a partir del control del robot, suponiendo el comportamiento de este como dos péndulos unidos. Se buscará un estilo de control más próximo al control clásico de dimensiones reducidas, ya que de hecho la locomoción es un *ciclo límite estable* que se da en un sistema no lineal.

Este proyecto se propone como objetivos para conseguir una locomoción tipo *dynamic walking*:

- Inducir el ciclo límite que provoque la locomoción.
- Hacer que las órbitas del ciclo límite tengan un espacio de atracción mayor, es decir, estabilizar la locomoción.
- Aprovechar las dinámicas propias del sistema para reducir la energía consumida (este punto se tendrá más en cuenta en un análisis de prestaciones posterior que en el propio desarrollo).
- Aplicar leyes de control que se correspondan a reacciones lógicas que realizamos los humanos, no necesariamente buscar algoritmos basados en formulaciones matemáticas complejas.

Aunque hay muchos trabajos teóricos que tienen este espíritu (por ejemplo [7][11]), las implementaciones en robots reales son escasas. Este proyecto se parecerá en forma a otros

y tomará ideas de ellos, pero a su vez diferirá de todos. Dado que este desarrollo es relativamente novedoso, es de esperar que no se consiga madurar suficientemente para obtener resultados comparables a otros ya existentes. Sin embargo puede suponer un buen comienzo para investigaciones futuras.

También es de esperar que ciertos resultados teóricos no se puedan aplicar a la práctica sobre el robot real. Por tanto a medida que avance la investigación se irán indicando los problemas que se encuentren y sus posibles soluciones.

Como inspiración se tomarán ideas de la locomoción pasiva (no se usará la filosofía de crear trayectorias tipo ZMP, sino inducir ciclos límite), algoritmo SIMBICON (control por máquina de estados), BigDog (pasos rápidos, controlando con la posición del siguiente paso, simetría), marcha humana (ángulos articulares) y otras que se irán explicando durante el desarrollo.

5.2.1. Inducción de un ciclo límite

Caminar en línea recta es de hecho la superposición sincronizada de dos movimientos:

- Oscilación lateral.
- Avance.

En primer lugar se buscará inducir una oscilación lateral y sobre ésta se buscará sobreponer el movimiento de avance.

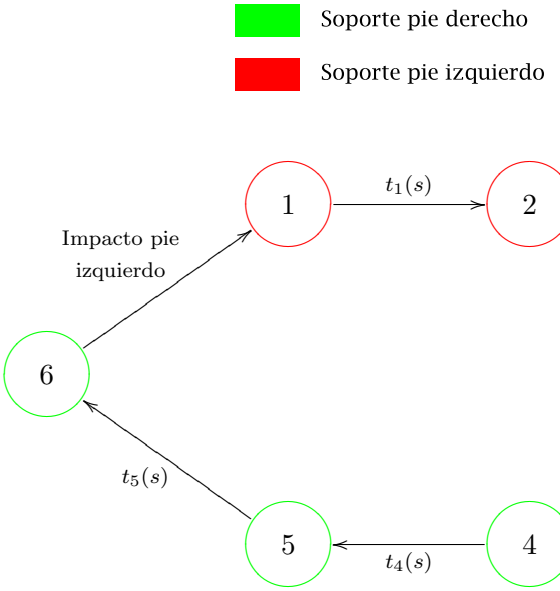
Para generar las trayectorias se opta por utilizar una máquina de estados que alterne entre diferentes posturas. Este algoritmo es el que se usa en SIMBICON [27]. Sin embargo, SIMBICON no se ha implementado en robots reales, así que su aplicación en Nao es una novedad. Además, no se usará directamente, sino que sólo se cogerá el concepto.

Existen otros robots, como por ejemplo *Spring Flamingo* [20], que también usan una máquina de estados. Sin embargo, este robot en concreto sólo se mueve en una dirección, ya que el movimiento lateral está bloqueado, y también proviene de una formulación matemática compleja. En el vídeo [5.2.1 Spring Flamingo] se puede ver su comportamiento.

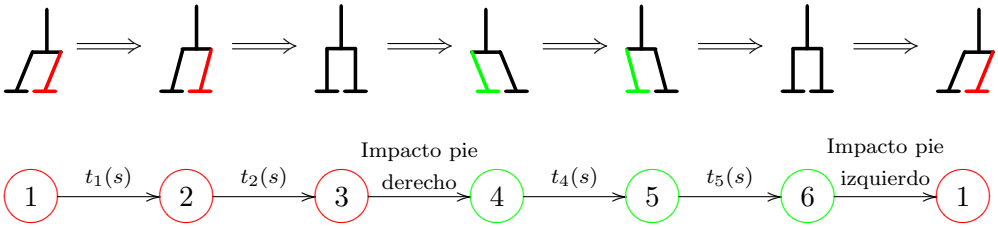
La máquina de estados implementada es la descrita en la Figura 5.5. Se usarán tres estados por fase de soporte para poder tener un mayor control. Ya que se pretende lograr un movimiento rectilíneo, los estados para la fase de soporte izquierdo 1, 2, 3 serán simétricos a los de la fase de soporte derecho 4, 5, 6. Así que realmente la locomoción se genera a partir de tres estados. El paso de los estados $1 \rightarrow 2$ ($4 \rightarrow 5$) y de $2 \rightarrow 3$ ($5 \rightarrow 6$) se producirá después de un tiempo determinado ajustable. El paso de $3 \rightarrow 4$ se producirá con un evento externo de sincronismo, el impacto del pie derecho (de $6 \rightarrow 1$ pie izquierdo)[2]. Cada postura será una configuración articular del robot.

Asimismo, hará falta definir la evolución de un estado a otro. Si los cambios se hacen directos, Nao usa el par máximo disponible y provoca movimientos muy bruscos e indeseables. Si se aplica el método de reducción de *stiffness* el robot pierde fuerza. Los movimientos continúan siendo muy rápidos, pero llega un punto en que si se reduce más, no es capaz de levantarse. No hay un punto intermedio que logre un movimiento suave.

[2]Para detectarlo se usarán los sensores de fuerza que Nao incorpora en los pies.



(a) *Grafo de estados y condiciones de transición*



(b) *Vista frontal del movimiento lateral*

Figura 5.5: *Máquina de estados*

El algoritmo SIMBICON usaba para la generación de estas trayectorias un controlador PD, ajustado de tal forma que los movimientos fueran suaves. Las articulaciones de Nao están en principio controladas por un controlador "tipo" PID, de parámetros configurables. Sin embargo, tras realizar diferentes pruebas se comprueba que el ajuste de los supuesto parámetros K_p, K_i, y K_d no da los resultados que se esperarían. La documentación no da detalles concretos si realmente es un PID ni de su estructura. Por tanto, la última opción que queda es usar algún tipo de interpolación. En vez de usar algún tipo de ajuste tipo polinómico o curvas *bézier* que son relativamente complejos, se escoge otra opción que ha dado grandes resultados.

Las consignas de las articulaciones, al venir de la máquina de estados, son de hecho escalones. Para suavizarlos se puede usar un filtro pasa bajos. Para evitar que en el instante que se produce el escalón ya haya salida, se usará un filtro simple de segundo orden, con dos polos reales. Se ha escogido de tal forma que tenga ganancia unitaria ante entradas escalón. Así, la función de transferencia discreta del filtro será:

$$G(z) = \frac{(1-a)^2 z}{(z-a)^2}, \quad a \in \mathbb{R}, \quad 0 < a < 1 \tag{5.1}$$

donde a es la posición del polo en el eje real, un parámetro ajustable que regula el tiempo en que se llega al valor final. Se escoge la posición del polo tal que tenga módulo menor que 1 (estable), que sea real y positivo para que no haya oscilaciones. Si $a \to 1$, el comportamiento será más lento, si $a \to 0$ el comportamiento será más rápido.

Recordamos que el código en tiempo real que se ejecutará en Nao tiene un período de $20ms$. Con este tiempo de muestreo, la Figura 5.6 muestra la respuesta del filtro para una entrada escalón unitario para varios valores del parámetro a.

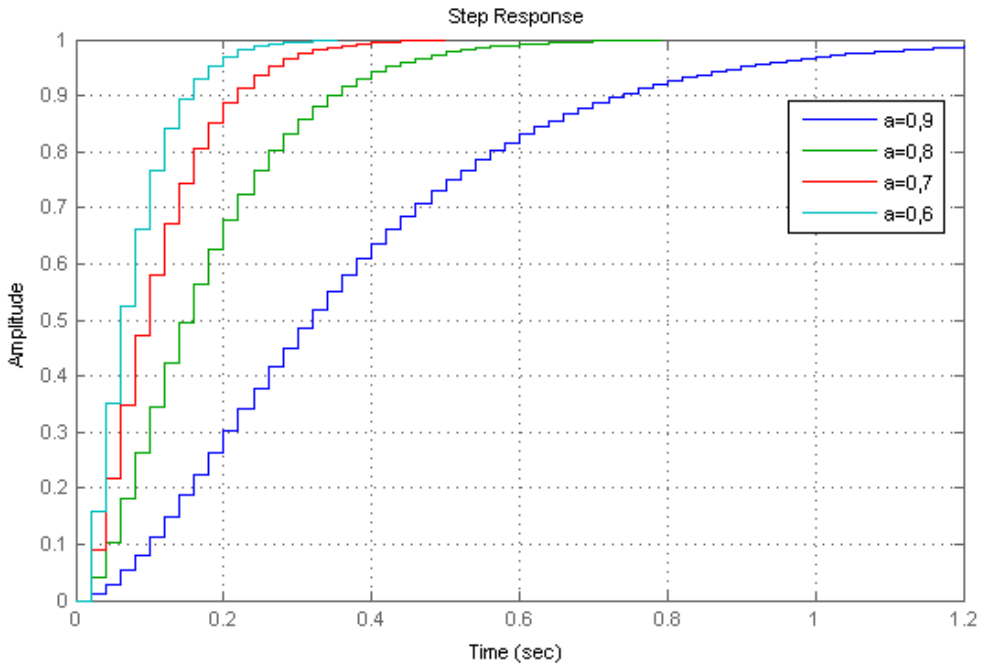


Figura 5.6: *Respuesta del filtro pasabajos al escalón unitario*

Ahora que ya está definida la estructura del algoritmo, se procederá a su implementación.

5.2.2. Simulación con *Webots*

En primer lugar se pensó en usar el entorno *Webots* para hacer las primeras pruebas con el simulador en vez de con el robot real. Esto debería permitir un mayor rapidez en las pruebas de parámetros y mayor seguridad, ya que aunque el robot se caiga es una simulación. Aunque la programación es relativamente sencilla, solventar las limitaciones de la versión demostración y aprender los comandos específicos de *Webots*, la estructura de programa, la configuración de las simulaciones desde cero no fue tarea fácil.

Entre otros se afrontaron los siguientes desafíos:

- Aprendizaje de la metodología de programación de Webots, y del robot simulado.
- Desarrollo de la primera versión del algoritmo *Sammy's Walk*.

- Para la locomoción resulta un punto clave el contacto con el suelo. Al ir a buscar oscilaciones del sistema físico, y no solamente movimientos lentos, se apreciaba que el contacto con el suelo provocaba comportamientos aleatorios indeseados.

Tras bastantes pruebas iterativas se consiguió inducir una oscilación lateral (ver vídeo [5.2.2 Simulación con Webots: Oscilación lateral estable]), y también que el robot caminara de forma estable relativamente rápido (ver vídeo [5.2.2 Simulación con Webots: Marcha]). El problema fue cuando se pasó a la implementación en el robot. El comportamiento era totalmente diferente, cuando en *Webots* caminaba bien, el robot real se caía estrepitosamente. Estas diferencias del robot real respecto al simulado se pueden deber a múltiples motivos, como por ejemplo:

- Desviaciones el los parámetros del modelo respecto al robot real.
- Errores numéricos de la simulación.
- Imprecisiones en el modelo. Se aprecia por ejemplo que los pies en el robot simulado son de base rectangular mientras que el robot real tienen una curvatura. Este aspecto es muy relevante.
- Imperfecciones del robot real. Las articulaciones y sus controladores no son perfectos, tienen juego mecánico, y existen deformaciones en las articulaciones.
- La superficie de contacto es un parámetro que varía mucho el comportamiento del robot, y es muy difícil encontrar una correspondencia entre el modelo y el comportamiento del suelo real.

Por tanto, esta vía no resulta práctica para el desarrollo del robot real. La liga *Robocup* que usa Webots tiene movimientos mucho más estáticos, y por tanto muchos de estos problemas no se afrontan. Pero para este proyecto se dejará de usar el simulador, y se harán las pruebas directamente con el robot físico, ya que se tiene acceso a él.

5.2.3. Oscilación lateral

Tal como se ha explicado anteriormente, el objetivo es inducir una oscilación lateral estable, y cruzarla con el movimiento de avance para de esta forma inducir un ciclo límite estable que provoque la locomoción. En un principio se usarán únicamente las piernas, los brazos se tratarán posteriormente. La oscilación lateral se pretende inducir únicamente con movimientos laterales (sin doblar las rodillas por ejemplo), es decir, con las articulaciones *roll* del robot.

Para inducir la oscilación lateral, hay que hacer primero un pequeño análisis del comportamiento del sistema. El sistema es equivalente a dos péndulos invertidos, mientras se está apoyado sobre un pie, el derecho por ejemplo, el sistema tiende a caerse. Si se cae hacia el lado del pie izquierdo, se producirá una transferencia de peso y pasará a estar apoyado sobre este último. Entonces el comportamiento del sistema cambia y pasa a moverse como otro péndulo, esta vez con la base en el pie izquierdo.

Sin embargo, es un sistema sobre el que se puede actuar. Como los pies tienen cierta superficie, la reacción del suelo no necesariamente está colocada sobre el centro, puede estar en los laterales. Esto hace que la articulación de los pies pueda ejercer un momento limitado sobre el cuerpo, que se puede controlar. Además, haciendo cambios en las articulaciones se pueden hacer cambios en los parámetros básicos del péndulo, por ejemplo su longitud, y así modificar su dinámica. También hay que recordar que en el momento del impacto de cada pie se producen pérdidas de energía, y por tanto el movimiento se irá amortiguando.

Las ideas que se ha seguido para inducir la oscilación lateral con un coste de energía mínimo son las siguientes, provenientes del control y el conocimiento de los sistemas físicos:

- Los sistemas físicos tienen frecuencias propias de oscilación, f_p.
- Si el sistema se excita a frecuencias parecidas a f_p, se tenderá a mantener la oscilación.
- En un principio, si la energía aportada por la oscilación es insuficiente, esta se extinguirá. Si se aumenta la energía puede llegar un momento que se automantenga. Si finalmente se aporta todavía más energía, o a frecuencias muy próximas a la frecuencia propia, el sistema puede llegar a convertirse en inestable.

Así, en primer lugar se analizará cuál es la frecuencia propia de oscilación lateral del sistema. En un péndulo normal, la frecuencia de oscilación para ángulos pequeños es de:

$$T = 2\pi\sqrt{\frac{l}{g}}, \quad f = \frac{1}{2\pi}\sqrt{\frac{g}{l}} \tag{5.2}$$

donde T es el período de oscilación, f la frecuencia, l la longitud del péndulo y g la gravedad. Para ángulos pequeños, la frecuencia no depende de la amplitud, para ángulos mayores sí. Aunque el comportamiento del robot no se corresponde al de un péndulo (es de hecho un doble péndulo inverso), se espera encontrar un comportamiento similar: al disminuir la altura del robot la frecuencia debería aumentar, y al aumentar la amplitud (apertura de las piernas) debería disminuir.

Los experimentos se realizarán levantando el robot de un pie y dejándolo caer, grabando la información de los sensores inerciales. En concreto se obtendrá la información del ángulo de inclinación respecto la vertical, la velocidad angular y la aceleración en la misma dirección. Estos resultados se analizará por medio de una FFT[3] para obtener la frecuencia propia.

En primer lugar se realizará el experimento con el robot totalmente de pie, y las piernas paralelas (apertura 0º) En la Figura 5.7 se pueden ver los gráficos, y en [5.2.3 Oscilación lateral propia] el vídeo del experimento.

La inclinación respecto a la oscilación vertical se obtiene del robot directamente en grados sexagesimales (º) Esta medida se obtiene mediante un algoritmo interno del robot que usa la aceleración y la velocidad angular para calcular el ángulo. Se observa que tiene una cierta deriva, aunque no afecta al comportamiento dinámico. La velocidad angular no es una medida directa, sino que cada unidad corresponde aproximadamente a 1/2,7 °/s. Esta medida se muestra la mejor para obtener la frecuencia de oscilación. La aceleración tampoco da una

[3] *Fast Fourier Tranform*, transformada rápida de Fourier.

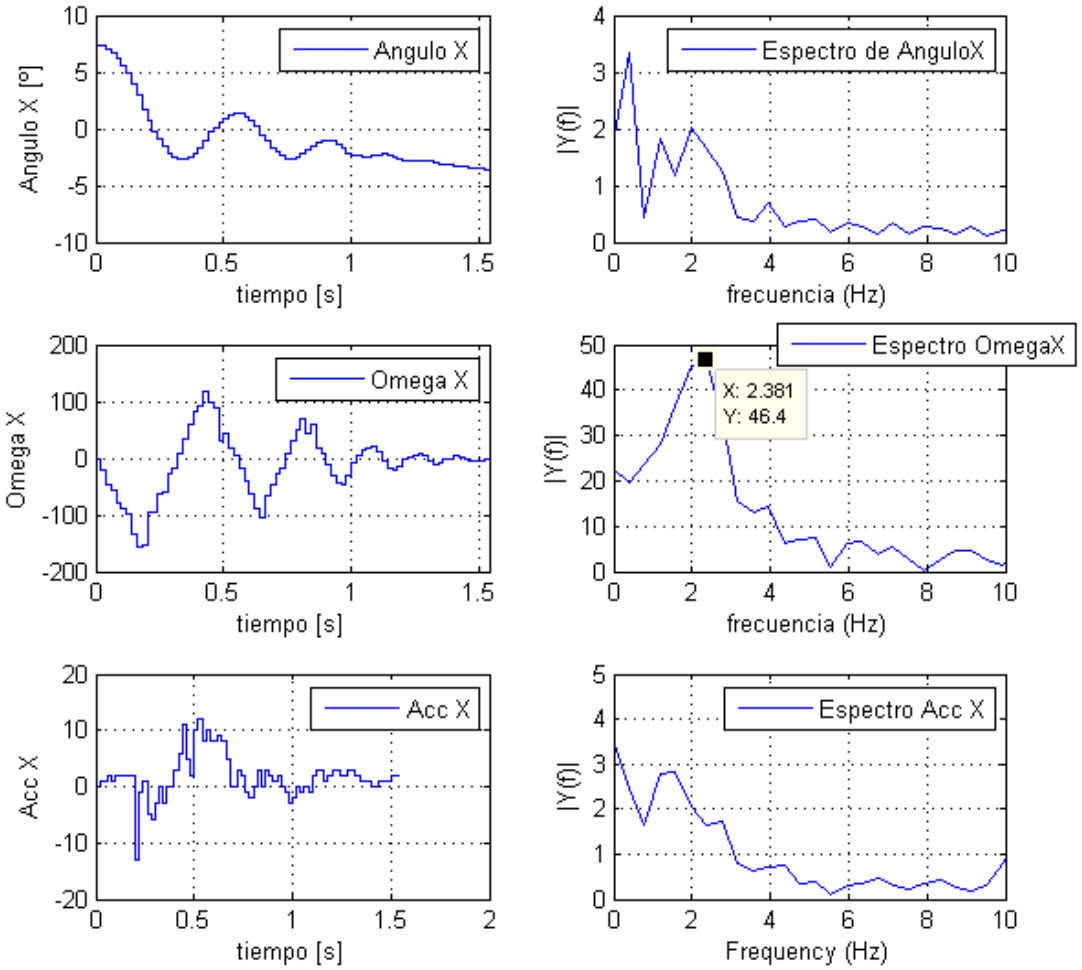


Figura 5.7: *Auto-oscilación con el robot de pie, apertura de las piernas 0º, altura caderas 26cm*

medida directa del valor, aproximadamente un valor de 56 equivale a g $(9,8m/s^2)$. Como tampoco es importante el valor absoluto, no se han corregido. La aceleración se observa que tiene mucho ruido, y su espectro no es como se esperaría. De los análisis frecuenciales se ve que la frecuencia de oscilación es de $f_p = 2,38hz$, y por tanto $T_p = 420ms$.

Se han hecho experimentos para aperturas entre piernas de 10º y 20º (manteniendo el robot totalmente erguido). También se han hecho pruebas para el robot un poco agachado (ángulo de la articulación de la rodilla de 60º) y muy agachado (ángulo de la articulación de la rodilla de 120º). Los resultados se muestran en la Tabla 5.3, y los gráficos en la Figura 5.8:

Apertura piernas	0º	10º	20º
Frecuencia oscilación	$2,38hz$	$2,38hz$	$2,38hz$

Altura caderas	26cm	23cm	17cm
Frecuencia oscilación	$2,38hz$	$2,42hz$	$2,42hz$

Tabla 5.3: *Frecuencia de oscilación según configuración*

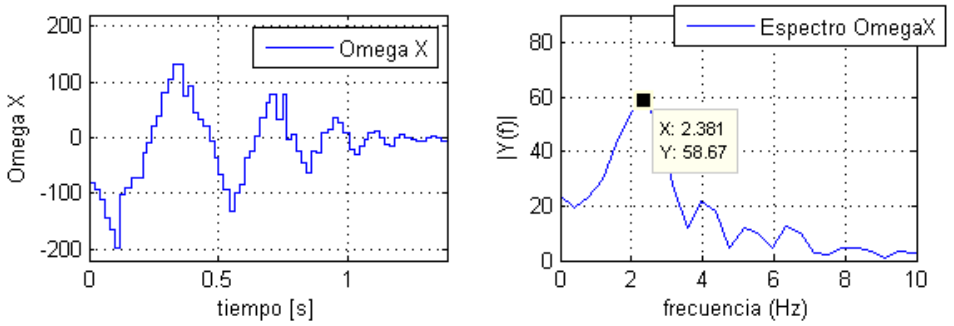
Se observa que los valores son todos muy similares, no se aprecian variaciones sustanciales por los cambios de configuración. Aunque sí parece que al bajar el centro de masa la frecuencia aumenta ligeramente, la diferencia es muy baja. Por tanto se toma como frecuencia propia de oscilación del sistema $f_p = 2,38hz$. Se tomará como configuración base el robot totalmente erguido, ya que en esa posición los pares necesarios en las articulaciones de rodillas, caderas y pies son menores. Por el mismo motivo se tomará de apertura de piernas 0º (piernas paralelas).

A partir de este valor de frecuencia se pueden sacar fácilmente los primeros valores tentativos de la configuración de la máquina de estados. Es de suponer que si se excita el sistema a la frecuencia propia, este se volverá inestable. En primer lugar se excitará para una frecuencia menor de $f = f_p/2 \approx 1,2hz$, escogida de forma arbitraria. En este caso el período sería de $T = 840ms$. Con este tiempo de ciclo, la mitad de un paso será aproximadamente $420ms$. Recordemos que el paso del estado $3 \rightarrow 4$ viene dado por un evento, no por un tiempo, así que no se puede imponer directamente, pero sí es de esperar que este tiempo sea pequeño cuando el sistema esté oscilando de forma nominal. Así, se tomará $t_1 = t_4 = 150ms$, y $t_2 = t_5 = 150ms$ de forma arbitraria para que se cumpla aproximadamente el período encontrado.

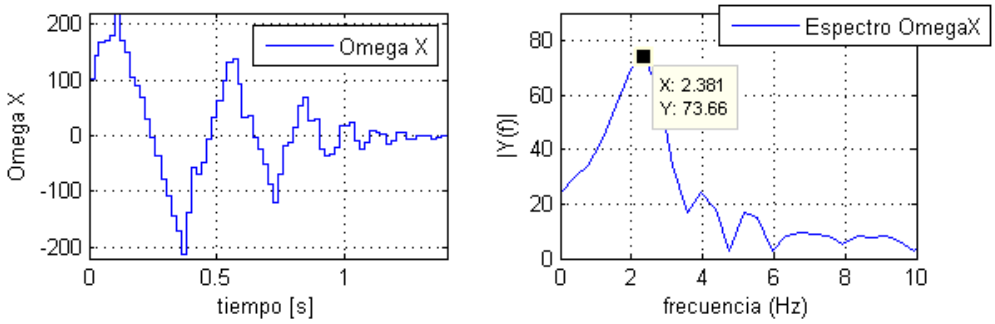
Respecto al parámetro a, que es la "velocidad" con que se llega a los estados, se optará por una dinámica lenta. No se pretende que se llegue a la configuración final en cada estado, sino *encadenarlos* de forma suave. Se toma en primera instancia un valor de $a = 0,8$, para que así en el tiempo que dura cada estado no se logre llegar al valor final (ver Figura 5.6 para ver la respuesta del filtro).

Ahora sólo es necesario definir los estados articulares. Estos se definirán únicamente para las piernas, y para la mitad de ciclo, ya que para la otra se imponen simétricos. Tal como se mencionó anteriormente, sólo se usarán los movimientos laterales, es decir las articulaciones *R/LAnkleRoll* y *R/LHipRoll*. Como criterio estilístico se impondrá que los movimientos de ambas piernas sean paralelos, es decir, que a partir de una apertura de piernas dada, el movimiento sea igual para ambas piernas. Así que idealmente en este plano sólo se tiene un grado de libertad, la inclinación de las piernas, φ (ver Figura 5.9).

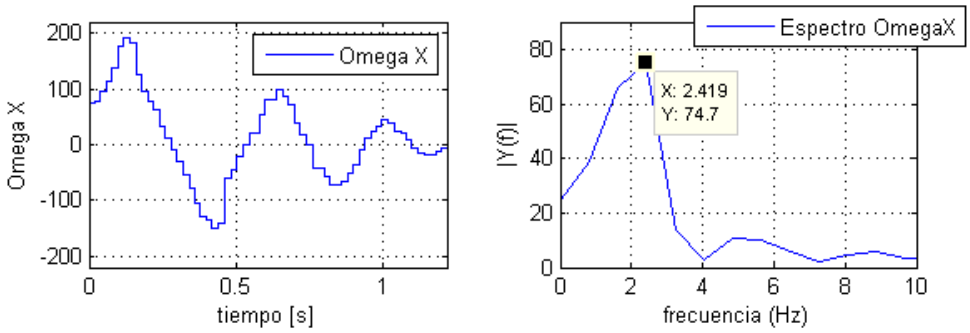
Se puede observar que la transferencia de peso se realiza aproximadamente cuando el valor



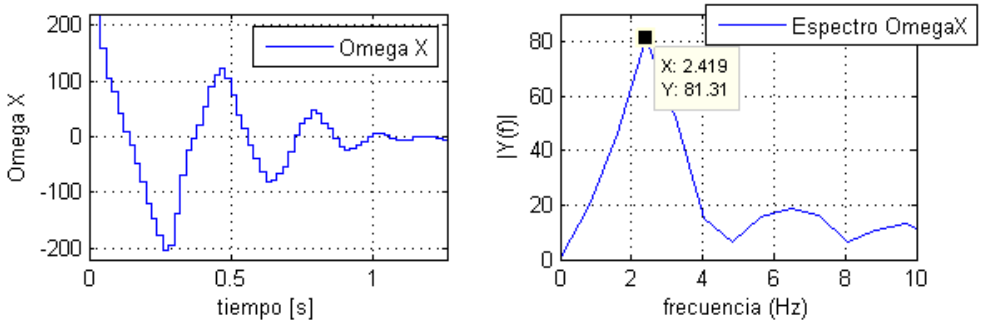
(a) *Erguido, apertura piernas 10º*



(b) *Erguido, apertura piernas 20º*



(c) *Agachado, altura caderas 23cm*



(d) *Agachado, altura caderas 17cm*

Figura 5.8: *Frecuencia de oscilación según configuración*

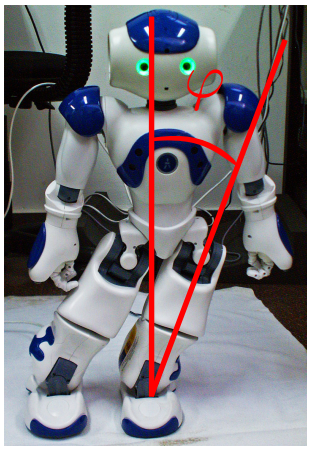


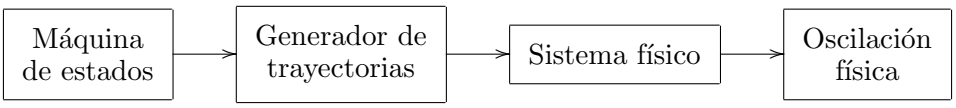
Figura 5.9: *Movimiento lateral*

$\varphi = 20^o$. Como en el estado no se llega al valor final, se impondrá para el estado 1 $\varphi_1 = 30^o$, y para el resto de estados se asigna de forma intuitiva los valores $\varphi_2 = 10^o$, $\varphi_3 = 0^o$, $\varphi_4 = -30^o$, $\varphi_5 = -10^o$, $\varphi_6 = 0^o$. Estos parámetros también se toman inspirados en los equivalentes para el cuerpo humano.

Después de todo este razonamiento llega el momento de probarlo en el sistema, y en este momento se produce la magia. El sistema empieza a oscilar de forma estable haciendo un movimiento de vaivén (ver Figura 5.11 y 5.12, y el vídeo [5.2.3 Balanceo lateral estable]). Aunque se ha impuesto que las piernas tengan un movimiento paralelo y sería de esperar que el torso estuviera vertical durante el movimiento, debido a los huelgos de las articulaciones y a que los controladores de bajo nivel en las articulaciones tienen también su error de seguimiento. Así, se ha logrado inducir un ciclo límite estable en la oscilación lateral. Éste se ha representado en la Figura 5.10 (el eje de velocidad se muestra la medida del sensor sin unidades). Tiene una frecuencia $f = 1,373hz$, $T = 728ms$. Uno de los motivos que funcione bien, y que permite inducir y estabilizar el ciclo límite, es que los estados 3 y 6 se sincronizan con la evolución del sistema.

Ahora se explorará los límites del parámetro φ. Si se cogen valores mayores a 40^o, el robot cae hacia uno de los lados, con valores inferiores a 15^o la oscilación no llega a levantar los pies, por tanto insuficiente. Respecto a los parámetros de tiempo t_i, tanto si se aumentan como si se disminuyen la oscilación o se vuelve inestable, o golpea de forma brusca con el suelo.

Para entender mejor la dinámica del robot con este algoritmo hay que hacer un análisis más profundo. El sistema se puede considerar formado por varios bloques:



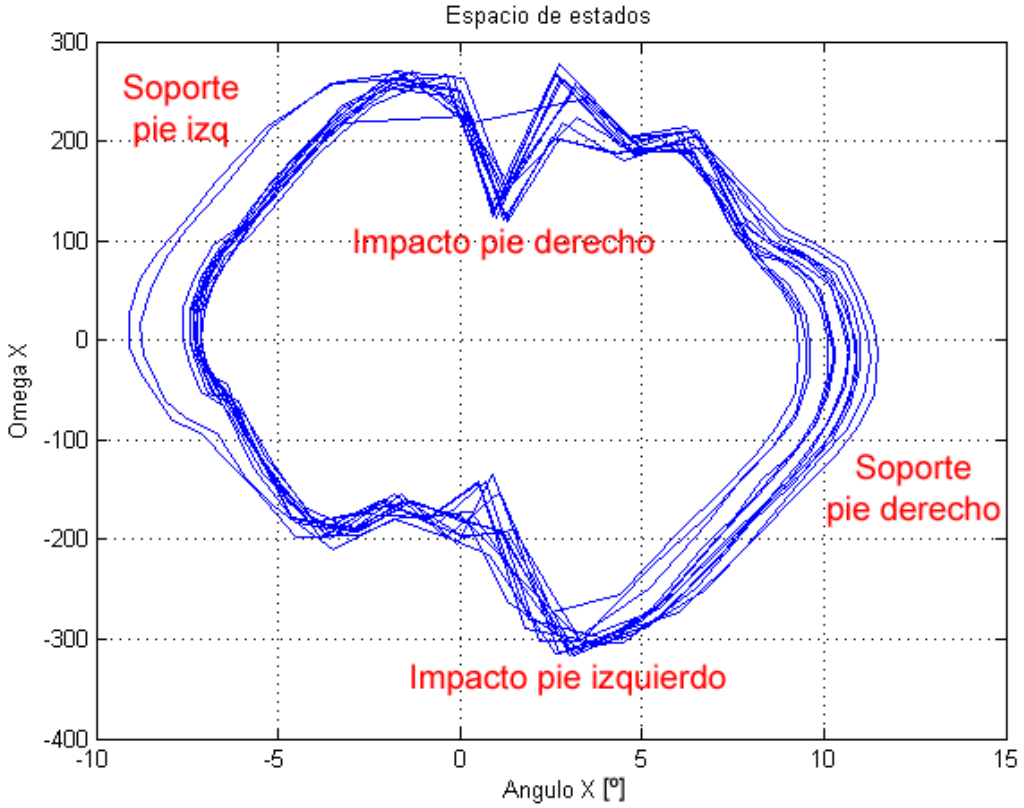


Figura 5.10: *Ciclo límite lateral*

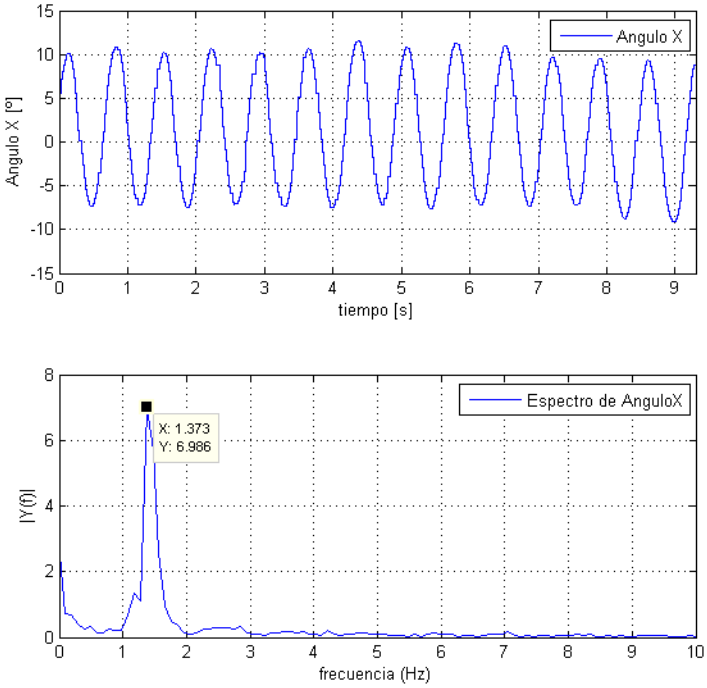


Figura 5.11: *Oscilaciones laterales inducidas – Ángulo X respecto la vertical*

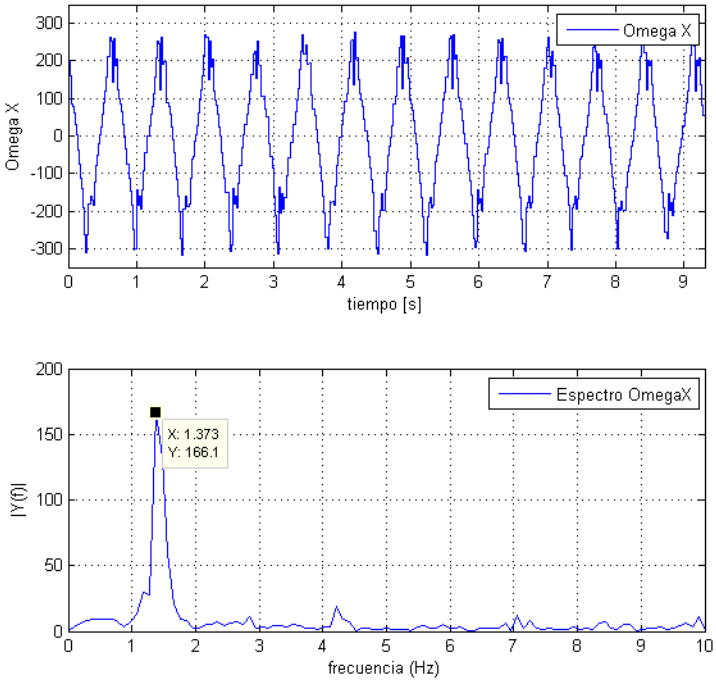


Figura 5.12: *Oscilaciones laterales inducidas – Velocidad angular X*

La máquina de estados se puede considerar un oscilador con una cierta frecuencia y amplitud, y los sucesivos bloques presentaran sus dinámicas propias, modificando en amplitud y fase la oscilación original. En concreto, el generador de trayectorias presentará un desfase importante para valores de a cercanos a la unidad (para la frecuencia de oscilación descrita anteriormente, y un valor de $a = 0,9$, el desfase es mayor a 200ms). El sistema físico, el robot, también presentará un cierto desfase, y por tanto la oscilación real no se corresponderá a la de la máquina de estados.

Este punto resulta un problema, ya que la máquina de estados está concebida para que en cada estado el sistema esté en una configuración determinada. Se podría hacer un análisis teórico para determinar los retardos de cada bloque, y corregirlo de tal forma que la máquina de estados estuviera en fase con la oscilación inducida deseada. Sin embargo, de forma experimental se puede corregir más fácilmente.

Para que se de este sincronismo, el tercer estado de la máquina de estados debe ser tal que el robot se esté moviendo hacia su derecha, apoyado en su pie izquierdo, y el pie derecho impacte, pasando así al cuarto estado. Haciendo un análisis temporal de las variables implicadas (estado, velocidad angular, contacto de los pies) se observa que el impacto del pie derecho *se produce en el segundo estado*. Para corregir este problema y sincronizar la máquina de estados con la oscilación del robot basta con desplazar los estados, hacer que el primero sea el segundo, el segundo el tercero, y el tercero el primero. De esta forma se consigue que en el tercer estado el robot esté sobre el pie izquierdo, se esté moviendo hacia la derecha y el pie derecho impacte en el suelo, tal como se diseñó inicialmente. La contrapartida es que se pierde un poco de vista en la configuración de la máquina de estados el significado físico de los valores, al no corresponderse directamente.

Haciendo un ajuste manual más fino se llega a la siguiente configuración de parámetros que realizan un movimiento suave y estable: $\varphi_1 = -\varphi_4 = 0^o$, $\varphi_2 = -\varphi_5 = 30^o$, $\varphi_3 = -\varphi_6 = 0^o$, $t_1 = t_4 = 100ms$, $t_2 = t_5 = 150ms$, $a = 0,9$. Con estos parámetros se produce una oscilación un poco más rápida ($f = 1,65hz$, $T = 605ms$), pero que resulta más estable ante perturbaciones, y tiene menor amplitud.

Para verificar la robustez y los límites de atracción de las órbitas se opta por dar golpes al robot en momentos determinados y ver su comportamiento. Las pruebas se pueden ver en el vídeo [5.2.3 Respuesta a perturbaciones externas]. Para pequeñas perturbaciones el sistema se vuelve a estabilizar.

Resulta interesante ver el consumo energético para esta oscilación (ver Figura 5.13). Al principio, cuando el robot se levanta, el pico de intensidad llega casi a 2,6A, después va disminuyendo hasta 1,3A. Para ver el consumo de intensidad empleado en la locomoción hay que restarle el consumo de la CPU, controladores y demás elementos no activos. Si se apagan totalmente los motores, el consumo de estos elementos es de 0,95A. Por tanto el consumo empleado en la locomoción mecánica es solamente de $I_m = 0,35A$ (aproximadamente la mitad que cuando caminaba con el algoritmo ZMP y *low-stiffness*). Se observa que al *tener en cuenta las frecuencias propias del sistema se consigue un consumo energético muy reducido*. El aspecto del consumo se tratará posteriormente en el capítulo 6 de análisis de prestaciones.

Ahora se tratará el tema de cómo ampliar la atracción del ciclo límite para hacerlo más robusto.

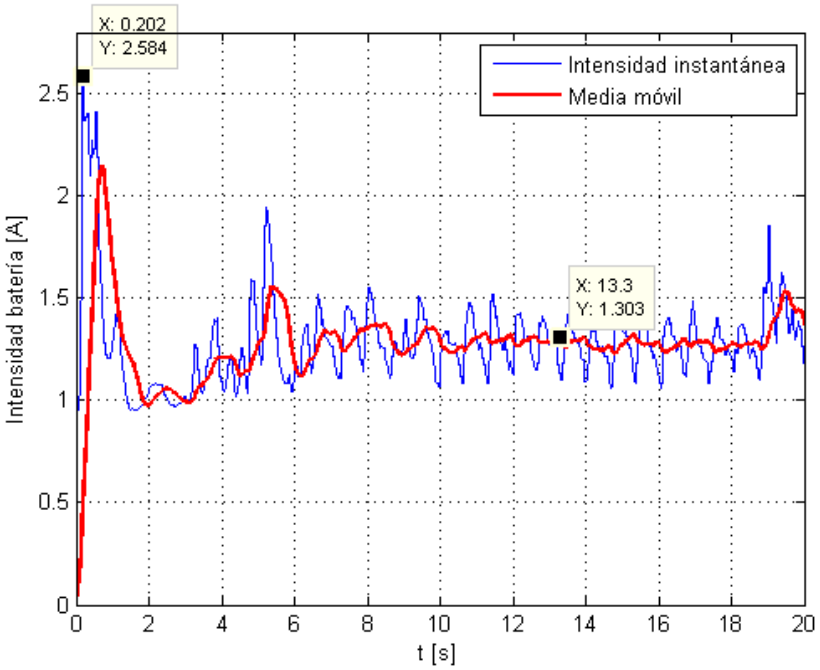


Figura 5.13: *Intensidad consumida en la oscilación lateral inducida*

5.2.4. Incremento de la estabilidad del ciclo límite

En [10] se encuentra una investigación interesante sobre como estabilizar el movimiento lateral en un caminador pasivo. Los diferentes métodos expuestos se pueden resumir en:

1. Control directo por realimentación de estado.
2. Cambio en la posición del siguiente paso. A partir de una medida de la perturbación se corrige la configuración de la pierna para estabilizar el paso. Las ecuaciones usadas son:
$$\Delta E = \frac{1}{2} I_x \left(\omega_x^2 - \omega_x^{*2}\right) = \frac{1}{2} I_x \left(\Delta\omega_x^2 + 2\omega_x\omega_x^*\right) \tag{5.3}$$
donde ΔE es la energía de la perturbación que se supone instantánea, ω_x la velocidad angular de la trayectoria nominal para la configuración dada, ω_x^* la velocidad angular actual con la perturbación, $\Delta\omega_x = \omega_x^* - \omega_x$, y I_x la inercia. Esta corrección presupone que el sistema pueda contrarrestar de forma quasi-instantánea la perturbación (a partir de un cambio de configuración que reduzca en el mismo valor la energía potencial por ejemplo).
3. Uso de una rueda de reacción en el torso para estabilizarlo.
4. Cambio en la inclinación del torso.
5. Actuación lateral en los tobillos.
6. Cambio del ancho del paso.
7. Disminución de la *stiffness* de los tobillos lateralmente para absorber las perturbaciones.

En la aplicación práctica a Nao y al ciclo límite ya inducido, se pueden hacer los siguientes comentarios sobre cada método:

1. Esta vía ya se descartó en el planteamiento inicial.
2. La medida exacta de la perturbación es inviable con los sensores que se disponen. La implementación directa de esta manera es inviable, y una posible compensación difícil.
3. No se dispone de este mecanismo.
4. Esta opción puede resultar viable.
5. De hecho, este método es el usado para inducir el ciclo límite y estabilizarlo. Ahora lo que se requiere es aumentar la estabilidad.
6. Esta opción puede resultar viable.
7. El propio artículo expone que no es un método viable. Se ha probado de todas formas, y las consecuencias son que como se pierde actuación, la oscilación lateral se amortigua, y tampoco se consigue el efecto deseado de aumentar la estabilidad.

En esta aproximación se usará la técnica de modificar la longitud del paso. Una reflexión sobre la mecánica del movimiento puede dar ideas de cómo reaccionar ante una perturbación. Suponiendo que el movimiento se realiza en el plano lateral, en la fase de soporte de un único pie el robot está de hecho rotando con una cierta velocidad angular alrededor del pie o uno de sus bordes. En el ciclo límite nominal para cada fase hay una energía cinética determinada, que se va intercambiando con la potencial. Tal como se mencionó en el segundo punto, una perturbación se puede considerar una variación en la energía cinética de rotación, esto es, una variación en la velocidad angular. Si la perturbación hace incrementar la velocidad angular (y la energía), es posible que en el siguiente paso el robot se caiga hacia afuera. Si la velocidad angular disminuye, no tendrá suficiente energía y acabe cayendo hacia adentro del paso.

Si la energía ha aumentado, es necesario que el siguiente paso sea más abierto, para que de esta forma cuando el otro pie entre en contacto tenga un recorrido mayor, es decir, tenga que utilizar más energía potencial, y viceversa. En el Figura 5.14 se ilustra esta idea.

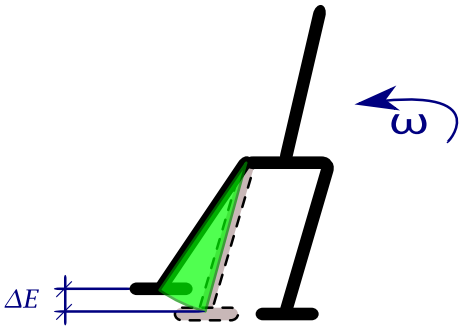


Figura 5.14: *Diagrama de corrección del ancho de paso*

Existe una posibilidad en que es necesario modificar esta idea. Si se imagina que el robot está apoyado sobre su pie derecho, y se está cayendo muy rápido hacia su derecha no se arregla nada abriendo más el paso, ya que acabará cayendo hacia afuera. En este caso la corrección lógica sería cruzar las piernas sin que colisionen para colocar el siguiente paso más a la derecha. Este comportamiento requiere una planificación ya que pretende corregir perturbaciones muy grandes, y no se considerará en este estudio. El objetivo de este apartado es aumentar la región de atracción del ciclo límite ya inducido para *pequeñas perturbaciones*, y para estas el método planteado es válido.

En la ecuación 5.3 se muestra el incremento de energía cinética de la perturbación. ω_x se puede determinar a partir de un análisis del ciclo límite, ω_x^* se puede medir con los sensores *on-line*. Este incremento de energía cinética debe corregirse idealmente con un incremento idéntico en la energía potencial del siguiente paso. El trabajo [10] previamente citado propone esta idea, con una formulación que se expone aquí de forma simplificada:

$$E_p = m \cdot g \cdot h = m \cdot g \cdot L(\mathbf{q}) \cdot cos(\theta) \tag{5.4}$$

donde E_p es la energía potencial del sistema, m la masa, g la gravedad, $L(\mathbf{q})$ la distancia entre la base del pie soporte y el centro de masa, que depende de la configuración del robot $\mathbf{q}$. Suponiendo que aproximadamente L es constante, se puede calcular el incremento diferencial en la energía potencial como:

$$\Delta E_p \approx -m \cdot g \cdot L \cdot sin(\theta) \cdot \Delta\theta \tag{5.5}$$

Imponiendo que este incremento de energía sea igual al de la perturbación, se obtiene:

$$\Delta\theta = \frac{\Delta E}{-m \cdot g \cdot L \cdot sin(\theta)} = \frac{\frac{1}{2} I_x \left(\Delta\omega_x^2 + 2\omega_x\omega_x^*\right)}{-m \cdot g \cdot L \cdot sin(\theta)} = -k\frac{\left(\Delta\omega_x^2 + 2\omega_x\omega_x^*\right)}{sin(\theta)} \tag{5.6}$$

Este planteamiento presenta varios problemas. Ya que es una aproximación diferencial, para la configuración $\theta = 0$ no se puede resolver. Además, desde el punto de vista de la implementación práctica, los datos provienen de sensores, con su consecuente ruido. Hacer operaciones como divisiones entre un número cercano al cero y cuadrados en el denominador traerán problemas numéricos e inestabilidades, que se han corroborado con diversas pruebas.

De este planteamiento se puede extraer que para corregir una velocidad angular ω^* diferente de la nominal hay que modificar la apertura de la pierna libre en un ángulo que depende del cuadrado de ω^* entre otras cosas. Sin embargo, el objetivo no era absorber la perturbación totalmente, sino lo suficiente como para que se regrese al ciclo límite. Además, dado el enfoque claramente experimental de este trabajo y las limitaciones que presentan los sensores, se opta por otra solución.

La corrección implementada hará una corrección en la apertura de la pierna proporcional a la desviación de la velocidad angular respecto a la nominal:

$$\Delta q_{HipRoll} = -\Delta q_{AnkleRoll} = -k\left(\omega_x - \omega_x^*\right) \tag{5.7}$$

donde $\Delta q_{HipRoll}$ es una variación en el ángulo de la articulación de movimiento lateral de la cadera de la pierna libre, $\Delta q_{AnkleRoll}$ ídem para la articulación lateral en el tobillo libre. Se aplica a ambas articulaciones para imponer que el pie sea en la medida de lo posible paralelo al suelo. Esta variación del ángulo no se pasará por el filtro de generación de trayectorias (ya que con el retraso existente volvería inestable el sistema), sino que se sumará a éste. La constante k se determinará ajustándola de forma experimental para que el sistema tenga un buen comportamiento. Además, se aplicará una zona muerta para evitar pequeñas oscilaciones indesables. ω_x^* será directamente la medida del giróscopo, filtrado por una media móvil de ancho 3 para eliminar un poco los ruidos. Para obtener la velocidad del ciclo límite nominal, ω_x, se tomará un valor aproximado correspondiente al valor medio para cada estado. Así, $\omega_{x,i}$ será función del estado i. Para determinar sus valores se hará la media entre el valor inicial y final para cada estado durante varios ciclos, y la media de todos ellos. Los valores obtenidos de la medida directa de los sensores son (en la Figura 5.15 se muestra una gráfica con los datos origen para realizar estos cálculos):

ω_1	-200,8	ω_4	173,7
ω_2	42,5	ω_5	-28,5
ω_3	188,0	ω_6	-249,3

Tabla 5.4: *Valores de la velocidad angular según el estado*

Experimentalmente se determina un valor de $k = 0,02$ y una zona muerta de 3º como valores que confieren estabilidad sin provocar autoosilaciones por ruido de los sensores. La bondad de los resultados se puede ver en el vídeo [5.2.4 Estabilización movimiento lateral]. Haciendo las mismas pruebas de provocar perturbaciones se comprueba el mejor comportamiento, ya que la realimentación de velocidad provoca una corrección predictiva. El espacio de

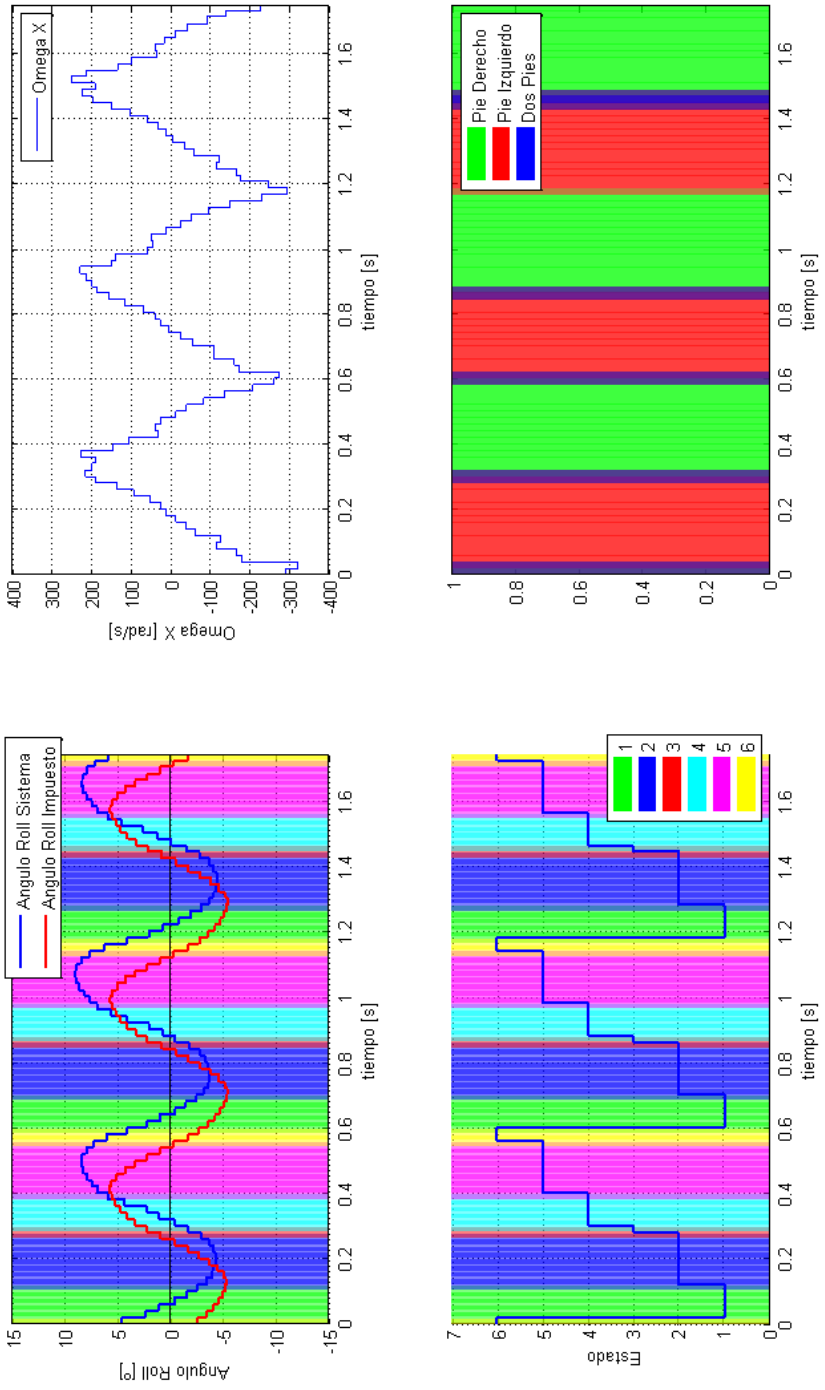


Figura 5.15: *Oscilación lateral nominal*

estados obtenido se representa en la Figura 5.16, donde se aprecia claramente que las órbitas de atracción son mayores. Una vez conseguido este punto, el siguiente apartado tratará cómo superponer el movimiento de avance.

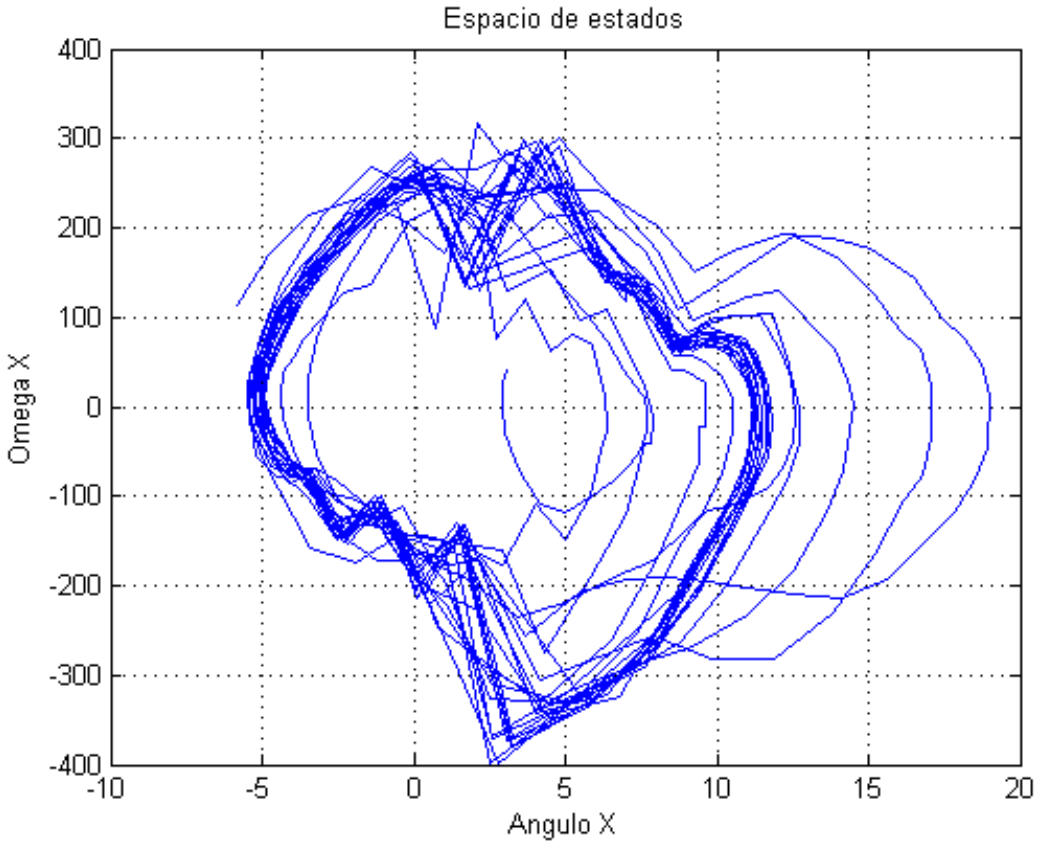


Figura 5.16: *Espacio de estados ante perturbaciones con corrección velocidad*

5.2.5. Sincronismo con movimiento de avance

Para cruzar la oscilación lateral con el avance es necesario que cuando un pie está levantado se mueva hacia delante. Tal como se vio en la Figura 5.5b, es de esperar que en el estado 1 y 2 el robot esté apoyado sobre el pie izquierdo, y por tanto se pueda avanzar el derecho. De forma análoga, en los estados 4 y 5 el robot es de esperar que esté apoyado sobre el pie derecho, y por tanto se avance el derecho.

Sin embargo, tal como se discutió en el apartado anterior, la máquina de estados concebida inicialmente no está en fase con la oscilación física. Por tanto se impondrá que la pierna oscilante avance (y la fija retroceda) en el mismo momento que el apoyo esté totalmente ubicado en la pierna soporte. Esto se producía en los estados 2 y 4.

Por tanto, se impondrá en el estado 2 que el valor deseado para la articulación frontal de la cadera de pierna oscilante sean -20º (hacia delante), la rodilla doblada 20º, y el tobillo recto para que permanezca paralelo al suelo. La pierna soporte se impondrá un comportamiento

parecido hacia atrás: cadera hacia atrás 20º, rodilla recta, tobillo -20º para que esté paralelo al suelo (ver Figura 5.17). En los estados 1 y 3 se dejaran las piernas rectas, al ser estados de transición.

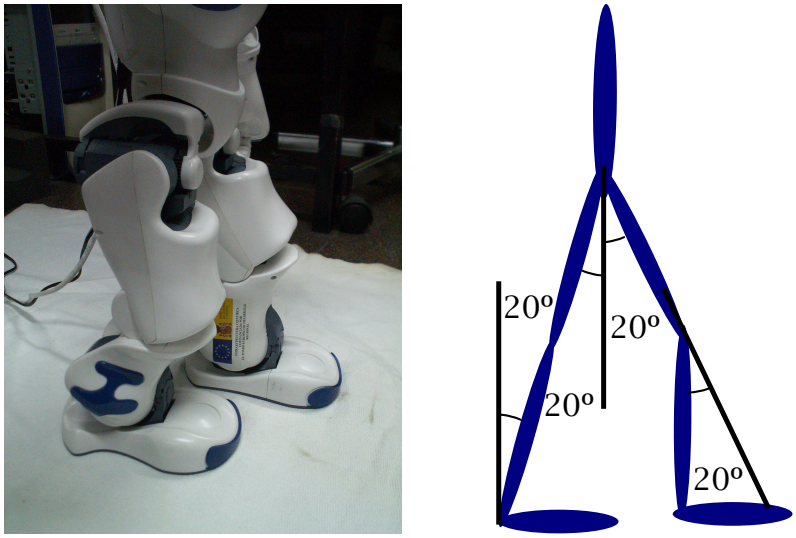


Figura 5.17: *Configuración de las piernas para el avance*

Con estos parámetros configurados, se prueba el comportamiento. El resultado es que el robot empieza a caminar, tal como se puede ver en el vídeo [5.2.5 Nao caminando con algoritmo Sammy's Walk].

5.2.6. Mejoras estilísticas y de estabilidad

Para mejorar la estabilidad del avance se han probado múltiples soluciones. El problema principal es que en el plano frontal el movimiento es muy inestable.

En primer lugar se pensó en usar el mismo concepto que con SIMBICON, que el torso y la pierna oscilante tengan como referencia el sistema mundo. Esto implica corregir instantáneamente la posición de las piernas y el torso en función de los sensores inerciales. Cuando se probó esto el sistema se puso a oscilar de forma violenta. Hay que tener en cuenta que existe un retraso entre que se recibe la información de los sensores y se actúa, y éste es suficiente como para inestabilizar el movimiento. Además, la oscilación en las medidas provoca pequeñas oscilaciones de alta frecuencia en los motores. Si se aplica un filtro como una media móvil estrecha para evitarlo, se incrementa el retraso y la inestabilidad.

Otra opción que se ha probado es corregir la posición de la pierna oscilante en función de la velocidad angular del torso, de modo parecido a como se hace lateralmente. Esto ha provocado también inestabilidades, y para ganancias bajas no era suficiente como para estabilizar el movimiento.

También se ha intentado reducir la *stiffness* de los tobillos para absorber perturbaciones, tal como se hizo con el algoritmo ZMP. Si se ponía un valor muy bajo el robot se caía porque era incapaz de mantenerse en pie, y con valores medios/altos continuaba inestable.

Finalmente se pensó en usar la información de los sensores de los pies para averiguar

hacia donde se estaba cayendo el robot, y así poder corregirlo. Se hizo un ajuste muy fino de cada sensor individualmente en estático, y con el robot quieto se consiguió amortiguar de forma suave las oscilaciones frontales a partir de la lectura de los sensores de fuerza de los pies. Si se notaba que el robot se inclinaba mucho hacia atrás se corregía, y viceversa. Cuando se probó esto con el robot en movimiento, empezó a oscilar de forma violenta. Tras múltiples intentos de corregir los parámetros los resultados fueron los mismos. Investigando los motivos, se encontró que los sensores tenían un comportamiento dinámico muy malo, y no eran válidos para esta aplicación.

Vistas que estas soluciones no funcionaban, se optó por reducir la amplitud frontal de los pasos hasta que el movimiento fuera estable. En el caso del algoritmo ZMP también se tuvo que optar por esta solución de acortar los pasos para estabilizarlo. El motivo se expuso en el apartado 3.1.2, si no existe la articulación de los dedos ni un mecanismo que permita la basculación talón–punta, como en el caso de Nao, la solución es acortar los pasos.

Respecto al movimiento de los brazos, se ha decidido no dotar de movimiento a éstos, ya que no son imprescindibles para la locomoción. Se deja para investigaciones futuras el estudio del movimiento de los brazos, siendo éste fácilmente implementable a partir del algoritmo ya desarrollado.

5.3. Conclusiones

Se ha conseguido diseñar un algoritmo que aprovecha las dinámicas propias del robot para inducir un ciclo límite estable que hace que el robot camine. Mediante el uso de atractores en el espacio de fases se ha conseguido aumentar la estabilidad lateral de la oscilación.

El movimiento frontal no se ha conseguido estabilizar con los métodos probados experimentalmente. Queda pendiente para estudios futuros el diseño de un algoritmo que estabilice el movimiento de avance.

Capítulo 6

Análisis de prestaciones

En este apartado se compararán las características de las locomociones conseguidas con los dos algoritmos que se han estudiado, *ALWalk* y *Sammy's Walk*. Los parámetros más interesantes de analizar son la velocidad de translación, la potencia involucrada y el coste de transporte c_{mt}. En el algoritmo *Sammy's Walk* se analizarán también otros aspectos.

6.1. Características de la locomoción con ALWalk

Resultaría interesante comparar la locomoción con ALWalk (algoritmo ZMP) con la *stiffness* de las articulaciones al 100 % con las modificaciones que se han hecho. Pese a los múltiples intentos de que la versión *full-stiffness* caminara bien, no se consiguió, por tanto no se puede comparar. El motivo más probablees que los robots difieren unos de otros y no con todos se puede conseguir.

En [9] se consiguió que alguna vez funcionará la versión *full-stiffness*, y se comparó con la versión *low-stiffness*. Los resultados se pueden resumir en la siguiente tabla:

Algoritmo	**Velocidad** $[cm/s]$	**ΔVelocidad** [%]	**Eficiencia**[1] c_mt
Algoritmo ZMP básico	$8,7 \pm 0,1$	100 %	5,8
Algoritmo con baja Stiffness	$13,9 \pm 0,2$	160 %	2,8

Tabla 6.1: *Configuración de Stiffness para cada articulación*

Se pueden ver las mejoras sustanciales. Los resultados encontrados con nuestro robot son parecidos a los de [9]. Se tiene una velocidad de translación de 10,3 cm/s, calculada en una distancia de 1m durante varios intentos. La intensidad consumida se puede ver en la Figura 6.1. El consumo medio es de 1,6A durante la locomoción, y el utilizado únicamente para los motores es de 0,65A[2], lo que supone una potencia consumida para la locomoción de 14,1W, y un $c_{mt} = 3,1$.

A nivel estilístico, ZMP produce una locomoción más limpia y constante. Sin embargo, los movimientos laterales se notan muy artificiales comparados con los humanos.

[1] Concepto definido en Apartado 2.2.2, página 22.

[2] El consumo del robot con todos los motores apagados, es decir, del PC incrustado y otros elementos electrónicos es de 0,95A.

6.2. Características de la locomoción con Sammy's Walk

La velocidad de locomoción con *Sammy's Walk* es de 6,7 cm/s. El consumo medio de intensidad es de 1,35A (Figura 6.2).

Restándole el consumo del PC i otros elementos queda que el consumo de intensidad utilizado en la locomoción es de 0,4A, y por tanto una potencia de 8,7W. Así, el valor de $c_{mt} = 2,93$. Esto supone una *reducción del coste específico de locomoción del 5 %*, aunque también una reducción de la velocidad de locomoción del 35 %.

Algoritmo	**Potencia** $[W]$	**Velocidad** $[cm/s]$	$\boldsymbol{c_{mt}}$
ALWalk	14,1 (100 %)	10,3 (100 %)	3,1 (100 %)
Sammy's Walk	8,7 (62 %)	6,7 (65 %)	2,93 (95 %)

Tabla 6.2: *Comparación algoritmos* ALWalk *y* Sammy's Walk

Recuperando la Tabla 2.1, se añaden los valores encontrados:

Robot	*"Cornell"*	*"Delft"*	*MIT*	*ASIMO*	*Humano*	*ZMP*	*Sammy's Walk*
$\mathbf{c_{mt}}$	0,055	0,08	0,02	1,5	0,05	3,1	2,93

Tabla 6.3: *Coste energético específico de la locomoción – Comparativa*

A nivel estilístico, aunque la locomoción parece más inestable que con ZMP, los movimientos se parecen más a los naturales de los humanos. Especialmente si se compara con un bebé (tienen aproximadamente las mismas dimensiones y peso), se pueden ver las analogías.

También es muy interesante comparar las trayectorias articulares del robot con el algoritmo *Sammy's Walk* (Figura 6.3, página 76) y las humanas (Figura 3.9, página 38). Se puede apreciar que las formas son muy parecidas, y las curvaturas son las mismas para cada fase de la marcha. De esta forma aprecia que el diseño del algoritmo inspirado en la locomoción humana efectivamente produce trayectorias similares.

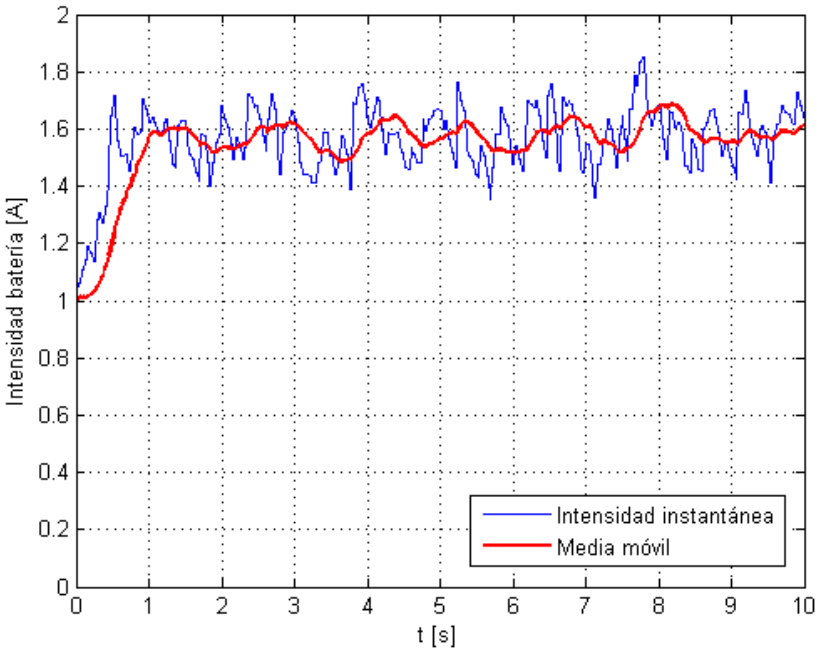


Figura 6.1: *Consumo de intensidad para la locomoción ZMP* low-stiffness

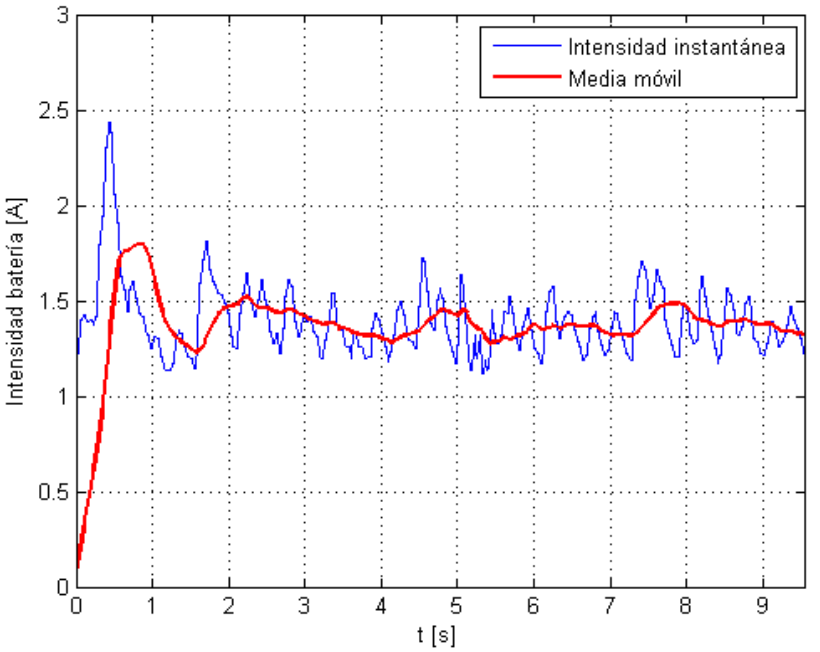
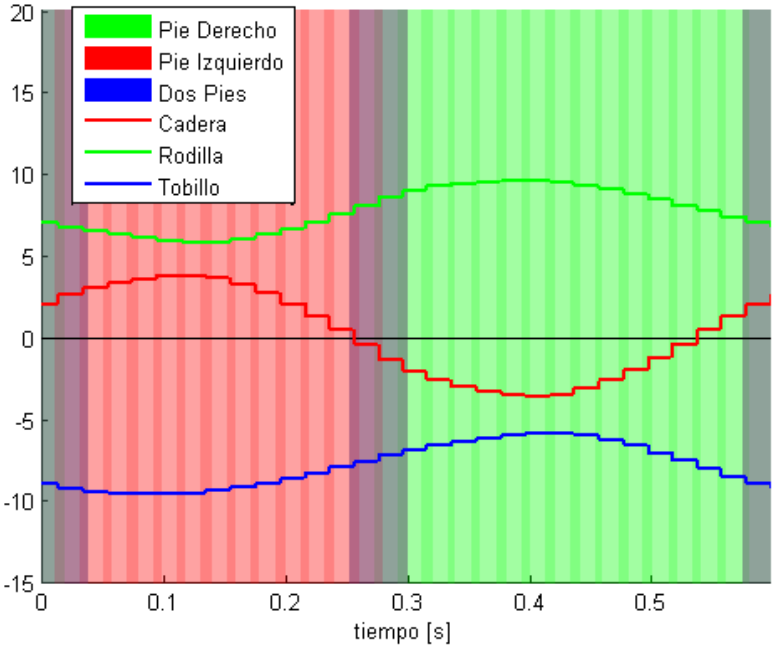
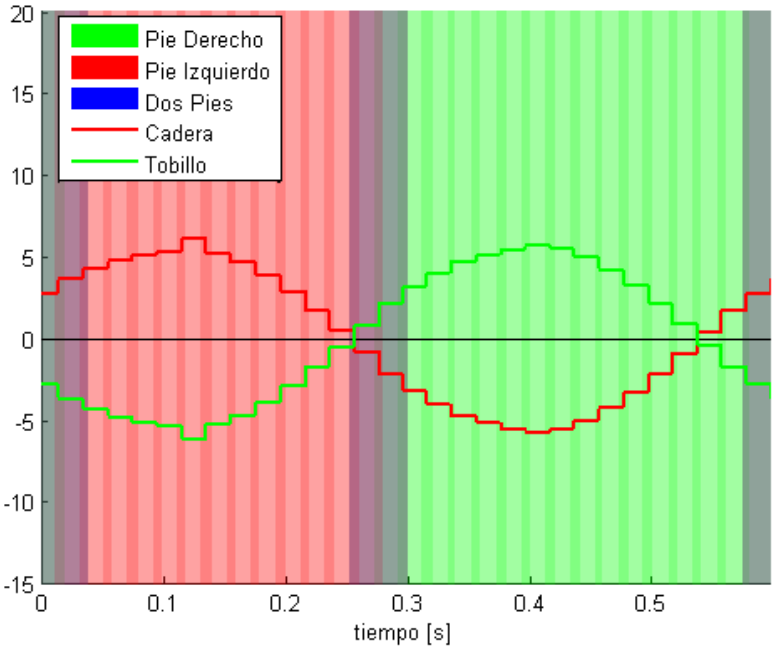


Figura 6.2: *Consumo de intensidad para la locomoción Sammy's Walk*



(a) *Ángulos eje Y de flexión/extensión*



(b) *Ángulos eje X de abd/adducción*

Figura 6.3: *Ángulos articulares de la pierna derecha y fase de paso*

Capítulo 7

Conclusiones

El objetivo principal de este proyecto era conseguir que el robot humanoide Nao caminara. En primer lugar se probó el algoritmo ZMP de locomoción propio de Nao, *ALWalk*, y se encontró que tal como viene implementado presenta diversos problemas (5.1.2). Por tanto, se ha realizado un rediseño del algoritmo *ALWalk* que los solucione (5.1.3). En primer lugar se ha disminuido la rigidez de las articulaciones para permitir movimientos más suaves, lo que ha permitido estabilizar el movimiento. Además se ha hecho que este sea más rápido para evitar que las oscilaciones propias interfieran, y más bien aprovecharlas. Así se ha conseguido que el algoritmo *ALWalk* tenga una locomoción estable. Sin embargo, su diseño está basado en una planificación de movimientos que no tiene en cuenta las dinámicas propias del robot.

El nuevo algoritmo de locomoción para Nao que se ha diseñado, *Sammy's Walk*, aprovecha las dinámicas propias del robot para obtener una locomoción más eficiente (5.2). La idea de partida es que la marcha es un ciclo límite en que se superponen una oscilación lateral con el movimiento de avance. Gracias a un estudio de las frecuencias propias de oscilación lateral se consiguió inducir un ciclo límite estable mediante una máquina de estados. Para aumentar las órbitas de atracción se han aplicado correcciones al movimiento de la pierna oscilante proporcionales a la desviación de la velocidad angular respecto al ciclo límite nominal.

Haciendo un análisis de prestaciones y de la eficiencia de ambos algoritmos de locomoción para el robot Nao se han obtenido los siguientes resultados:

- El algoritmo *AlWalk* tiene un consumo de potencia mecánica para la locomoción de $P_m = 14,1W$, una velocidad de translación de $v = 10,3cm/s$ y un coste específico (2.2.2) de $c_{mt} = 3,1$. El algoritmo *Sammy's Walk* consume para la locomoción $P_m = 8,7W$, una velocidad de translación de $v = 6,7cm/s$ y un coste específico de $c_{mt} = 2,93$.

Algoritmo	**Potencia** $[W]$	**Velocidad** $[cm/s]$	$\boldsymbol{c_{mt}}$
ALWalk	14,1 (100 %)	10,3 (100 %)	3,1 (100 %)
Sammy's Walk	8,7 (62 %)	6,7 (65 %)	2,93 (95 %)

Tabla 7.1: *Comparación algoritmos* ALWalk *y* Sammy's Walk

- Tal como se puede ver, **aprovechar las dinámicas propias del sistema para la locomoción bípeda provoca una disminución del coste específico de transporte del 5 %**. Esta reducción, aunque pequeña, es un indicio claro de que este es un buen camino, y sin duda es uno de los mejores logros de este proyecto.
- Además, tanto las trayectorias articulares como el movimiento conjunto resultan parecidos a los humanos (6.2).

Fruto del desarrollo de este proyecto se han obtenido los siguientes resultados paralelos:

- Creación de un conjunto de scripts que permiten la puesta en marcha de Nao: instalación, acceso al robot, compilación remota, compilación local y ejemplos de uso (A).
- Creación de una interfaz gráfica de usuario, NaoGUI que permite acceder a las funciones principales del robot, las funciones de locomoción desarrolladas en este proyecto, y control mediante el mando de la Wii (B.2).

Como en todo trabajo de investigación y aplicación tecnológica, a medida que se resuelven los problemas aparecen otros desafíos a solucionar. Así, de este proyecto se desprenden las siguientes líneas de trabajo de cara a investigaciones futuras:

- El coste específico de los *passive walkers* sigue siendo muy inferior a los obtenidos. Uno de los motivos más importantes es el diseño de los pies. El robot Nao no puede conseguir un movimiento de balanceo ya que los pies son una superficie plana, y este es un factor clave que hace que el coste específico aumente mucho. Por tanto, se podrían investigar otros mecanismos para conferirle al robot mayor estabilidad y eficiencia en su locomoción.
- No se ha conseguido estabilizar de forma eficaz el movimiento de avance. En el momento que se consiga este punto es muy posible que se pueda aumentar mucho la velocidad de translación manteniendo una potencia consumida parecida, con la consecuente disminución del coste específico de transporte.
- En el presente proyecto solamente se ha investigado sobre la locomoción en línea recta sobre una superficie plana. Posteriores estudios podrían aplicar los conceptos aquí expuestos para generar locomoción en curva, locomoción lateral y locomoción capaz de adaptarse a irregularidades del terreno.
- Añadir movimiento a los brazos para conseguir marchas más estables y suaves.

Como conclusión final, el algoritmo *Sammy's Walk* que se ha diseñado, basado en la inducción de un ciclo límite estable y aprovechamiento de las dinámicas propias del robot, ha conseguido una mejora en la eficiencia de la locomoción del robot humanoide Nao. Además tiene trayectorias articulares similares a las humanas, lo que de alguna forma deja entrever que es una vía de investigación que apunta en la dirección adecuada. Se podría decir que a medida que pase el tiempo los robots tendrán un papel más importante en nuestra sociedad. Su evolución, e integración, estará determinada en gran manera por su capacidad de interactuar con el entorno físico, y en concreto de moverse en él de una forma eficiente. Así, el desarrollo de algoritmos de locomoción basados en aprovechar las dinámicas propias puede constituir un punto clave en esta evolución, tal como se ha podido analizar en este proyecto.

Bibliografía

[1] *Els ordinadors*, Opcions nº 6, (2002), pp. 8–12.

[2] *Electronic waste guide*, http://www.e-waste.ch/.

[3] Aldebaran, *Documentación de Nao*, 2009.

[4] Carpin, S. et al., *The challenge of motion planning for soccer playing humanoid robots*, (2008).

[5] Castán Salinas, A., *Material informático y contaminación medioambiental*, Abril 2008.

[6] Collins et al., *Efficient Bipedal Robots Based on Passive-Dynamic Walkers*, Science, 307 (2005), pp. 1082–1085.

[7] Goswami, A. et al., *Limit cycles and their stability in a passive bipedal gait*, tech. report, INRIA Rhône-Alpes, 1996.

[8] Honda, *Asimo - technical information*, tech. report, Honda Motor Co., Ltd., 2007.

[9] Kulk, J., Welsh, J., *A low power walk for the nao robot*, tech. report, University of Newcastle, Australia, 2008.

[10] Kuo, A., *Stabilization of lateral motion in passive dynamic walking*, The International Journal of Robotics Research, (1999).

[11] Laszlo, J. et al., *Limit cycle control and its application to the animation of balancing and walking*, tech. report, Department of Computer Science and Department of Electrical and Computer Engineering, University of Toronto, 2005.

[12] Lee J., Ho Oh, J., *Biped walking pattern generation using reinforcement learning*, tech. report, Humanoid Robot Research Center, KAIST, Republic of Korea, 2007.

[13] Lelièvre, J., Lelièvre J-F., *Patología del pie*, Masson, 1987.

[14] Manoonpong P. et al., *Adaptive, fast walking in a biped robot under neuronal control and learning*, PLoS Comput Biol, (2007).

[15] McGeer, T., *Passive dynamic walking*, tech. report, Simon Fraser University, Burnaby, British Columbia, Canada, 1988.

[16] ——, *Stability and control of two-dimensional bipedal walking*, tech. report, Simon Fraser University, Burnaby, British Columbia, Canada, 1988.

[17] ——, *Passive bipedal running*, tech. report, Simon Fraser University, Burnaby, British Columbia, Canada, 1989.

[18] ——, *Passive walking with knees*, Proc. 1990 IEEE Robotics & Automation Conference, Cincinnati, OH, (1990), pp. 1640–1645.

[19] Morimoto, J. et al., *A simple reinforcement learning algorithm for biped walking*, International Conference on Robotis and Automation, (2004).

[20] Pratt, J., Pratt, G., *Intuitive control of a planar bipedal walking robot (proc. of the 1998 international conference on robotis and automation)*, tech. report, MIT Leg Laboratory, Cambridge, 1998.

[21] ——, *Virutal model control of a bipedal walking robot*, International Conference on Robotis and Automation, (2007).

[22] Raibert, M., *Bigdog, the rough-terrain quaduped robot*, (2008).

[23] Shuuji, K., *Humanoid robot hrp-2: Overview of zmp-based biped walking*, tech. report, DynamicWalking2008, Delft, Netherlands, 2008.

[24] Vukobratovic, M., Borovac, B., *Zero-moment point - thirty fiveve years of its life*, International Journal of Humanoid Robotics, 1 (2004), pp. 157–173.

[25] Westervelt, E.R., Grizzle, J.W., *Design of asymptotically stable walking for a 5-link planar biped walker via optimization*, tech. report, Department of Electrical Engineering and Computer Science, 2002.

[26] Williams, E., *It and environment initiative*, http://www.it-environment.org/.

[27] Yin, K. et al., *SIMBICON: simple biped locomotion control*, ACM, New York, NY, USA, 2007.

Apéndice A

Puesta en marcha y programación de Nao

A.1. Instalación del robot

El robot se arranca apretando el botón del pecho, y se apaga manteniéndolo pulsado durante 4s. Una vez el robot ha arrancado las luces de los ojos dejan de parpadear y "saluda" con un mensaje de bienvenida. Si se da un pulso al botón del pecho el robot se presenta con su nombre y número de serie, así como la IP a la que se ha conectado. Es muy importante establecer la conexión correctamente para poder comunicarse con el robot. Esta debe estar físicamente lista antes de encender Nao.

Se ha elegido como sistema de desarrollo Linux, por la facilidad de configuración de las tareas necesarias, y por ser el único que permite compilación de librerías dinámicas, tal como se explicará posteriormente. Todas las explicaciones se realizarán suponiendo este entorno. En concreto, para el desarrollo de este proyecto se ha usado la distribución *Ubuntu 9.04* de 32bits. Para el desarrollo en otros sistemas se puede consultar la documentación de Nao.

A.1.1. Conexión Ethernet

En primer lugar hay que asegurarse que la conexión está físicamente realizada:

1. Router conectado a la alimentación.
2. PC conectado con el cable de red al router.
3. Nao conectado en la cabeza con el cable de red al router.

Antes de encender el robot hay que configurar la conexión, y ejecutar un servidor DHCP[1] que asignará la IP al robot automáticamente. En primer lugar hay que establecer una IP al PC. Suponiendo que la red ethernet se llama `eth0`, desde una consola se realiza de la siguiente manera:

```
sudo ifconfig eth0 10.0.0.1
sudo /etc/init.d/dhcp3-server start
```

[1] *Dynamic Host Configuration Protocol*, Protocolo Configuración Dinámica de Servidor.

Ahora ya se puede encender el robot. Una vez encendido, apretando el botón del pecho se reproduce un mensaje con la IP de Nao. También se puede buscar la IP asignada mediante el programa `nmap`, que hace un escaneo de las IP activas. Si la salida del programa se filtra se puede obtener la IP del robot (en caso que en la red haya más de una IP activa habrá que seleccionar la correcta):

```
sudo nmap -v -sP --system-dns 10.0.0.0/24 | grep "latency"
```

A partir de este momento se supondrá que la IP del robot es 10.0.0.8. Para ejecutar comandos *dentro* del Linux del Nao se puede hacer con el protocolo SSH[2] que los redirige al robot de forma segura, es como abrir una consola de Linux en Nao:

```
user@user-PC:~$ ssh root@10.0.0.8
root@Nao [0] [~]#
```

Para salir basta con ejecutar la orden `exit`:

```
root@Nao [0] [~]# exit
Connection to 10.0.0.8 closed.
```

Si desde cualquier navegador se introduce la IP de nao se accede a su configuración. En la Figura A.1 se muestra la pantalla que aparece.

También se puede navegar por el sistema de archivos del robot mediante un protocolo SSH. Esto es muy fácil de hacer desde Linux. Desde el menú `Lugares->Conectar con servidor`, se introduce la IP del robot, `root` como usuario y listo. Creará una carpeta que contendrá la raíz del sistema de archivos de Nao llamada `sftp://root@10.0.0.8/`. En la Figura A.2 se muestra cómo hacerlo.

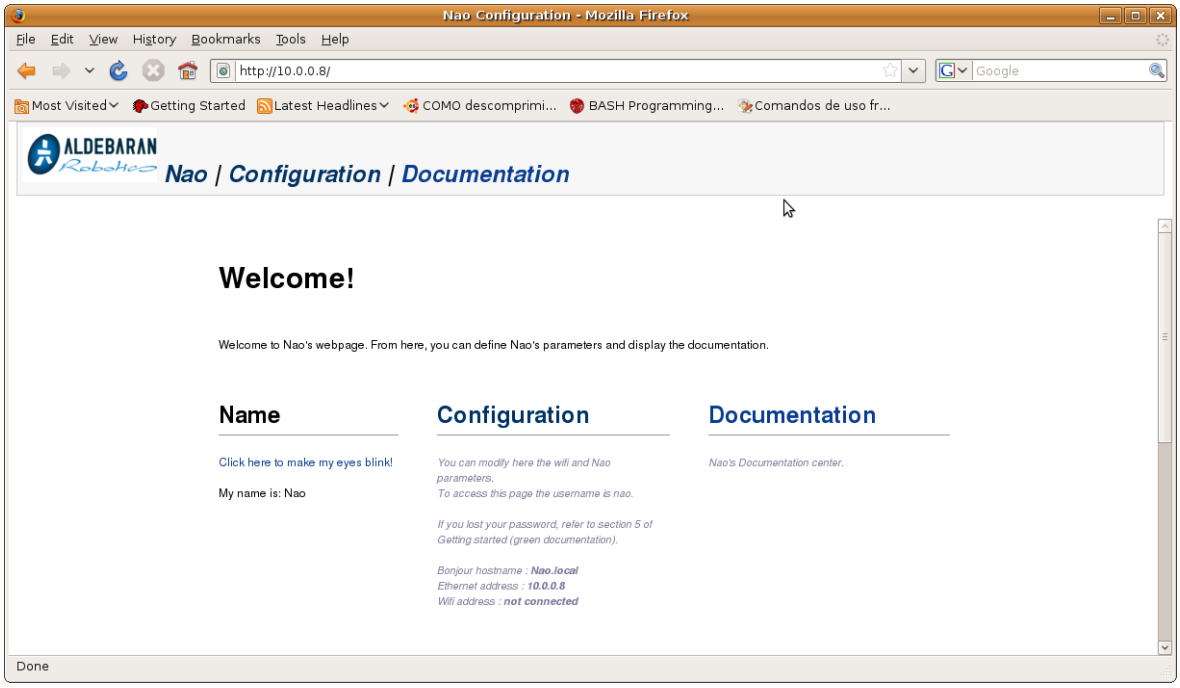
Para apagar el robot a se puede hacer manteniendo pulsado el botón del pecho más de 4s, o enviando la orden `halt` a través de SSH.

A.2. Instalación SDK

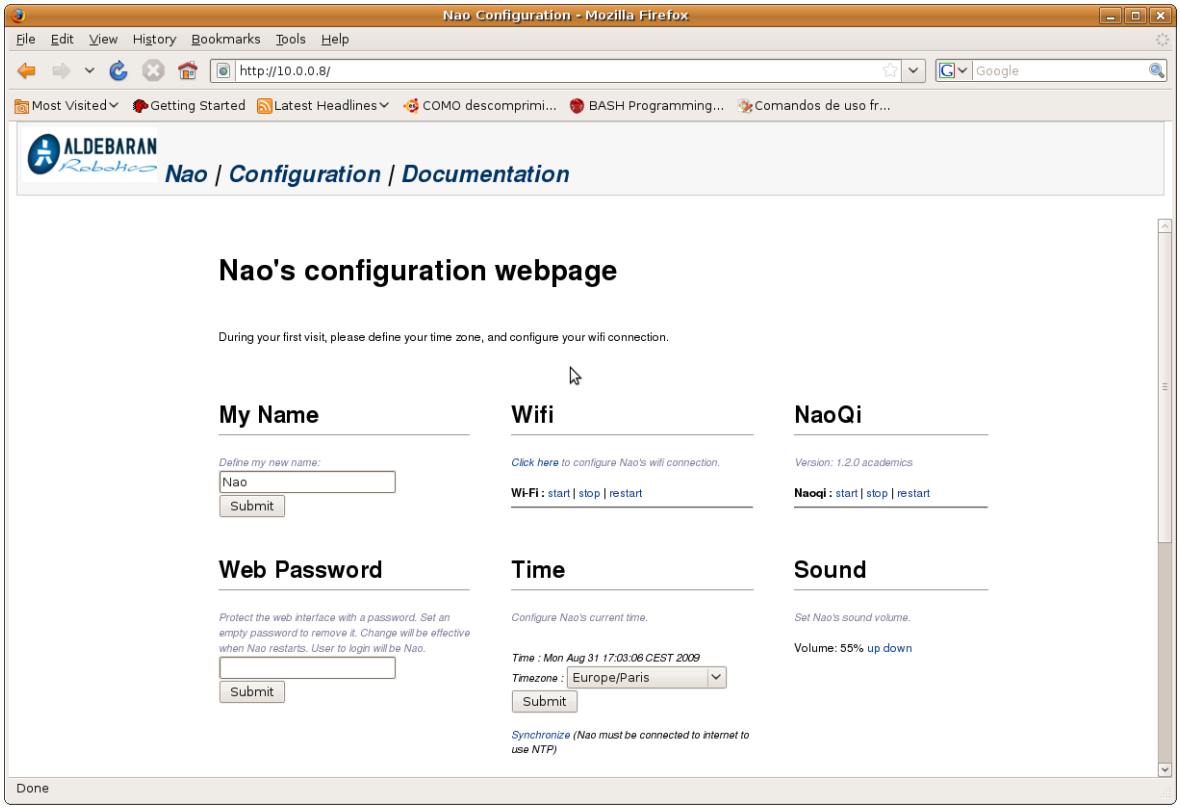
El SDK[3] proporcionado por Albedaran incluye los siguientes archivos:

- `NaoQiAcademics-1.2.0-Linux.tar.gz` Incluye los archivos necesarios para el desarrollo de programas: ejecutables, librerías, módulos, ejemplos... Este entorno de desarrollo se llama NaoQi (se explicará más adelante). Se puede utilizar con el programa `Choregraphe`, el suministrado por Aldebaran para desarrollo de programas de alto nivel, así como con otros simuladores. Nao internamente ejecuta NaoQi para su control, este SDK es el mismo NaoQi para ejecución remota en un PC. En la Figura A.3 se muestra el contenido del archivo.
- `ctc-academics-1.2.1.tar.bz2` Es la herramienta `CrossTool` que permite compilar librerías dinámicas para luego ser ejecutadas dentro de Nao. Su uso se explicará posteriormente.

[2] *Secure SHell*, intérprete de órdenes seguro.
[3] *Software Development Kit*, kit de desarrollo de software.



(a) *Página principal*



(b) *Página de configuración*

Figura A.1: *Página web de conexión a Nao*

Figura A.2: *Configuración de servidor en Nao*

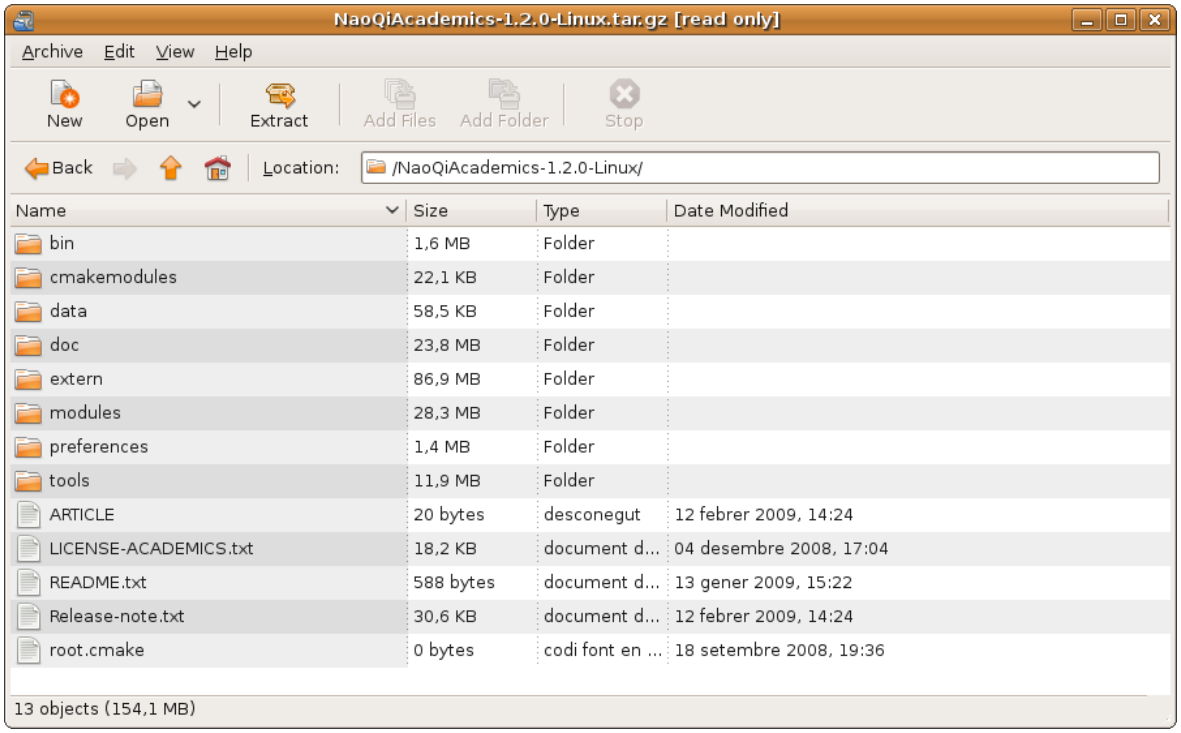


Figura A.3: *Contenido de NaoQiAcademics-1.2.0-Linux.tar.gz*

Con el siguiente *script* (se supone ubicado en el mismo directorio que los archivos mencionados) se puede instalar en la carpeta de trabajo ∼/naoqi. También realiza las configuraciones necesarias. En concreto es necesario definir una variable del sistema llamada AL_DIR que contenga la ruta de la instalación de NaoQi.

```
#!/bin/bash
if [ "$AL_DIR" = "" ]; then

    echo "Instalando Naoqi en ~/naoqi/"
    tar -xzvf NaoQiAcademics-1.2.0-Linux.tar.gz
    mv NaoQiAcademics-1.2.0-Linux ~/naoqi

    echo "Instalando CrossTool en ~/naoqi/crosstoolchain
       "
    bzip2 -dc ctc-academics-1.2.1.tar.bz2 | tar -xv
    mv ctc-academics-1.2.1 ~/naoqi/crosstoolchain

    echo "Creando variable AL_DIR. Se añade a bashrc"
    echo "export AL_DIR=~/naoqi" | tee -a ~/.bashrc

    echo "Instalando paquetes necesarios para el
       desarrollo"
    sudo apt-get install gcc g++ cmake python python-dev
else
    echo "Existe instalacion anterior de naoqi."
    echo "Borre todos los archivos y elimine la variable
        AL_DIR"
fi
```

A.3. Programación

Nao se puede programar usando diversos lenguajes: C, C++, Pyhton y URBI. Python es un lenguaje interpretado (no requiere compilación) novedoso y flexible, que permite crear *scripts* complejos para interactuar con los módulo de NaoQi. URBI es un lenguaje específico de robots. En los siguientes apartados se explicarán más detalles.

A.3.1. NaoQi

NaoQi es el entorno de desarrollo de Nao, se proporciona conjuntamente con el robot. En si es un programa que se ejecuta en Linux y que hace de intermediario entre el robot y los programas. Después que el sistema operativo del robot ha arrancado se ejecuta directamente NaoQi.

NaoQi se puede ejecutar local o remotamente. Por localmente se entiende *dentro del robot*, es decir, ejecutado en el linux incrustado de Nao. Para facilitar el desarrollo de aplicaciones, también se puede ejecutar NaoQi *desde el PC*, remotamente. Como es lógico, los comandos

que se envíen no tendrán ninguna actuación ya que no existe ningún robot. Aun así, esto es una herramienta grandiosa, ya que se puede usar con simuladores como Webots, o con el programa Coreographe facilitado por Aldebaran. También permite la compilación de programas ya que incorpora todas las librerías necesarias para crear programas propios.

A.3.2. Características y uso de NaoQi

- Multi-lenguaje: se puede programar en diversos lenguajes.
- Ejecución de procedimientos secuencialmente, en paralelo, o llamadas por eventos.
- Modularidad: se pueden ejecutar programas en local para aplicaciones que requieran control con un bajo tiempo de ciclo, o remotamente mediante llamadas a los módulos de NaoQi, para aplicaciones que no tengan estas requisiciones.
- Multi-plataforma: al tener un sistema operativo incrustado con comunicaciones vía Ethernet (o Wi-Fi), las llamadas pueden realizarse desde cualquier sistema operativo.
- Uso de la memoria compartida: se puede leer, escribir o subscribirse a datos.

A continuación se explicará la estructura de NaoQi. Hay unos programas llamados *brokers* que escuchan comandos en una dirección IP en un puerto determinado. NaoQi es el *broker* principal *(main broker)*, que escucha en la dirección del robot (o la dirección local en caso que NaoQi se ejecute localmente), y por defecto en el puerto 9559.

Un módulo es una clase que contiene funcionalidades del robot (movimientos, leds...). Los módulos son de hecho librerías cargadas desde `AL_DIR/modules/lib/autoload.ini`. Cuando NaoQi se carga (u otro *broker*), se inicializan automáticamente sus módulos asociandos.

Finalmente, hace falta enviar los comandos a los módulos. Esto se hace mediante *proxys*, esto es, accesos a módulos.

Desde el punto de vista del programador, para ejecutar comandos sobre el robot hay que ejecutar los siguientes pasos:

1. Ejecutar un *broker*. La forma más fácil es utilizar el *broker* principal NaoQi, ya sea local o remotamente.
2. En la configuración del *broker* previamente se han indicado los módulos que se quieren cargar.
3. Crear un *proxy*, y mediante este enviar las órdenes al módulo correspondiente, es decir, al robot.

A.3.3. Módulos de NaoQi

Los módulos principales facilitados por NaoQi son:

ALMemory Es la memoria del robot. Permite acceder y subscribirse con funciones como `getData` o `subscribeOnDataChange`.

ALMotion Permite enviar comandos de posición a las articulaciones con funciones como `setAngle`, ya sea a una articulación o una cadena. Incorpora funciones de alto nivel para conseguir las matrices de cinemática directa. También se puede controlar directamente la posición del centro de masa, equilibrio con un único pie o con dos, consignas de posición en el espacio cartesiano. Además, tiene funciones de alto nivel para caminar que usan el algoritmo ZMP. Es tan fácil como enviar comandos tipo `walkStraight(distancia)`.

ALLeds Actuación sobre los LEDs.

DCM Siglas de *Device Communication Manager*, es el módulo que se encarga de las comunicaciones con los dispositivos del robot (placas, sensores, actuadores...). Así, es el vínculo entre las funciones de alto nivel y las de bajo nivel (programas en las placas electrónicas). Mediante este módulo se pueden enviar comandos directamente a los actuadores.

ALUltrasound Se encarga del control de los sensores de ultrasonidos.

ALAudioPlayer Permite reproducir archivos de audio .wav o .mp3 con comandos como `aup.playFile("/freemp3/feelgood.mp3")`.

ALTextToSpeech Sintetizador de texto en inglés, reproduce palabras por los altavoces con comandos como `tts.say("Hello World!")`

A.3.4. Ejemplo de programación en Pyhton

Suponiendo que se ha creado la conexión ethernet o Wi-Fi PC-robot correctamente, que Nao tiene una IP correcta y que el robot y NaoQi está en funcionamiento, el siguiente sería un código ejemplo para conectarse al *main broker* y hacer algunas cosas:

```
1  # -*- coding: utf-8 -*-
2  import os
3  import sys
4  import time
5
6  #Conexión con NaoQi
7  aldir = os.environ.get("AL_DIR")
8  home = os.environ.get("HOME")
9
10 # import NaoQi lib
11 if aldir == "None":
12     print("La variable de entorno AL_DIR no está definida")
13     sys.exit(1)
14 else:
15     aldir = aldir + "/extern/python/aldebaran"
16     aldir = aldir.replace("~", home)
17     aldir = aldir.replace("'", "")
18     sys.path.append(aldir)
19
```

```
20      from naoqi import ALBroker
21      from naoqi import ALModule
22      from naoqi import ALProxy
23      from naoqi import ALBehavior
24
25  IP = "10.0.0.8" #Insertar aquí la IP
26  PORT = 9559 #Puerto por defecto de NaoQi
27
28  try:
29      broker = ALBroker("pythonBroker","127.0.0.1",9999,IP,
            PORT)
30  except RuntimeError,e:
31      print("Imposible establecer conexión con broker")
32      exit(1)
33
34  try:
35      motion = ALProxy("ALMotion")
36      print("ALMotion")
37  except(RuntimeError,e):
38      print("Error creando vínculo con módulo ALMotion")
39      print(str(e))
40      exit(1)
41
42  try:
43      dcm = ALProxy("DCM")
44      print("DCM")
45  except(RuntimeError,e):
46      print("Error creando vínculo con módulo DCM")
47      print(str(e))
48      exit(1)
49
50  try:
51      leds = ALProxy("ALLeds")
52      print("ALLeds")
53  except(RuntimeError,e):
54      print("Error creando vínculo con módulo ALLeds")
55      print(str(e))
56      exit(1)
57
58  try:
59      tts = ALProxy("ALTextToSpeech")
60      print("ALTextToSpeech")
61  except(RuntimeError,e):
62      print("Error creando vínculo con módulo ALTextToSpeech")
63      print(str(e))
64      exit(1)
65
```

```
66
67 leds.set("FaceLeds",1.0)
68
69 #Establece la stiffness (rigidez de las articulaciones) de
       todo el cuerpo a 1 (, usando una interpolación que dura %)
        1s
70 motion.gotoBodyStiffness(1.0 , 1.0, motion.
       INTERPOLATION_SMOOTH)
71
72 tts.say("Hi. I am going to walk. To stop me move my head.")
       #Habla
73
74 motion.addWalkStraight(10, 80) #Se envian las órdenes para
       caminar
75 walkTaskId = motion.post.walk() #Se inicia la locomoción
       como una tarea (se ejecuta en paralelo, sigue el hilo de
       ejecución)
76
77 previousHeadAngle = motion.getAngle("HeadYaw");
78 motion.setChainStiffness("Head",0.0)
79
80 time.sleep(2)
81
82 while (motion.isRunning(walkTaskId)):
83     #Si se mueve la cabeza más de
84     if(abs(motion.getAngle("HeadYaw") - previousHeadAngle) >
           0.1):
85         motion.clearFootsteps() #Se finaliza el movimiento
86     time.sleep(1)
87
88 tts.say("Bye Bye")
```

Solamente es necesario guardar este código en un archivo de Python, por ejemplo `prueba.py` y llamar al intérprete para ejecutar los comandos deseados. Desde una consola de Linux se llamaría de la siguiente forma:

```
python prueba.py
```

El resultado se puede ver en el vídeo [A.3.4 Ejemplo de programación en Python].

Junto con el pack de desarrollo se incluyen numerosos ejemplos de programación en todos los lenguajes.

A.3.5. URBI

URBI *(Universal Real-time Behavior Interface)* es un lenguaje de programación usado para el control de robots, desarrollado por la empresa francesa *Gostai* (www.gostai.com).

Hay distribuciones para Windows, Linux y Mac. Es un lenguaje cuyo núcleo es de bajo nivel, diseñado para trabajar con motores y sensores, aunque permite el desarrollo de comandos complejos de alto nivel. En su diseño se ha cuidado la simplicidad, y permite la interconexión con otros lenguajes de programación, como C++, Java, Python, Matlab y muchos otros. Como puntos clave destacan la capacidad de ejecución de comandos en paralelo, secuencialmente sin retraso, manejo de eventos de forma muy simple y uso del tiempo en condiciones. Por ejemplo, se puede usar el operador | para que una órden se ejecute exactamente después de otra; el operador & para que dos órdenes empiezen al mismo tiempo. Las órdenes a las articulaciones se pueden enviar directamente, imponiendo un tiempo de interpolación, una velocidad e incluso un movimiento sinusoidal. Los objetos tienen un alto grado de abstracción y métodos muy intuitivos.

A continuación se muestran algunos comandos válidos en URBI para programar Nao, es muy sencillo de entender aunque no se conozca el lenguaje:

```
1  motors.on(); //un ; al final de la sentencia obliga a
       esperar a que esta acabe
2
3  //Se pueden importar y usar fácilmente objetos de NaoQi
4  var tts = ALProxy( "ALTextToSpeech" );
5  tts.say( "Youhou!" );
6
7  { ( headPitch.val = -0.6 time:1s ) & ( headYaw.val = 0.6
       time:1s ) } | { ( headPitch.val = 0.0 time:3s ) | (
       headYaw.val = 0.0 time:3s ) } ;
8  //En primer lugar mueve headPitch a -0.6 rad y headYaw a 0.6
       rad, ambos movimientos tienen duración de 1s, y empiezan
       al mismo tiempo ya que se ha usado el operador &
9  //Justo después que ha acabado este movimiento (operador |),
       se mueve headPitch a 0.0 en 3s y después headYaw a 0.0
10
11 movimiento: //Definición de una etiqueta
12 {//Movimientos sinusoidales
13 //el operador , permite separar órdenes, que se ejecutarán
       cuando llegue el operador ;
14     headYaw.val = 0 sin:3s ampli:1,
15     headPitch.val = 0 cos:3s ampli:0.6,
16 };
17
18 wait( 3s );
19 stop movimiento;
```

Cuando se enciende el robot, NaoQi ejecuta un broker intérprete de URBI (si está habilitado en `AL_DIR/modules/lib/autoload.ini`) que escucha en el puerto 54000. Para enviar comandos, por ejemplo los que se han explicado antes, se puede hacer de forma tan sencilla como abrir una consola de Linux y enviarlos mediante el programa telnet:

```
telnet 10.0.0.8 5400
```

El intérprete de URBI ya está configurado, los sensores y actuadores se declaran automáticamente. Para saber el nombre se puede mirar el archivo[4]:

```
sftp://root@10.0.0.8/opt/naoqi/extern/urbi/URBI.INI
```

Simplemente con ejecutar el terminal telnet y escribir el script mencionado anteriormente el robot realiza los movimientos que se pueden ver en el vídeo [A.3.5 Ejemplo de programación en URBI]:

A.3.6. Ejecución de código en tiempo real

La plataforma de desarrollo de Nao tiene herramientas para la implementación de código en tiempo real. En caso que simplemente se quiera ejecutar un bloque de código de forma periódica existe una metodología más fácil.

El módulo DCM, encargado de enviar los comandos a los actuadores y actualizar el valor de los sensores a las variables de memoria tiene un período de refresco de 20ms (50hz). Las consignas de posición se pueden enviar a través del DCM para instantes futuros arbitrarios, y será el propio DCM que se encargará de hacer la interpolación necesaria. En cada instante que se ejecuta el código del DCM se analizan las consignas futuras y, con la posición actual, se hace la planificación necesaria para cada articulación. Por debajo del DCM existen las placas electrónicas que tienen una frecuencia de refresco mucho más alta (para los motores 1ms) que se encargan de controlar los dispositivos directamente.

Es decir, que desde el punto de vista del programador, a este nivel sólo se puede leer y enviar información con una frecuencia de 50hz. La documentación da a entender que la actualización de los sensores y el envío de consignas a los actuadores se realiza de forma prácticamente instantánea, tampoco se da información precisa al respecto.

En caso de ejecutar programas de forma local (dentro el robot) existe la posibilidad de sincronizarse con la actualización del DCM. Es decir, será el propio DCM que llame a una función determinada que se desee. Por tanto, se puede crear una función que incorpore el código de control, y al sincronizarla con DCM se ejecutará cada 20ms. La duración del código debe ser tal que no colapse el sistema, por tanto es recomendable hacer los menores accesos a memoria i escritura en ficheros.

También cabe destacar el hecho que para poder ejecutar código en tiempo real es necesario que el programa esté compilado (no se podría hacer con Python o URBI, C++ es el idóneo) como una librería dinámica, es decir, un módulo. Esto quiere decir que será el propio NaoQi *(broker)* que la llame al arrancarse si está habilitada en el archivo `autoload.ini`. El siguiente apartado explica cómo se compila.

A.3.7. Compilación cruzada para librerías dinámicas

Para poder compilar una librería dinámica es necesario hacerlo en entorno Linux. Ya que el sistema operativo de Nao es un Linux, hay que compilar el archivo en el mismo entorno. Los

[4]Este archivo está dentro de Nao. Se accede mediante un acceso SSH.

archivos necesarios deben estar en la ruta ~/naoqi/modules/src/[Nombre del programa], con la estructura y los archivos de configuración necesarios para ser compilado por cmake. Lo más práctico resulta copiar la estructura del directorio ~/naoqi/modules/src/examples.

El SDK de NaoQi ya trae ejemplos de programas en C++, así como los archivos de configuración y las dependencias para hacer una compilación rápida y limpia. El archivo generado (una librería .a) se copia al robot vía ethernet. A continuación solo hace falta reiniciar NaoQi y se cargará la librería programada. Finalmente hace falta llamar a las funciones deseadas del módulo de la forma deseada, por ejemplo con Python.

En la documentación explica paso por paso como realizar este tipo de compilación. También se podría compilar como un ejecutable que se conecte a un *broker* ya creado, pero para realizar esta tarea quizá sea más fácil escribir directamente un script en Python. A continuación se muestra un script de consola que permite compilar el ejemplo incluido en el SDK. Se supone que este script se ejecuta en la ruta ~/naoqi/modules/src/[Raíz del proyecto], y el código fuente está en ~/naoqi/modules/src/[Raíz del proyecto]/[Programa]/src.

```
1  #!/bin/bash
2  echo "Borrando archivos antiguos"
3  echo ""
4  rm -r crossbuild
5  mkdir crossbuild
6  cd crossbuild
7
8  echo "Llamando herramienta CrossTool"
9  echo ""
10 $AL_DIR/tools/crosscompile.sh $AL_DIR/crosstoolchain $AL_DIR
      /modules/src/NaoqiProgram > temp
11
12 echo ""
13 echo "Ahora se cargará CMAKE"
14 echo ""
15 echo "Se compilará como una librería, por tanto opción ..
      REMOTE=OFF"
16 echo "Así se podrá ejecutar código de tiempo real"
17 echo "Presione <c> y <g> para configurarlo."
18 echo ""
19 echo "Presione <enter> para continuar"
20 echo ""
21 echo ""
22 read DUMMY
23 ccmake .
24
25 echo "Compilando con make..."
26 echo ""
27 make > temp
28 if [ "$(cat temp | grep 'Built target')" == "" ]; then
29     cat temp
```

```
30      echo "Errores en compilación. Abortando"
31      echo ""
32      rm temp
33  else
34      rm temp
35      echo "Compilación exitosa"
36      echo ""
37      cd ..
38
39      echo "IP usada 10.0.0.8, para cambiarla edite este
           programa"
40      echo ""
41
42      echo "\Copiando la librería compilada al robot"
43      echo ""
44      scp $AL_DIR/modules/lib/libNaoqiProgram.so root@10
           .0.0.8:/opt/naoqi/modules/lib
45
46      echo "Se supone que la librería está añadida a 'autoload
           .ini'. Si no lo está, edite el fichero con"
47      echo ""
48      echo "\gedit    sftp://root@10.0.0.8/opt/naoqi/modules/
           lib/autoload.ini"
49      echo ""
50      echo ""
51
52      echo "Reiniciando naoqi"
53      echo ""
54      echo ""
55      ssh root@10.0.0.8 "/opt/naoqi/bin/restart.sh; exit"
56
57      echo "\Presione <enter> para ejecutar el programa en Nao
           "
58      echo ""
59      read DUMMY
60      ./test #Programa Python que se conecta al broker y
           ejecuta las funciones del módulo
61  fi
```

Apéndice B

Programación del algoritmo y la interfaz gráfica

En el Apéndice A.3.1 se explicó el funcionamiento de NaoQi, el paquete de desarrollo para Nao. Como se ejecutará código en tiempo real, será necesario compilarlo como una librería dinámica (módulo) dentro de Nao de forma local. A esta librería, que llevará implementado el algoritmo explicado se la llamará `ALSammysWalk`. Remotamente, desde el PC, correrá el programa que será la interfaz con el usuario llamada `NaoGUI`[1] y permitirá acceso a funciones útiles del robot, además de al algoritmo diseñado. Aunque con Nao se distribuye *Choregraphe*, una interfaz de desarrollo para Nao, es muy pesada, y un poco impráctica de usar. Por este motivo se plantea el desarrollo de una interfaz de diseño totalmente propio que sirva como herramienta de trabajo, y que permita futuras mejoras. Esta se proporciona libremente con este proyecto con amplios comentarios para futuras ampliaciones. La estructura del programa es la representada en la Figura B.1.

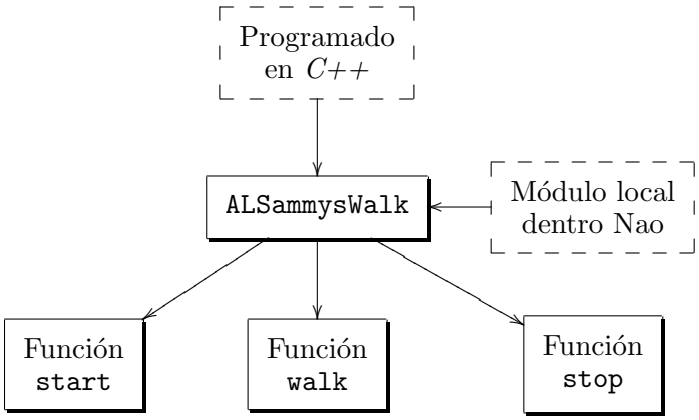


Figura B.1: *Estructura del programa local*

`ALSammysWalk` será un módulo de Nao encargado del control del algoritmo de locomoción diseñado. Desde el punto de vista externo tendrá disponibles funciones para interactuar tales como `start` para inicializar la locomoción, `walk` para caminar y `stop` para parar. Además, se programará de tal forma que sea posible cargar diferentes estilos de locomoción preprogramados mediante la función `style`. Internamente tendrá funciones privadas encargadas entre otros aspectos de: sincronización del bucle de control, generación de trayectorias (filtro pasabajos), detección de colisión de los pies, telemetría, etc. Se programará en *C++*, ya que es el lenguaje estándar de NaoQui para programar un módulo.

[1] *GUI Graphical User Interface*, Interfaz Gráfica de Usuario.

`NaoGUI` será una interfaz gráfica para interactuar con Nao. Accederá a diversos módulos para la actuación sobre Nao y para recibir información. Los módulos principalmente usados son: `ALMotion`, `DCM`, `ALWalk`, `ALTextToSpeech`, `ALLeds`, `ALSammysWalk`. Desde la ventana principal se accederá a las funciones de actuación sobre los leds de los ojos, reproducción de texto y caminar. Se podrá acceder a otras ventanas, una de ellas será un editor de posturas y configurador de *stiffness* articular, otra será destinada al control del algoritmo diseñado `ALSammysWalk`. Se programará en *Python* por su facilidad de uso y portabilidad.

En la explicación de este apartado se presuponen los conocimientos de *C++* y *Python*, y se dirige al lector a libros convenientes en caso de necesitar más detalles. Aun así, las sentencias son muy intuitivas, muchas de ellas se pueden entender fácilmente sin necesidad de dominar los lenguajes de programación citados.

B.1. Programación de *Sammy's Walk*

Este módulo se programará y compilará en el PC y posteriormente se cargará en Nao. Para más detalles, se puede consultar la documentación de Nao.

Los archivos que se generarán son `ALSammysWalk.cpp` (más de 900 líneas de código), que incluirá todo el código, y `ALSammysWalk.h`, con las cabeceras necesarias. La compilación se realiza con la herramienta `cmake`.

La siguiente exposición explicará el funcionamiento todo el programa, entrando en detalle sólo en los puntos clave. Gran parte de la exposición se hará sobre el propio código con comentarios.

B.1.1. Principales variables usadas

Para la máquina de estados, y en general para el control de las articulaciones, se usan una serie de variables, que se llamarán *variables articulares*. Éstas son el valor final que se quiere para cada articulación expresadas en grados sexagesimales. El interpolador las toma para generar los valores articulares a través del filtro pasa-bajos.

```
1  //Variables que almacenan el valor final deseado de cada
       articulación
2  //Son las variables que usa el interpolador
3  //Right Leg (0 to 4)
4  float RHipPitch, RHipRoll, RKneePitch, RAnklePitch,
       RAnkleRoll;
5  //Left Leg (5 to 9)
6  float LHipPitch, LHipRoll, LKneePitch, LAnklePitch,
       LAnkleRoll;
7  //Hip (10)
8  float LHipYawPitch;
9  //Right Arm (11 to 15)
10 float RShoulderRoll, RShoulderPitch, RElbowRoll, RElbowYaw,
       RWristYaw;
```

```
11 //Left Arm (16 to 20)
12 float LShoulderRoll, LShoulderPitch, LElbowRoll, LElbowYaw,
       LWristYaw;
```

Como hace falta un histórico de estas variables para el filtro, se crea otra variable que las incluirá todas. Las de arriba se usan por el programa para mayor facilidad, la siguiente será la que realmente se use.

```
1 //variables usadas por el interpolador de trayectorias para
      su filtro
2 float servo[21][2];
```

La máquina simétrica requiere tres estados, y para cada estado la configuración final y el tiempo de transición. Las siguientes variables, que contienen la representación del estado y se llamarán *variables de estado actual*, son las que de forma sucesiva se irán asignando a las previamente citadas para así inducir el movimiento al robot.

```
1 int state; //Estado actual
2 int tinc[3]; //Tiempo de transición
3 float SwingHipPitch[3], SwingHipRoll[3]; //Posturas
4 float SwingKneePitch[3];
5 float SwingAnklePitch[3], SwingAnkleRoll[3];
6 float StanceHipPitch[3], StanceHipRoll[3];
7 float StanceKneePitch[3];
8 float StanceAnklePitch[3], StanceAnkleRoll[3];
```

Además de las citadas, hay variables para la configuración del parámetro a del interpolador (un valor para los movimientos laterales y otro para los frontales), acceso a ficheros, *proxys* de los módulos de Naoqi, y otras variables secundarias.

B.1.2. Inicialización y sincronización con DCM

La función `start` es la encargada de iniciar el algoritmo. Se hacen las inicializaciones de las variables del sistema necesarias. Para evitar cambios bruscos en las articulaciones, al inicio se leen los valores de cada articulación y se establecen como valores del interpolador.

A continuación se leen los archivos de configuración. Estos están ubicados dentro del sistema de archivos de Nao, en la ruta `/data/[estilo]/`. Se definen dos estilos, `stand` es el primero que tiene como objetivo inducir la oscilación, y `walk` que es el que hace caminar al robot. Esta forma de acceder a las configuraciones permite desarrollos futuros, y la generación de nuevos estilos. Para cargar un estilo determinado basta con llamar a la función interna `loadConfig(const char * estilo)`. Esto se explicará con más detalle posteriormente.

Una vez hecho esto, se inicializa la máquina de estados, los temporizadores necesarios, se encienden los motores y se envía la posición inicial (robot de pie). También se abren dos ficheros, uno para "telemetría" y otro como un registro. Finalmente se inicia el lazo de control, subscribiéndose al DCM. Este se actualiza cada 20ms, y en cada actualización se ejecutará una función programada.

B.1.3. Bucle de control

El lazo de control se ejecuta cada 20ms. Consta de la máquina de estados pura, donde se realizan consecutivamente las siguientes acciones:

- Verificar condiciones externas. En caso que se apriete el botón del pie izquierdo se finaliza la ejecución. En caso que se apriete el del pie derecho se reinicia.
- Se define el estado 0, que sirve para hacer levantar al robot. El movimiento se realiza a lo largo de 3s. A partir de aquí se alterna sucesivamente de los estados 1 a 6, tal como se definió en la memoria.
- Para cada estado se realizan las siguientes acciones:
  - Se asignan a las *variables articulares* los valores de las variables de *estado actual*. En el caso de los estados 1 a 3 esta asignación es directa. En los estados 4 a 6 se toman los mismos valores cambiando la pierna oscilante por la fija, y cambiando el signo de las articulaciones laterales, ya que el movimiento es simétrico.
  - Se comprueba la condición de fin de estado. Para los estados 1, 2, 4, 5 es un tiempo determinado, que se calcula con temporizadores. Para los estados 3 y 6 es el impacto del pie con el suelo. La forma de detectarlo se explica posteriormente.
- En primer lugar se realizan dos pasos en el sitio, para dar tiempo a que la oscilación lateral se induzca. A partir de aquí se prosigue con la máquina de estados normal.
- Se añade la información de "telemetría" al archivo para su posterior análisis y el registro.
- Se llama al interpolador para que actualice los valores articulares.
- En caso que se apriete el botón del pie izquierdo, o se envíe la orden, se detendrá la ejecución del programa. Para hacerlo de forma segura se establece el estado -1, que llevará el robot a una posición de sentado de forma gradual, y finalmente se cancelará la subscripción al DCM y se cerrarán los archivos.

Es importante verificar que efectivamente el código se ejecute en tiempo real. Haciendo un análisis de activación (ver Figura B.2) se puede ver que el tiempo entre sucesivas activaciones es de $20 \pm 1ms$, por tanto muy correcto. La ejecución del bucle de control dura en su totalidad 1-2ms, un valor muy pequeño que asegura que el código programado se ejecuta en tiempo real.

B.1.4. Generador de trayectorias

El generador recorre las articulaciones una por una actualizando su valor. Tal como se ha explicado, el valor que se envía es el de las posturas (se puede interpretar como un escalón) filtrado a través de un filtro pasa-bajos. La función implementada, de ganancia unitaria, es:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{(1-a)^2 z}{(z-a)^2}, \quad a \in \mathbb{R}, \quad 0 < a < 1 \tag{B.1}$$

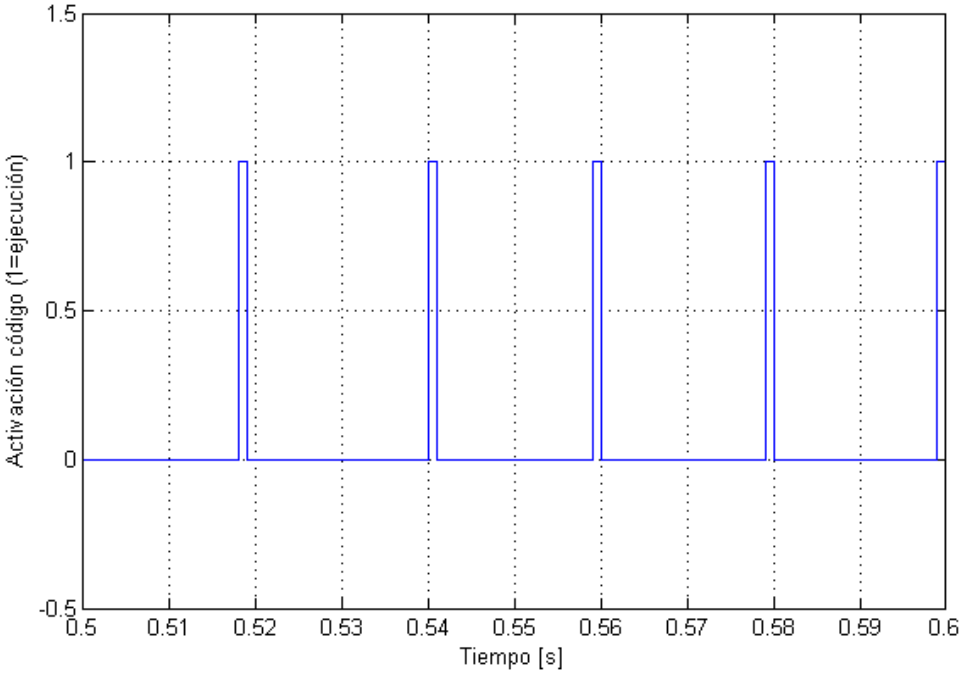


Figura B.2: *Análisis de ejecución del bucle de control*

siendo a el parámetro que controla la velocidad con que se llega al valor final. Haciendo la antitransformada z al dominio temporal se tiene que:

$$y_{k+1} = (1-a)^2 u_k + 2a y_k + a^2 y_{k-1} \tag{B.2}$$

Esta ecuación en diferencias es la que se ha implementado para cada articulación.

B.1.5. Detección de impacto con el suelo

Nao tiene en cada pie sensores capacitivos de fuerza que se pueden usar para detectar el impacto con el suelo. Estos sensores tienen una amplia variabilidad, y requieren ser calibrados individualmente. El valor que proporciona Naoqi de la lectura de los sensores tiene una característica inversa respecto a la fuerza ejercida. Haciendo una calibración individual de cada sensor se consigue un buen comportamiento.

Para saber si un pie ha impactado en el suelo se comprueban todos los sensores, y si alguno está por debajo de un umbral establecido (presionado) quiere decir que el pie ha colisionado con el suelo.

B.1.6. Carga de configuraciones

Para cargar y guardar las configuraciones se usan las funciones `loadConfig(const char * estilo)` y `saveConfig(const char * estilo)`, a las que se pasa como parámetro una ca-

dena con el nombre del estilo que contiene la configuración (de momento sólo `stand` y `walk`). Dentro de la carpeta de cada estilo se guarda en orden la siguiente información:

- $aRoll$: parámetro del interpolador para el movimiento lateral.
- $aPitch$: parámetro del interpolador para el movimiento frontal.
- $t1$: duración de los estados 1 y 4 de la máquina de estados.
- $t2$: duración de los estados 2 y 5 de la máquina de estados.
- k_{vel}: ganancia de la compensación del desvío de velocidad angular.

B.2. GUI

Se ha programado una interfaz gráfica de usuario, NaoGUI, que servirá para acceder de forma fácil a las funciones principales de Nao, y en concreto se diseñará una interfaz para usar el módulo de locomoción propio de Nao *ALWalk* y el diseñado *Sammy's Walk*. También se incluirá la funcionalidad adicional de controlar el robot Nao a través del mando inalámbrico de la Wii, el Wiimote.

Se ha programado en Python, por su flexibilidad y porque permite acceder a las funciones de NaoQi de forma fácil. Para crear la interfaz gráfica desde Python se ha usado la librería GTK+. GTK+ (o The GIMP Toolkit) es un conjunto de bibliotecas multiplataforma para desarrollar interfaces gráficas de usuario, principalmente para los entornos gráficos GNOME, XFCE y ROX aunque también se puede usar en el escritorio de Windows, MacOS y otros. Esto tiene la ventaja añadida que la migración a otros sistemas operativos es fácil, ya que tanto las funciones de NaoQi, la NaoGUI programada en Python, como GTK+ son multiplataforma.

La estructura de la interfaz es la siguiente:

B.2.1. Programación

El programa de Python desarrollado tiene la siguiente estructura:

- Inclusión de librerías y declaración de constantes.
- Conexión con el *broker* principal de Nao, NaoQi.
- Definición de la clase que representa la GUI.
- Ejecución de la GUI.

Para el diseño de la GUI se ha usado el programa Glade, que permite crear ventanas, incluir objetos (*widgets*), y declarar eventos. La programación se hace dentro de la clase de Python. Es necesario declarar objetos que representen a los *widgets*[2] de la GUI, es decir,

[2]En programación, un *widget* es un elemento o bloque de una interfaz gráfica que muestra información o permite su modificación por el usuario, como por ejemplo una ventana o un botón.

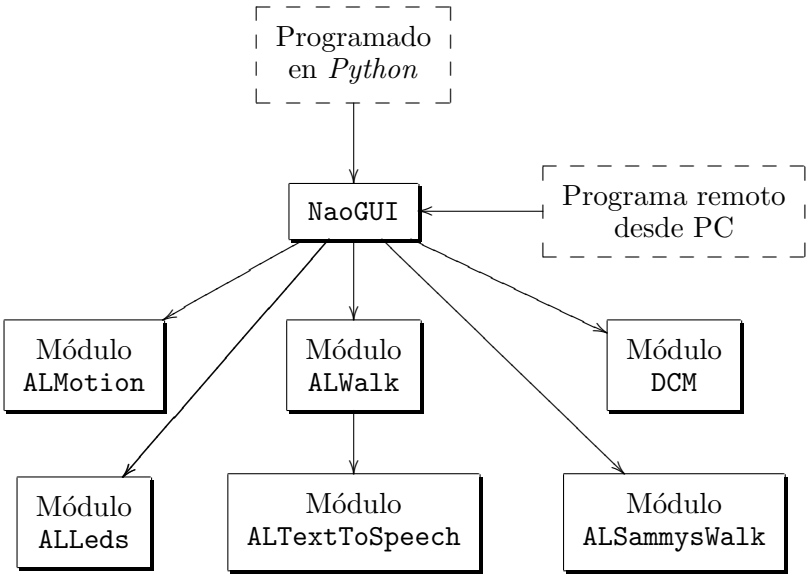


Figura B.3: *Estructura de la GUI*

crear un vínculo entre ambos. Además, hay que vincular los eventos de la GUI a funciones de la clase de Python, que se programarán de acuerdo al comportamiento que se quiera obtener. Todas estas funcionalidades las proporciona la librería de GTK de Glade.

El archivo principal de la GUI se llama `naoGUI.py`, y contiene más de 1800 líneas de código.

B.2.2. Ventana principal

La ventana principal (ver Figura B.4) está contiene una barra de herramientas que permite acceder a otras tres ventanas (editor de posturas, ALWalk, y Sammy's Walk). También acceso a funciones generales de Nao, como por ejemplo cambiar el color de los ojos mediante barras de desplazamiento (una para cada color), o reproducir un texto por los altavoces de Nao, mediante su sintetizador vocal en inglés. Finalmente también permite acceder al control de Nao con el mando de la Wii, que se explicará en el siguiente apartado.

B.2.3. Control con Wiimote

El mando de la Wii, el Wiimote (ver Figura B.6), es de gran utilidad por su facilidad de uso. Tiene la capacidad de detectar la aceleración a lo largo de tres ejes mediante la utilización de un acelerómetro. También cuenta con un sensor óptico PixArt, lo que le permite determinar el lugar al que el Wiimote está apuntando (esta funcionalidad no se usa). Además tiene una serie de botones y una utilidad para la mano izquierda, Nunchuck, que tiene un joystick, botones y un acelerómetro.

Para la programación se ha usado la librería de Python `cwiid`, que permite acceder al mando. La conexión se hace mediante bluetooth. Para saber la dirección bluetooth del mando se puede ejecutar en una consola, previo apretar los botones 1 y 2 del mando a la vez:

Figura B.4: *Ventana principal*

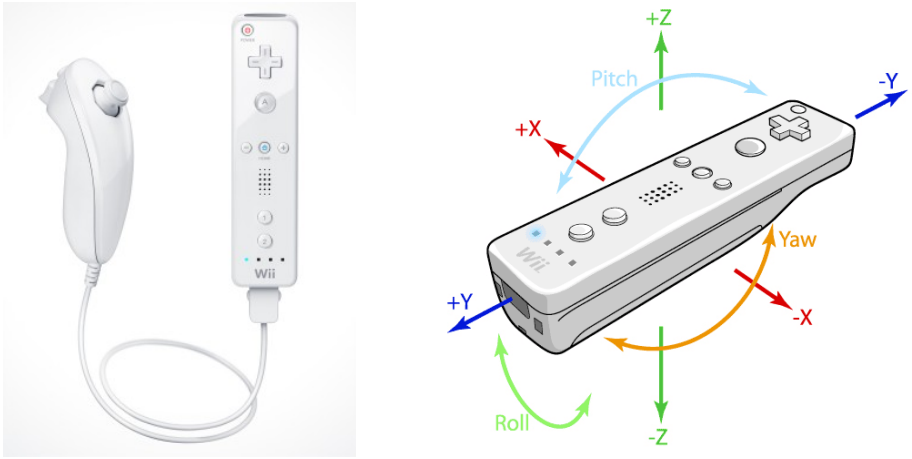
Figura B.5: *Ventana principal de NaoGUI*

```
sudo hcitool scan
Scanning...
    00:19:FD:EE:C8:5A   Nintendo RVL-CNT-01
```

Esta dirección es la que hay que introducir en la GUI para la conexión. Para obtener el estado de los sensores se entra en un bucle que actualiza toda la información del mando cada 50ms, realiza los cálculos necesarios y envía las órdenes a Nao.

Las funcionalidades que se obtienen con el mando son (ver demostración en el vídeo [B.2.3 Nao controlado con mando de la Wii]):

- Control de la cabeza de Nao mediante el *joystick* del Nunchuck.
- Cambio del color de los ojos con el botón A del Nunchuck.
- Reproducción de un texto con el botón B del Nunchuck.
- Si se aprieta el botón 1 del Wiimote, manteniéndolo apretado, al mover el mando hacia arriba el robot se levanta, y al moverlo hacia abajo se sienta.
- Si se aprieta el botón B del Wiimote, manteniéndolo apretado, se controla el movimiento del brazo derecho. El cabeceo del mando (*pitch*) se corresponde al cabeceo del hombro, y lo mismo con el balanceo (*roll*). Si se aprieta el botón + abre la mano, y con el botón - se cierra.
- Si se aprieta el botón A del Wiimote, manteniéndolo apretado, se puede controlar la inclinación del torso, tanto lateral como frontal. Cada segundo se actualiza el cabeceo y balanceo del robot según el balanceo y cabeceo del mando.
- Está preparado para programar funciones de locomoción con las flechas.
- Apretando el botón *home* del mando o en la interfaz directamente, termina la conexión.



(a) *Vista del mando* **(b)** *Ejes de los acelerómetros*

Figura B.6: *Mando de la Wii, Wiimote*

B.2.4. Editor de posturas

La ventana del editor de posturas (Figura B.7) es muy útil para hacer mover el robot. Dispone de una barra de desplazamiento para cada articulación para poder moverlas independientemente. Además cuenta con los botones "Brazos simétricos" y "Piernas simétricas" para poder enclavar las articulaciones derechas e izquierdas de tal forma que realicen movimientos simétricos. También permite abrir y cerrar las manos.

Para enviar los comandos al robot hay que seleccionar un tiempo de interpolación, encender los motores con el botón de la barra de herramientas, y hacer clic en el botón "¡Enviar!" para que los cambios se reflejen en el robot. Asimismo incluye el botón "Posición Actual" para leer la posición actual del robot y actualizarla en la interfaz, haciendo que ambas se corresponden.

Además, se incluyen funciones para abrir y guardar posturas, crear nuevas o borrarlas. Las configuraciones se guardan en archivos .xml, con los valores pada cada articulación. Por ejemplo, para una configuración de reposo se ha creado la postura "sentado.xml", que contiene la siguiente información:

```
<postura>
    <HeadYaw>0.261270957452</HeadYaw>
    <HeadPitch>9.31411894283</HeadPitch>
    <RShoulderPitch>68.0306034165</RShoulderPitch>
    <RShoulderRoll>-23.9968451205</RShoulderRoll>
    <RElbowRoll>61.4387195466</RElbowRoll>
    <RElbowYaw>16.5212406399</RElbowYaw>
    <RWristYaw>-5.10012445136</RWristYaw>
    <LShoulderPitch>68.0306034165</LShoulderPitch>
    <LShoulderRoll>23.9968451205</LShoulderRoll>
    <LElbowRoll>-61.4387195466</LElbowRoll>
    <LElbowYaw>-16.5212406399</LElbowYaw>
```

```
13      <LWristYaw>5.10012445136</LWristYaw>
14      <RAnklePitch>-70.7504393855</RAnklePitch>
15      <RAnkleRoll>0.00240422658784</RAnkleRoll>
16      <RHipPitch>-40.7841660248</RHipPitch>
17      <RHipRoll>-0.876513053546</RHipRoll>
18      <RKneePitch>125.599679289</RKneePitch>
19      <LAnklePitch>-70.7504393855</LAnklePitch>
20      <LAnkleRoll>-0.00240422658784</LAnkleRoll>
21      <LHipPitch>-40.7841660248</LHipPitch>
22      <LHipRoll>0.876513053546</LHipRoll>
23      <LKneePitch>125.599679289</LKneePitch>
24      <HipYawPitch>-14.9391895357</HipYawPitch>
25  </postura>
```

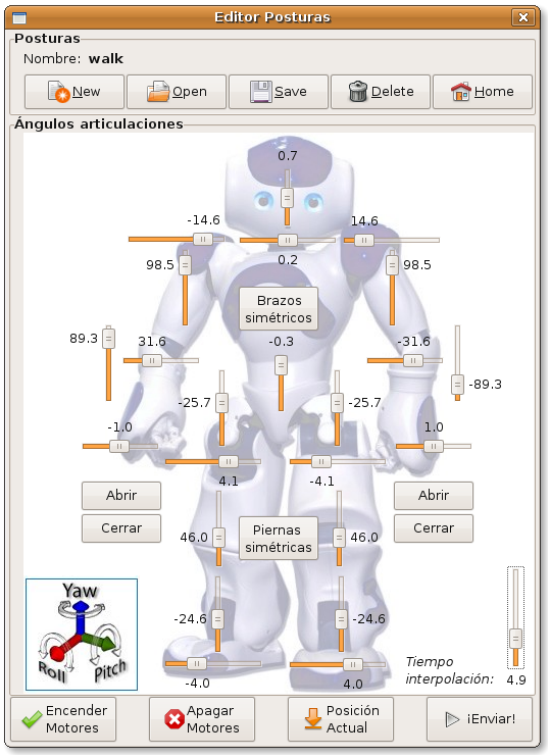


Figura B.7: *Ventana del editor de posturas*

B.2.5. Interfaz ALWalk

La ventana ALWalk (ver Figura B.8) contiene las funciones necesarias para hacer caminar el robot con el algoritmo ZMP propio de Nao. Permite configurar los parámetros básicos del algoritmo ZMP, los parámetros avanzados y enviar las órdenes correspondientes a locomoción frontal, lateral, en curva y girando (para más detalles de los parámetros ver apartado 5.1.1).

Además permite configurar la *stiffness* de cada grupo de articulaciones, para poder generar una locomoción estable. Todos los valores por defecto de esta ventana están configurados

acorde con los encontrados en la memoria principal que hacen estable la locomoción (ver apartado 5.1.3).

Para que la configuración introducida se aplique sobre Nao hay que hacer clic en los correspondientes botones "Enviar".

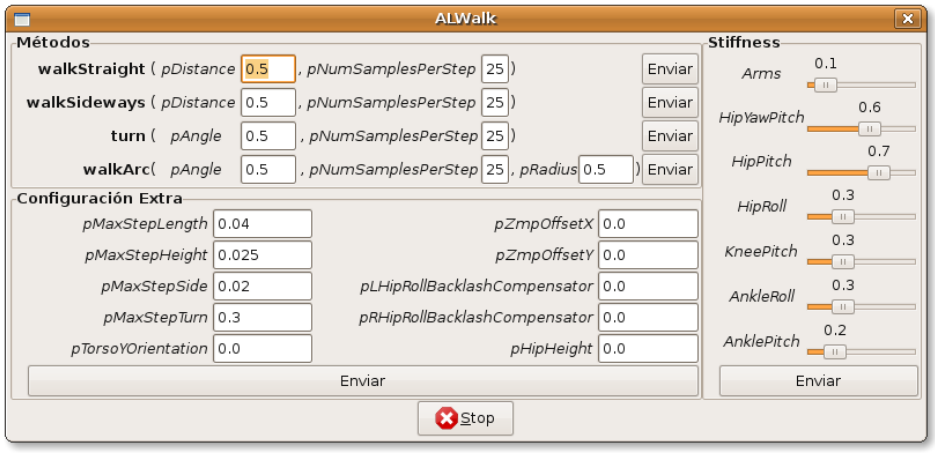


Figura B.8: *Ventana ALWalk*

B.2.6. Interfaz Sammy's Walk

La ventana Sammy's Walk (ver Figura B.8) contiene las funciones necesarias para hacer caminar el robot con el algoritmo *Sammy's Walk* diseñado en este proyecto. Permite configurar todas las articulaciones para cada estado (ver apartado 5.2), así como los tiempos correspondientes. También se puede modificar el parámetro a del generador de trayectorias, y k_{vel} para la corrección de velocidad.

Los estados se guardan localmente con la misma estructura de archivos que en Nao. Cuando se hace clic en el botón "Enviar datos" se copian el robot los archivos, y a continuación ya se puede ejecutar el algoritmo con el botón "Caminar". Con el botón "Descansar" se para el algoritmo, y con "Descargar y ver registro" se copia al PC la telemetría generada.

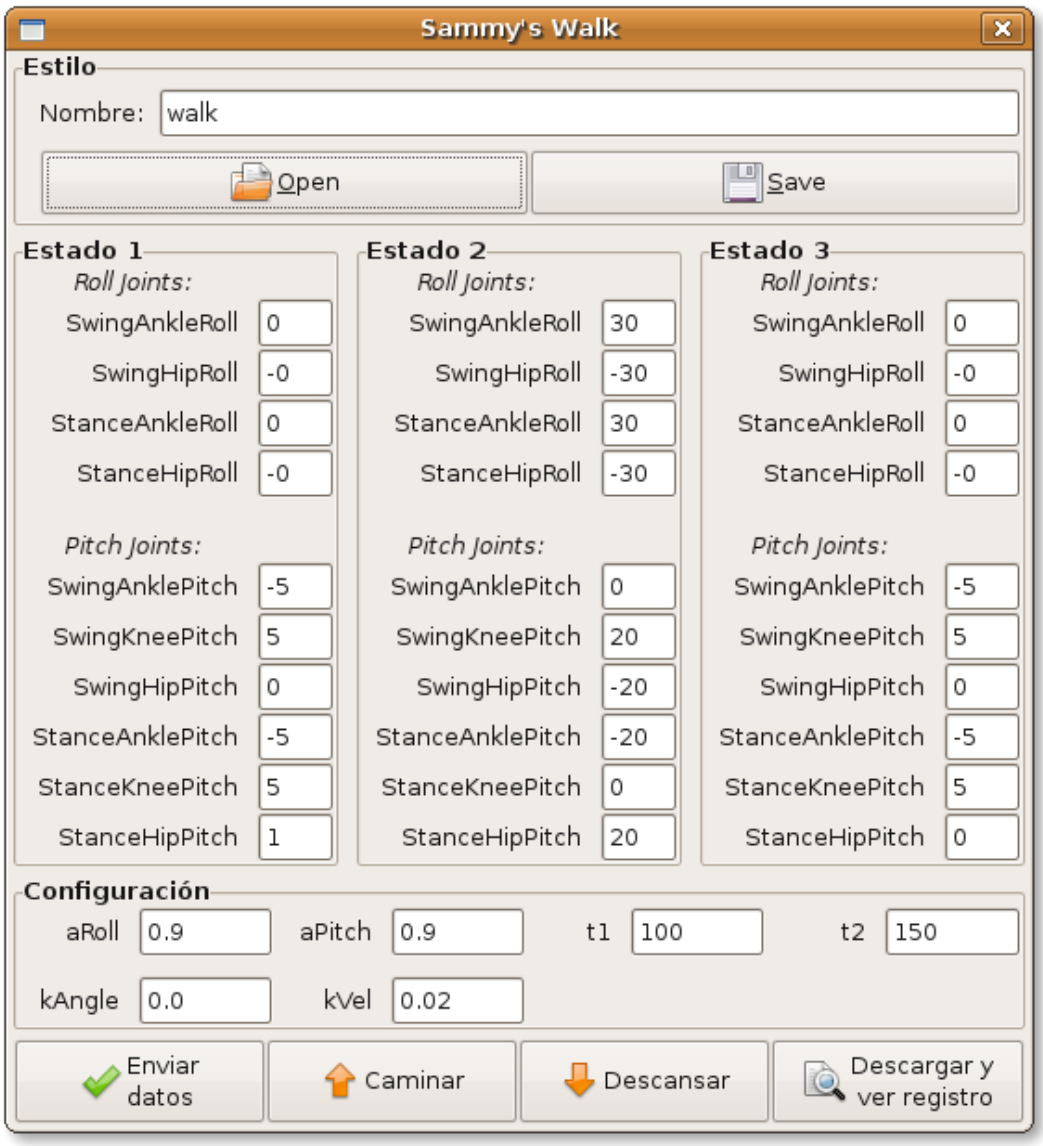


Figura B.9: *Ventana Sammy's Walk*

Apéndice C

Presupuesto

Como en todo proyecto de investigación resulta imprescindible hacer un análisis de costes y posibles beneficios. Particularizando al proyecto presente, se hará un análisis de los costes que ha implicado el desarrollo de este proyecto, y las repercusiones de los resultados obtenidos.

C.1. Análisis de costes

En el proceso de desarrollo se tendrán en cuenta diversos factores: *hardware, software*, tiempo involucrado y otros. La Tabla C.1 explica los costes de los diferentes elementos (algunos de ellos son estimaciones).

Elemento	***Coste unitario***	***Unidades***	***Coste***
Ordenador Procesador AMD 2Ghz Memoria RAM 2GB Disco duro 160GB Monitor, Conexión ethernet y otros	**1.000 €/ud**	**1 ud**	**1.000 €**
Robot Nao Nao Academics Edition Software de desarrollo Licencia de *Coreographe*	**11.000 €/ud**	**1 ud**	**11.000 €**
Webots (licencia gratuita)	–	–	–
Tiempo dedicado	**540h**	**15€/h**	**8.100€**
Investigación con Webots	120h	15€/h	1.800€
Puesta en marcha de Nao	60h	15€/h	900€
Desarrollo del algoritmo (pruebas, programación...)	240h	15€/h	3.600€
Otros	120h	15€/h	1.800€
Tiempo tutorización	**100h**	**35€/h**	**3.500€**
		TOTAL	***23.600€***

Tabla C.1: *Presupuesto*

C.2. Beneficios

Los beneficios económicos de este proyecto son difícilmente medibles. Como resultados importantes, se ha obtenido una reducción consumo energético específico de locomoción del 5 % respecto al algoritmo básico ZMP implementado en Nao. Esto implica una mayor duración de la batería, y el consecuente ahorro energético. Además plantea una línea de investigación que en el futuro puede reportar mayores beneficios.

Apéndice D

Impacto medioambiental

En sí mismo Nao está hecho de material electrónico e informático: plástico en la carcasa, múltiples placas y procesadores internos, motores, y otros componentes varios. Se tratará conjuntamente el impacto ambiental de Nao y el PC necesario para su uso.

El impacto que tienen este tipo de componentes se basa en los siguientes puntos [5][2][26]:

- El impacto ambiental de la producción de material informático y plásticos.
- El consumo eléctrico en la fase de uso.
- El impacto ambiental de almacenar ordenadores en vertederos.

D.1. Proceso de fabricación

En [1] se hace una exposición del impacto medioambiental de la producción de material informático. A continuación se muestra un extracto de este documento. La exposición hecha para el material de los ordenadores es perfectamente válida para los materiales de Nao.

Existen tres problemas medioambientales relacionados con la fabricación de ordenadores: el uso de muchas substancias tóxicas en el proceso de producción, un consumo muy elevado de agua y energía, y el gran volumen de residuos (también tóxicos) que generan. Los materiales más abundantes en un ordenador son plásticos, acero, silicio, aluminio y cobre. Pero en la fabricación de los chips y las placas se utilizan hasta un millar de sustancias químicas, algunas de ellas muy contaminantes y conocidos cancerígenos.

Una de las sustancias problemáticas son los retardantes de llama con que la ley obliga a cubrir los circuitos impresos, los cables y las carcasas para hacerlos poco inflamables. Los usados más habitualmente son halogenados: contienen bromo o flúor, lo que causa que durante la fabricación, el vertido o la incineración de los ordenadores se liberen dioxinas y otros contaminantes en el medio. Pero también se liberan al aire mientras los ordenadores se usan: algunos estudios han detectado una concentración de bromo en la sangre más elevada que la media entre la gente que trabaja en oficinas. Estas substancias causan sobretodo desorden en el sistema hormonal (glándula tiroidea), pero posiblemente también cáncer y desordenes en el desarrollo neuronal. Se acumulan en los tejidos grasos (y por lo tanto, también en la leche materna) y se mueven hacia arriba en la cadena alimentaria.

También se utilizan metales pesados, sobretodo plomo, cadmio y mercurio. El plomo se utiliza para soldar los chips a las placas, y en las pantallas de rayos catódicos (las que no son planas) para absorber una parte de las radiaciones electromagnéticas que generan las

pantallas. El cadmio y el mercurio también se utilizan en dichas pantallas. Durante el uso de los ordenadores no estamos expuestos a dichos elementos, pero se convierten en un peligro cuando se liberan al medio durante la fabricación y al lanzar el ordenador. Pasan a los seres vivos a través de la cadena alimentaria y, como no los podemos metabolizar, se acumulan en los tejidos y son una causa de cáncer. Durante la fabricación de los chips se emiten al aire perfluorocarbonos (PFCs), que son gases que permanecen durante mucho tiempo en la atmósfera y contribuyen al efecto invernadero. Forman parte de los productos cuya emisión se acordó reducir en el Protocolo de Kyoto para frenar el cambio climático.

Otras sustancias tóxicas que utilizan los ordenadores son arsénico, benceno, tolueno y cromo hexavalente. Las carcasas se suelen proteger con pinturas que contienen disolventes orgánicos; durante la aplicación se liberan compuestos orgánicos volátiles, que provocan que se acumule ozono en las capas bajas de la atmósfera. El ozono al nivel del suelo causa problemas respiratorios y dificulta el crecimiento normal de los vegetales. Por otro lado, los cables suelen ser de PVC. Los procesos más sencillos, como el montaje de placas y ordenadores, los suelen hacer empresas subcontratadas en Malasia, Tailandia, Filipinas, Vietnam, Indonesia, China, recientemente Europa del Este, y en menor cantidad Centroamérica, Brasil y Sudáfrica. En las plantas de montaje suelen trabajar mujeres jóvenes cobrando salarios bajos, con jornadas muy largas, presión por producir deprisa, y sin sindicatos. A diferencia de lo que pasa en el sector de los juguetes o del textil, las grandes empresas de material electrónico todavía no han comenzado a elaborar códigos de conducta que establezcan unas condiciones laborales mínimas en sus fábricas y empresas proveedoras.

Las empresas son reticentes a colaborar en estudios de las substancias tóxicas sobre la salud. Parece claro que hay una tasa de abortos y malformaciones en bebés más alta de lo normal entre las mujeres que trabajan en salas blancas (los trajes especiales que usan evitan la exposición de las obleas de chips a las impurezas que puedan portar los trabajadores, pero no evitan la exposición de los trabajadores a los tóxicos). Durante la década de los 90, en EEUU y Escocia se ha demandado a algunas empresas porque la frecuencia de cáncer de cerebro entre los trabajadores de salas limpias es 2'5 veces más alta que la media, pero los casos todavía están pendientes por falta de evidencias concluyentes. En las plantas de montaje de placas, el peligro más grande es el plomo que se utiliza para soldar. A principios de los 90 murieron cuatro trabajadores en Tailandia: la autopsia les detectó un nivel de plomo en la sangre más alto de lo normal. El resultado fue negado por la empresa donde trabajaban y silenciado por el gobierno, el principal interés del cual es atraer inversores extranjeros.

D.2. Uso diario

El consumo eléctrico de un ordenador sencillo se puede estimar en funcionamiento normal en 300W. El consumo de Nao en funcionamiento oscila entre 30W (quieto) y 70W (en actividad), se tomará para los cálculos 50W.

A través de la información proporcionada por Red Eléctrica de España se estima la producción de CO_2 para la generación de electricidad en 290 $(g/h)/kW$. Por tanto, se puede estimar que el uso del PC + Nao hacen un total de 350W, y por tanto la electricidad necesaria generará $\approx 100 gCO_2/h$.

El fabricante de Nao no proporciona información sobre la vida útil del robot. Se estima durante el desarrollo de este proyecto que se han empleado 450h usando el PC+Nao, lo que supondría una emisión de 45kg de CO_2.

D.3. Fin de la vida útil

Al final de la vida útil del PC y Nao hay que desechar sus componentes. El siguiente análisis se ha extraído de [5], complementado con explicaciones propias.

Los aparatos electrónicos están constituidos por un conjunto de componentes de entre los cuales conviene destacar: aparatos de visualización como tubos de rayos catódicos y pantallas de cristal líquido, vidrio, plásticos con materiales ignífugos, circuitos impresos, cables, interruptores de mercurio y magnetotérmicos, pilas, condensadores, resistencias, relés, etc. Aproximadamente el 50 % del peso de aparatos electrónicos y eléctricos son metales, principalmente aceros, aluminio, cobre, plomo, mercurio y metales preciosos. El resto de materiales quedan repartidos entre dos fracciones que se encuentran en porcentajes similares y que son plásticos y vidrios. Dependiendo del aparato considerado, estos datos pueden variar. Así, mientras los ordenadores contienen un 23 % de peso en plásticos, en los equipos dedicados a telecomunicaciones puede llegar hasta un 50 %.

Los compuestos más problemáticos desde el punto de vista medioambiental contenidos en los residuos eléctricos y electrónicos son los metales pesados, el PVC, los materiales ignífugos bromados y los compuestos binéfilos policlorados (PCB). Hablando de metales, que poseen el 70 % del valor residual de un ordenador, podemos encontrar plomo en las soldaduras y los tubos de rayos catódicos, bario en los tubos de rayos catódicos, cadmio en las baterías, antimonio en el encapsulado de los chips, berilio en los PCs antiguos y las conexiones de teléfonos móviles, cromo en los metalizados, mercurio en baterías, interruptores y las bombillas que iluminan las pantallas planas, fósforo en monitores, arsénico y silicio en los microprocesadores, acero en las carcasas, aluminio en los discos duros, cobre en toda la electrónica, y metales preciosos en las placas de circuitería.

A pesar de todo, y a diferencia de los desechos tradicionales, el principal impacto ambiental de la basura electrónica se debe principalmente a un procesado inadecuado, más que a su contenido tóxico inherente.

En la actualidad, en Europa la mayor parte de los residuos eléctricos y electrónicos se incorporan a los flujos de los residuos urbanos, lo que quiere decir que se desechan en vertederos o se incineran sin ningún tratamiento previo. En 1998 en los EEUU sólo se reciclaron un 11 % de los ordenadores personales y un 26 % de los periféricos de los ordenadores obsoletos. Así, buena parte de los agentes contaminantes que se encuentran en los flujos de residuos urbanos proceden de dichos aparatos.

En el vertido de residuos electrónicos se liberan metales pesados (como plomo, cadmio y mercurio) y otras substancias tóxicas que acaban contaminando la tierra y los acuíferos. En la incineración de los retardantes de llama brominados y del PVC se generan dioxinas y furanos extremadamente tóxicos, y el cobre, abundante en la basura electrónica, empeora la situación debido a que es un buen catalizador de la formación de dioxinas. Si además la incineración o quemado se realiza al aire abierto, la inhalación de las emisiones de gas puede provocar ataques de asma, problemas respiratorios e irritación de ojos. La exposición crónica a estas emisiones genera enfisema y cáncer.

Se calcula que, con un tratamiento adecuado, se podría reaprovechar entre el 70 % y el 90 % de lo que se lanza, reusándolos cuando fuera posible o reciclándolos. En éste último caso, los equipos se desmontan y los componentes potencialmente peligrosos se aíslan y se

entregan a gestores autorizados para su tratamiento. En la fase de trituración, los materiales se clasifican por tipos, se revalorizan, se tratan para ser recuperados y, finalmente, se venden a las industrias que los pueden aprovechar.

Trece países, la mayoría europeos, ya aprobaron normas que prevén la obligación de reciclar los ordenadores. En este aspecto, la directiva *Wastes from Electrical and Electronic Equipment* de la Unión Europea, hace responsables a las empresas de electrónica de deshacerse de sus productos una vez los usuarios ya no los quieran. Se estima que para ello el coste medio a añadir al precio de producción de un equipo ronda los 90 euros.

En España dicha directiva se traduce en el real decreto 208/2005 (que entra en vigor el 13/8/2005), que se puede consultar en http://www.boe.es/boe/dias/2005-02-26/pdfs/A07112-07121.pdf (con una pequeña corrección en http://www.boe.es/boe/dias/2005-03-30/pdfs/A10800-10800.pdf). Esta ley indica que son los ciudadanos los que deben depositar los equipos en puntos de recogida específicos y que a partir de ahí son los fabricantes quienes deben hacerse cargo del procesado de los residuos.

Pero gran parte de los residuos informáticos (se calcula que el 80 % en el caso de EEUU y el 60 % en el caso de Europa) se envían a países en vías de desarrollo, donde los materiales contaminantes acaban en campos y costas, ensuciando aguas y suelos, cultivos, animales y agua potable. El 2002 se trasladaron a Asia entre 6 y 10 millones de ordenadores obsoletos.

Nos encontramos con que las regiones más ricas del planeta, el llamado primer mundo, está haciendo caso omiso del Convenio de Basilea de las Naciones Unidas, que se puede consultar en http://www.basel.int/, y que desde 1989 prohíbe la exportación de residuos peligrosos de las naciones ricas a las pobres bajo cualquier pretexto, incluyendo el reciclaje. Hasta la fecha, Estados Unidos es el único país industrializado que no ha ratificado dicho convenio.

Así las regiones más pobres de Asia (principalmente en China, India y Pakistan) se están convirtiendo en las colonias-vertedero de residuos tóxicos del resto del planeta, donde la gente más pobre se expone a envenenamientos intentando extraer los diferentes metales y componentes mediante tecnología medieval y sin ningún respeto hacia el medio ambiente, por un mísero salario de 1,5 dólares diarios.