UNIVERSIDAD DE CUENCA

FACULTAD DE INGENIERÍA

ESCUELA DE ELECTRÓNICA Y TELECOMUNICACIONES

Diseño e implementación de un sistema de control para la caminata de un Bípedo con 10 DOF en el plano sagital

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTOR:

GALO DANIEL ASTUDILLO HERAS

C.I. 1720438512

DIRECTOR:

ING. LUIS ISMAEL MINCHALA ÁVILA, PhD.

C.I. 0301453486

CUENCA – ECUADOR

2018

RESUMEN

En este documento se presenta una aplicación de control óptimo para estabilizar la dinámica de un bípedo con diez grados de libertad (10 DOF). La trayectoria del robot se genera siguiendo un patrón de caminata humano, cuyos ángulos son modificados considerando el punto de momento cero (ZMP). Se tiene como objetivo modelar, simular y controlar la estabilidad del robot empleando un prototipo cuyas masas de sus miembros son conocidas junto con la posición de sus respectivos centros de masa (CoM). Las trayectorias de caminata se plantean para el plano sagital y son construidas siguiendo el ZMP del bípedo y manteniéndolo dentro del polígono de soporte, dividiendo el proceso en dos etapas: etapa de soporte único y etapa de soporte doble. El modelo lineal de péndulo invertido (LIPM) se usa para diseñar el controlador con retroalimentación y predicciones futuras del comportamiento del sistema. Se emplea la acción correctiva para ambas etapas. En las pruebas por software se observó que el control actúa de manera robusta ante perturbaciones a pesar de utilizar un modelo simplificado. Con este resultado en el controlador y la trayectoria generada a través del patrón natural del ser humano, la metodología se considera adecuada para su aplicación en el bípedo real.

Palabras Clave: ROBOT, BÍPEDO, LIPM, ZMP, BIOMECÁNICA DE LA MARCHA.

ABSTRACT

In this document, an application of optimal control is presented to achieve the stability of the dynamics of a biped robot with 10 degrees of freedom (10 DOF). The trajectory of the robot is generated following the pattern based on human gait, modifying the angles of the articulations to create steps that consider the zero moment point (ZMP). The goals are to model, simulate and control the stability of the robot utilizing a prototype whose mass and center of mass (CoM) of each joint are known. The gait cycle is proposed for the sagittal plane, and it is built following the ZMP of the biped, considering the polygon of support. This process has been divided into two stages: single support phase and double support phase. The linear inverted pendulum model (LIPM) is used to design the controller with feedback and future predictions of the behavior of the system. The corrective action applies for both stages mentioned before. In the software tests, it was shown that the control is robust when perturbations are present. With this result, the trajectory generated, the control algorithm and methodology used in the software is applied successfully to the real robot.

Keywords: ROBOT, BIPED, LIPM, ZMP, HUMAN GAIT, BIOMECHANICS.

Índice

Cláusula de Propiedad Intelectual

Yo, Galo Daniel Astudillo Heras, autor del trabajo de titulación "DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL PARA LA CAMINATA DE UN BÍPEDO CON 10 DOF EN EL PLANO SAGITAL", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 16 de marzo del 2018

DAH

Galo Daniel Astudillo Heras

C.I: 1720438512

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Yo, Galo Daniel Astudillo Heras, en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL PARA LA CAMINATA DE UN BÍPEDO CON 10 DOF EN EL PLANO SAGITAL", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 16 de marzo del 2018

Galo Daniel Astudillo Heras

C.I: 1720438512

.

DEDICATORIA

Dedico este trabajo a Dios, que me dio la oportunidad de poder realizarlo. A mi familia, en especial a mis padres, Carmita y Galo, que me apoyaron en todo en cuanto pudieron. A mi hermana Lizeth, que me motivó a seguir adelante. A mis abuelos Rogerio y Martha, que siempre están en mi corazón. A mi amada novia Alexandra, por su ánimo y su apoyo emocional para culminar este trabajo.

Galo Daniel Astudillo Heras

AGRADECIMIENTOS

Agradezco a Dios por haber estado en todo momento conmigo. Agradezco a mi familia y a mi novia por todo el apoyo recibido. A mi profesor y mentor, Ing. Ismael Minchala, PhD., por ser una gran persona y también una motivación para aventurarme en el hermoso mundo del control. Estoy contento por haber recibido su guía e instrucción a lo largo de la elaboración de este proyecto.

Galo Daniel Astudillo Heras

Capítulo I: INTRODUCCIÓN

Introducción

En el Ecuador, desde la vicepresidencia de Lenin Moreno durante el periodo 2007-2013, se ha incrementado la ayuda gubernamental dirigida a personas con discapacidades físicas (i.e. falta de algún miembro o extremidad) (Vicepresidencia de la República, 2012). Durante el programa "Misión Solidaria Manuela Espejo", el gobierno ha tenido como objetivo dotar de prótesis a las personas que hayan perdido miembros tanto inferiores como superiores. A pesar de que ya se ha empezado la producción de prótesis a nivel local, los centros de prótesis vinculados al MSP (Ministerio de Salud Pública) proveen miembros mecánicos de "baja calidad, poco avance tecnológico y escasa cobertura" (Secretaría Nacional de Planificación y Desarrollo, 2014). Además de estas falencias, todavía no se producen exoesqueletos que ayuden también con la rehabilitación física de las personas con discapacidad.

Los exoesqueletos, prótesis mecánicas y robots humanoides en general, son máquinas complejas que requieren de una integración de sensores, actuadores y algoritmos de control. La integración de estos elementos es importante porque permite al robot realizar el seguimiento de una trayectoria que depende de la tarea a realizar. Para el caso de la biomecánica de marcha de un bípedo, la trayectoria de caminata puede ser dirigida por patrones generados en tiempo real utilizando un modelo lineal de péndulo invertido o LIPM (Kajita et al., 2002). Otra manera para generar el patrón de caminata es utilizar una trayectoria de referencia basada en el punto de momento cero o ZMP (Crisóstomo, Coimbra, & Ferreira, 2009). Al mismo tiempo que el patrón es generado, la dinámica requiere algún tipo de control que puede estar basado en el centro de masa o CoM (Muscolo et al., 2011).

Tanto la trayectoria como el control dinámico son de trascendental importancia para la estabilización del sistema. Las trayectorias generadas a través de la aplicación de la cinemática inversa del robot resultan en posiciones y ángulos que carecen de naturalidad en el ciclo de caminata. Con lo antes mencionado, se plantea una simulación que sirva de base para futuras construcciones de exoesqueletos y prótesis mecánicas de las extremidades inferiores a nivel local. Se presenta un modelo para simular la caminata de un bípedo a escala con 10 grados de libertad (DOF) siguiendo un conjunto de patrones de caminata capturadas siguiendo un patrón humano y utilizando un control con retroalimentación en conjunto con un observador.

Primero, se presenta el bípedo junto con los parámetros Denavit-Hartenberg (D-H). Posteriormente, se utiliza una trayectoria creada por el Departamento de Ingeniería Eléctrica, Electrónica y Telecomunicaciones (DEET) de la Universidad de Cuenca basada en la biomecánica de la marcha humana para que el robot siga un desplazamiento natural. Esta trayectoria es ajustada al bípedo para cumplir con las dos etapas de caminata: etapa con suporte único y etapa con soporte doble. Después, se modifican los patrones de caminata en función del ZMP para facilitar el control. Se elabora el control dinámico del sistema para estabilizar el robot utilizando el centro de masa (CoM) como referencia además de las predicciones futuras del observador. Finalmente, se visualizan los resultados en MATLAB.

1.1 ZMP para robots bípedos

El punto de momento cero o de un bípedo fue introducido por Vukobratovic y Stepanenko y se considera un lugar en el suelo en el que la reacción de la fuerza dinámica en el punto de contacto del polígono de soporte y el suelo

no produce momento en la dirección horizontal. En otras palabras, en este punto, las fuerzas inerciales y la gravedad deben en resultante ser cero en el plano horizontal. Durante la caminata, el ZMP debe siempre estar dentro del polígono de soporte; cuando esta condición se cumple, la caminata se considera dinámicamente estable.

Si el robot tiene en cada pierna al menos una articulación como tobillo y siempre mantiene al menos una extremidad en contacto con el suelo, entonces el ZMP puede ser utilizado como criterio de estabilidad (Ao, Amouzegar, & Rieger, 2011).

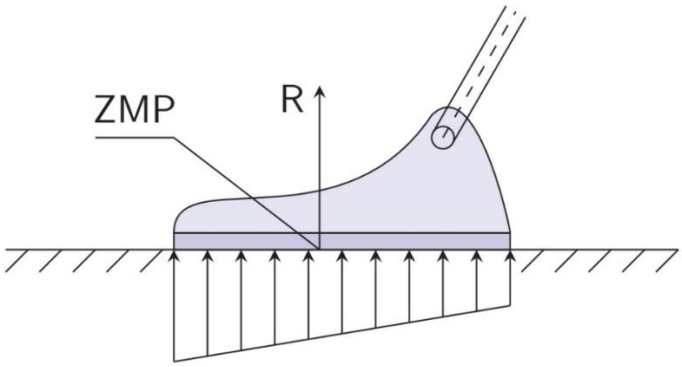

Figura 1.1. Punto de momento cero (Dang, 2012)

Para el cálculo del ZMP, consideremos las fuerzas $f_1, f_2, \dots, f_N$ como las fuerzas de contacto con el suelo, siendo $p_1, p_2, \dots, p_n$ sus respectivas posiciones y R la fuerza resultante (Figura 1.1). Posteriormente, la posición del ZMP puede calcularse utilizando la ecuación (1.1).

$$ZMP = \frac{\sum_i^N p_i f_i}{\sum_i^N f_i} \qquad (1.1)$$

Al considerar que el ZMP nunca deja el polígono de soporte y que el desplazamiento se da en el plano sagital con un piso horizontal, el par mecánico se reduce a:

$$\mathrm{T}_x^{ZMP} = \mathrm{T}_y^{ZMP} = 0 \qquad (1.2)$$

Este método es bastante útil cuando se utilizan sensores de presión en el pie, básicamente la ecuación (1.1) realiza una suma ponderada de las fuerzas medidas por los sensores en función de su ubicación. Sin embargo, no se utilizó el método de sensores y se empleó un modelo que permite calcular el ZMP con otros parámetros. El modelo que se presenta a continuación sirvió de base para la simulación y el diseño del controlador.

1.2 Modelo lineal de péndulo invertido (LIPM)

El modelo dinámico del bípedo puede ser tan complejo como se desee si se añade cada vínculo y sus respectivas inercias. Por esta razón, se considera el sistema como un péndulo invertido. Con esta simplificación, el bípedo ya no se analiza por vínculos, sino como una sola masa concentrada. Para poder utilizar un modelo simplificado, aplicamos las restricciones y el modelo que se detallan en (Kajita et al., 2002) con una aplicación en (Czarnetzki, Kerner, & Urbann, 2009).

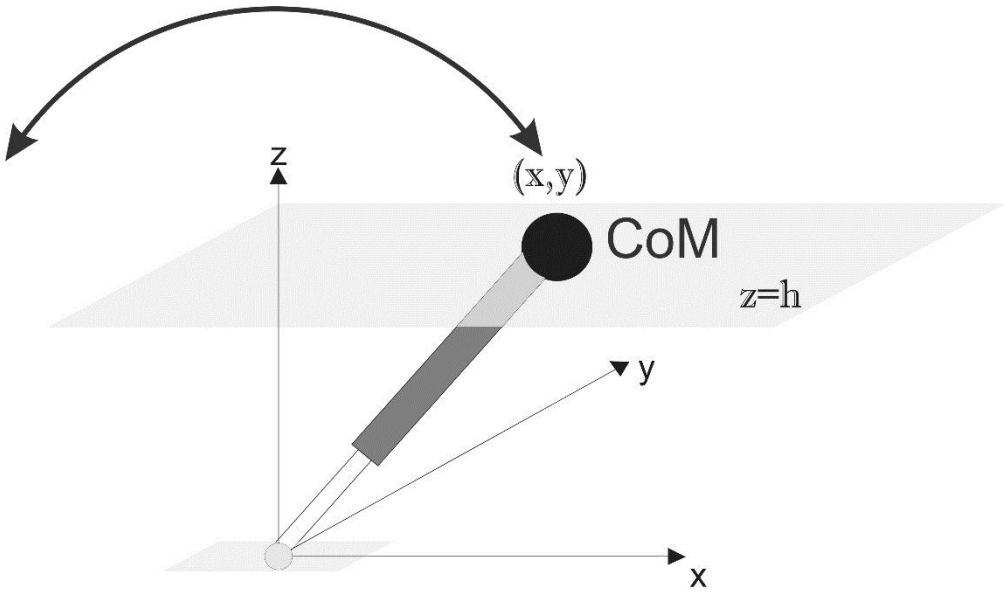

Figura 1.2. Modelo LIPM

En este modelo (Figura 1.2), la dinámica se reduce a una sola masa concentrada en su CoM que se mueve arbitrariamente sobre un plano que tiene como vector normal $(k_x, k_y, -1)$ y se intersecta con el eje z en el punto z_d. El valor de z_d en la Figura 1.2 es h y pasa a ser la altura del péndulo desde el punto de contacto con el suelo.

Debido a que el plano en el que se mueve el CoM es el plano xy y solo el plano xy, $k_x = 0$ y $k_y = 0$. Si m es la masa de péndulo, g es la aceleración de la gravedad, y $T_x\ y\ T_y$ los respectivos pares mecánicos de los ejes, la dinámica del péndulo invertido se representa con las ecuaciones (1.3) y (1.4).

$$\ddot{x} = \frac{g}{z_d} x - \frac{1}{m * z_d} T_x \tag{1.3}$$

$$\ddot{y} = \frac{g}{z_d} y - \frac{1}{m * z_d} T_y \tag{1.4}$$

De dónde x e y son las posiciones del CoM para el plano horizontal. Los demás parámetros se mencionaron en el párrafo anterior.

Si se conoce la posición (p_x, p_y) del ZMP del péndulo, se puede aplicar la ecuación (1.5) para encontrar los torques $T_x\ y\ T_y$.

$$-m\,.\,g\,.\,p_i = Ti \tag{1.5}$$

Utilizando la ecuación (1.5) y reemplazando para x e y en (1.3) y (1.4), tenemos las ecuaciones (1.6) y (1.7) para la posición del péndulo.

$$p_x = x - \frac{z_d}{g}\ \ddot{x} \tag{1.6}$$

$$p_y = y - \frac{z_d}{g} \ddot{y} \tag{1.7}$$

Como se muestra en las ecuaciones, es posible tratar por separado la posición en x e y para planificar la estrategia de control y al mismo tiempo generar la trayectoria basada en ZMP. Además, se puede observar que el

ZMP se convierte en el CoM cuando la aceleración es cero. También se puede lograr el equilibrio en la caminata dinámica, incluso si el CoM esté fuera del polígono de soporte, siempre que el ZMP se mantenga dentro del área del pie.

1.3 Biomecánica de la marcha humana

El cuerpo humano tiene un sofisticado mecanismo de control para realizar una caminata bípeda de manera eficiente, periódica y estable. Sin embargo, hay modelos simplificados (como el LIPM, mencionado anteriormente) que se convierten en herramientas para poder analizar y entender de mejor manera los principios de caminata. Por esta razón, en esta tesis se parte de mediciones tomadas de los ángulos de las articulaciones de una persona al caminar (estas mediciones fueron tomadas por el DEET en la Universidad de Cuenca) y se modifican para que el bípedo utilice el ZMP en todo el ciclo. En cuestión de la trayectoria de caminata, se consideran dos fases en cada ciclo. La fase de soporte doble y la fase de soporte único, como se representa en la figura 1.3. En la fase de soporte doble, ambas extremidades soportan el peso. En la fase de soporte único, todo el peso está concentrado en una sola pierna. En todo momento, al menos uno de los dos pies se encuentra en contacto con el suelo.

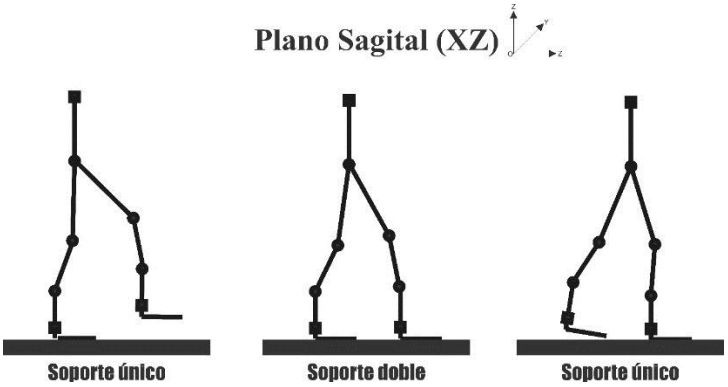

Figura 1.3. Fases de ciclo de caminata: soporte doble y soporte único

Si el tiempo de cada paso es un intervalo $\{0, t_{paso}\}$, entonces lo podemos dividir en la fase de soporte único con duración $\{0, t_{su}\}$ y la fase de soporte doble en $\{t_{su}, t_{paso}\}$. Según (Mummolo, Mangialardi, & Kim, 2013), cada extremidad pasa aproximadamente un 38 % del ciclo como pierna de soporte único, 24% como parte del soporte doble y 38% como pierna de balanceo como se muestra en la figura 1.4.

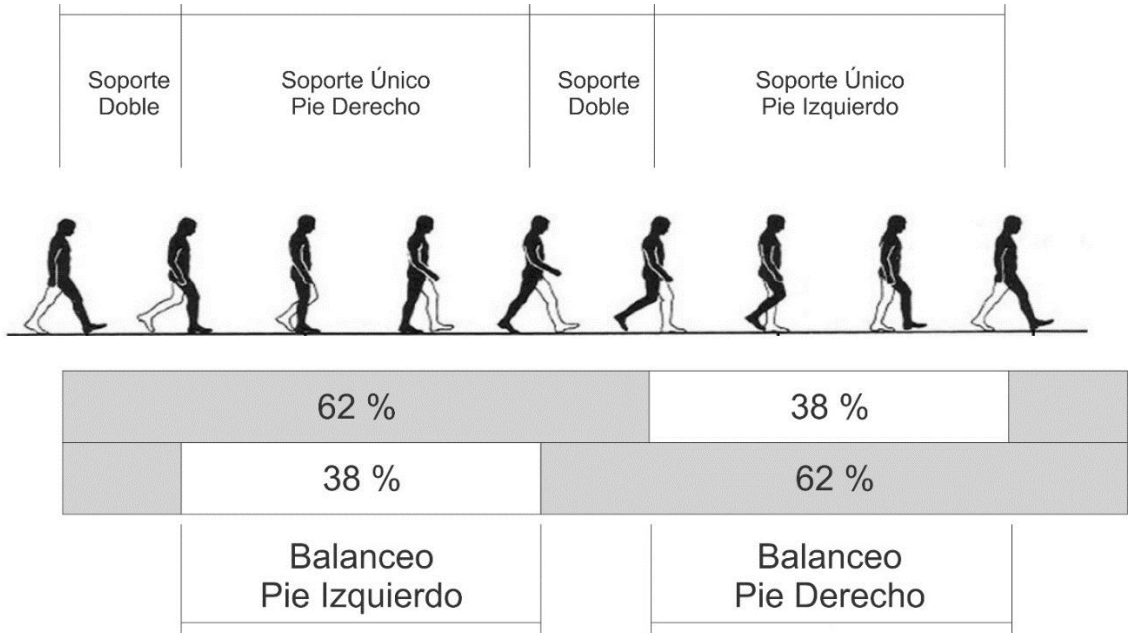

Figura 1.4. Pie de soporte y pie de balanceo (adaptada y modificada de (Mummolo, Mangialardi, & Kim, 2013))

Para que el robot sea dinámicamente estable, el ZMP debe estar sobre el suelo de manera que este no salga del polígono de soporte en cualquier momento del ciclo, caso contrario se ubicaría en una posición inestable. Si solo se utilizaría el CoM para la estabilidad, la caminata sería estáticamente estable y fuera necesario que la velocidad de desplazamiento sea relativamente lenta para que las fuerzas inerciales sean despreciables y terminar el ciclo sin caer. Por esta razón, no se utiliza el enfoque con posiciones estáticamente estables, sino con posiciones dinámicamente estables.

El ajuste de las trayectorias obtenidas por el centro de investigación debe generar un nuevo desplazamiento que facilite la planificación de la estrategia de control basado en el ZMP. El diagrama de flujo de este procedimiento se encuentra en la figura 1.5.

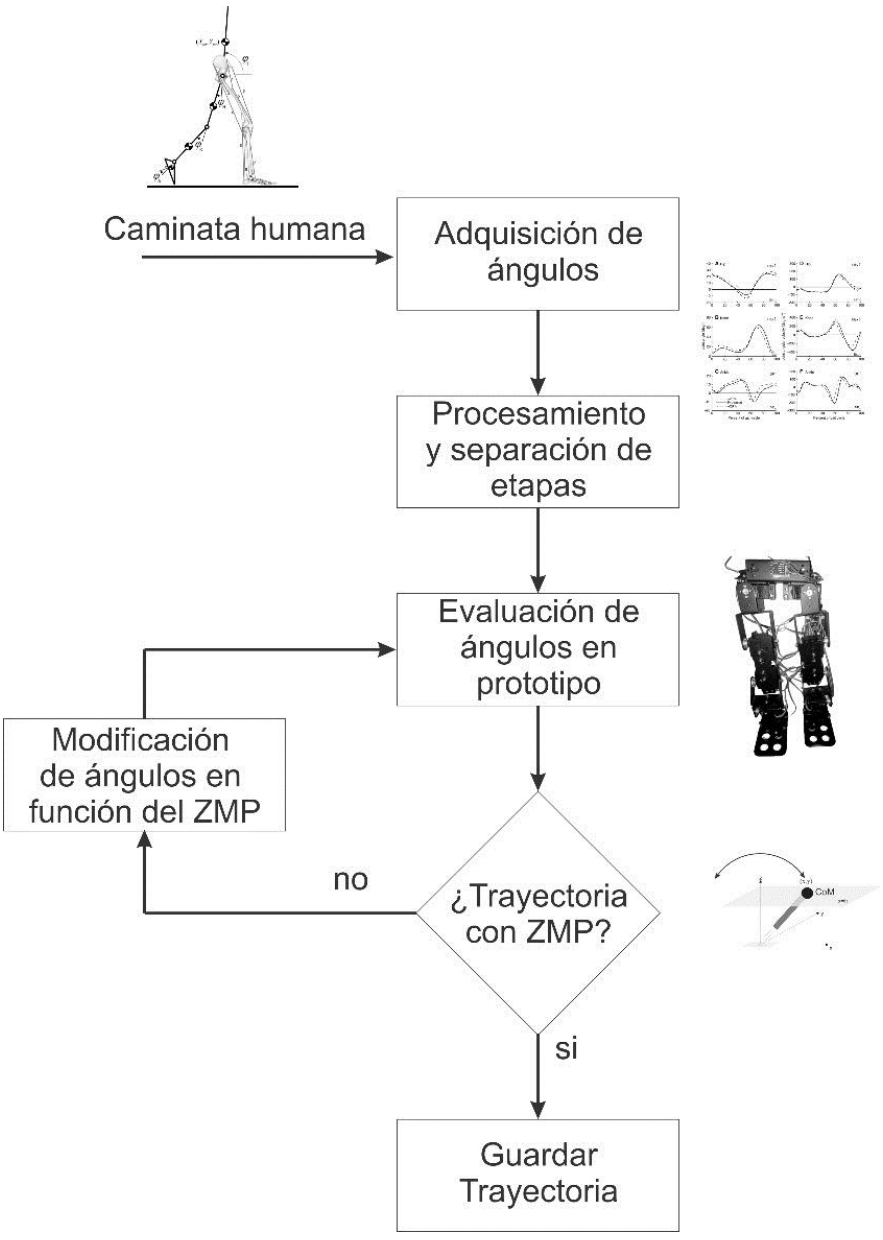

Figura 1.5. Diagrama de flujo para ajuste del ZMP

Cuando se obtiene la trayectoria con ZMP, la inercia horizontal es igual a cero asumiendo que la fricción con la superficie del suelo es lo

suficientemente alta para evitar deslizamientos (para más detalle sobre ZMP, revisar Vukobratović & Borovac, 2004). Con la trayectoria generada, el siguiente paso es aplicar la estrategia de control para realizar una acción correctiva entre el modelo aplicado y el bípedo real.

1.4 Bípedo de 10 diez grados de libertad

Para propósitos de prueba y simulación, se utilizó un robot con 10 grados de libertad (10 DOF) (Figura 1.6) y una simulación en MATLAB. Las dimensiones del robot se muestran en la Figura 1.7. Para la medición de los vínculos, se considera la distancia entre los centros de rotación respectivos.

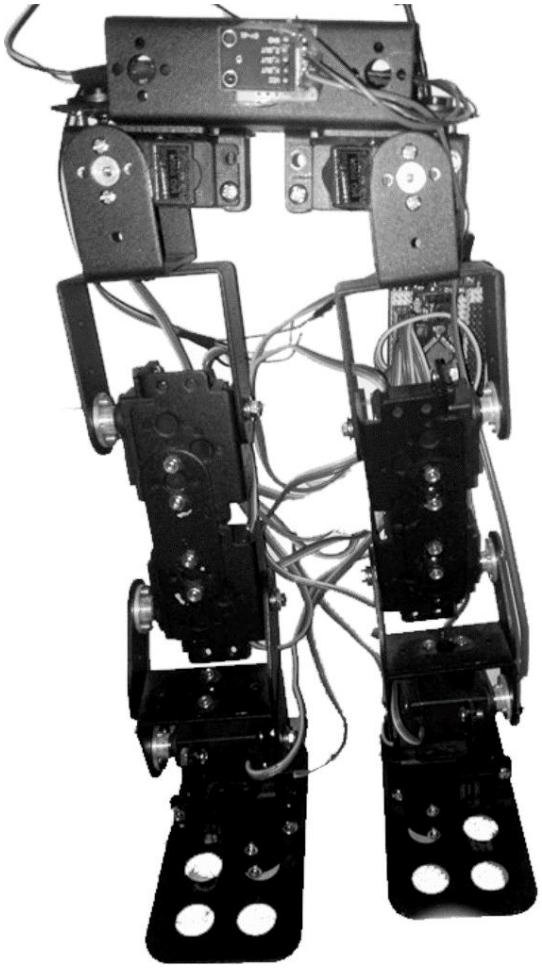

Figura 1.6. Robot bípedo

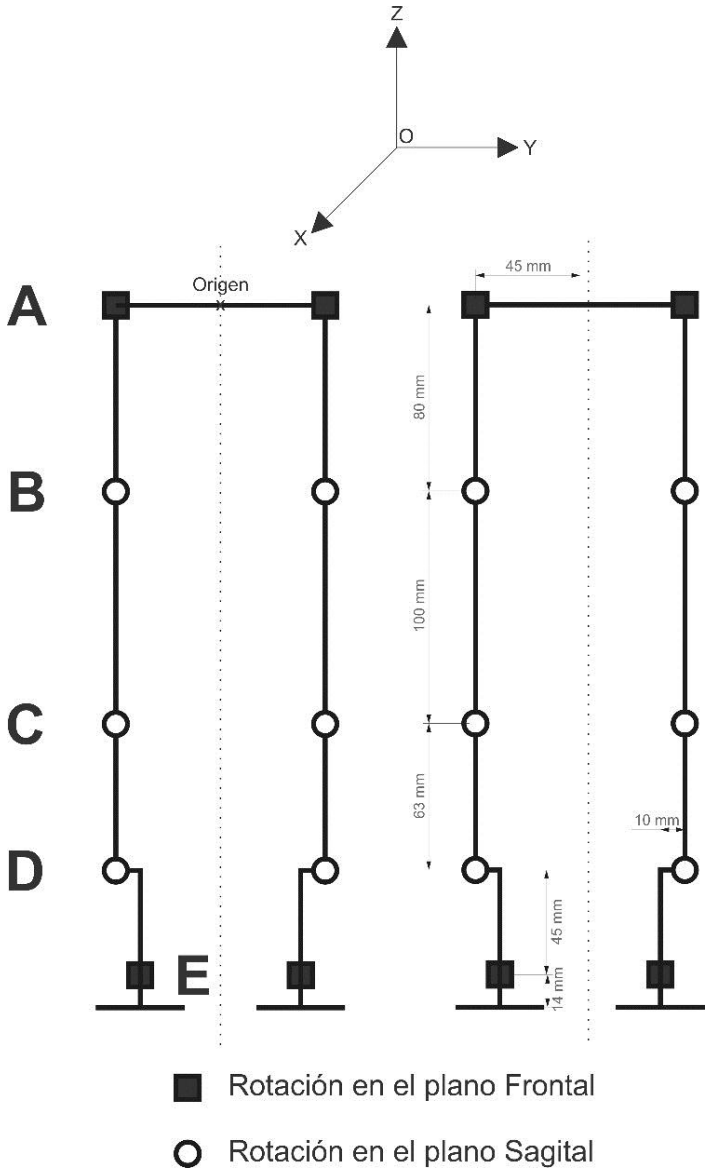

Figura 1.7. Dimensiones y vínculos del Bípedo

Por comodidad, para la asignación de los parámetros D-H de la Figura 1.7, primero se realizan ciertas rotaciones (para mayor detalle sobre los parámetros D-H, revisar Craig, J. J. ,2006). Los parámetros D-H se pueden visualizar en la Tabla 1.1. El análisis de cada pierna se realiza por separado para facilitar la solución cinemática del sistema, utilizando el centro de la cadera como el origen de coordenadas.

Tabla 1.1. Parámetros D-H para el bípedo

Vínculo	**θ**	**d (m)**$*$	**a (m)**	**α**
O	0	0	0	90
	90	$0.045 * k$	0	90
A	θ_A	0	-0.080	-90
B	θ_B	0	-0.100	0
C	θ_C	0	-0.063	0
D	θ_D	$-0.0105 * k$	-0.045	90
E	θ_E	0	-0.014	0
PP (Punta-pie)	0	0.06	0	0

$* k = 1$ para pierna derecha, $\mathrm{k} = -1$ para pierna izquierda

Se conoce también el CoM de cada vínculo y su ubicación respecto al origen (utilizando las matrices de transformación), siendo las coordenadas pertinentes (x_i, y_i, z_i) y masa m_i para $i = O, A, B, C, D, E$. Por lo tanto, el centro de CoM del sistema se puede calcular en tiempo real utilizando las ecuaciones (1.8), (1.9) y (1.10).

$$X_{CoM} = \frac{\Sigma_i(m_i * g * x_i)}{\Sigma_i(g * x_i)} \tag{1.8}$$

$$Y_{CoM} = \frac{\Sigma_i(m_i * g * y_i)}{\Sigma_i(g * y_i)} \tag{1.9}$$

$$Z_{CoM} = \frac{\Sigma_i(m_i * g * z_i)}{\Sigma_i(g * z_i)} \tag{1.10}$$

Los ángulo $\theta_B, \theta_C, \theta_D$ corresponden a rotaciones en el plano sagital, mientras que los ángulos θ_A, θ_E corresponden a rotaciones en el plano lateral.

Para las articulaciones, se utiliza el servo MG995 (Figura 1.8) debido a que es un motor bastante robusto con un torque de 9.40 Kg-cm a 4.8 V y 11.00 Kg-cm a 6 V con una rotación mayor a 180^{o} (TowerPro, Hoja técnica). Este modelo ha sido recomendado por el fabricante del bípedo.

Figura 1.8. Servo MG995 (TowerPro, Hoja técnica)

Para el control, se ha seleccionado un Raspberry Pi III (Figura 1.9) por su buen rendimiento en varias aplicaciones a nivel académico e industrial, como en tareas de automatización (Merchant & Ahire, 2017). Esta computadora se comunicará con un controlador de 32 servos (Figura 1.10) para dar posiciones a los diez servos del bípedo. El sensor que se usa para medir la aceleración e inclinación del bípedo es el giroscopio y acelerómetro MPU-6050 (Figura 1.11), este dispositivo está conectado a un microcontrolador de Microchip, el PIC16F886 (Figura 1.12), que también se comunica con el Raspberry PI III utilizando comunicación serial.

Figura 1.9. Raspberry Pi III

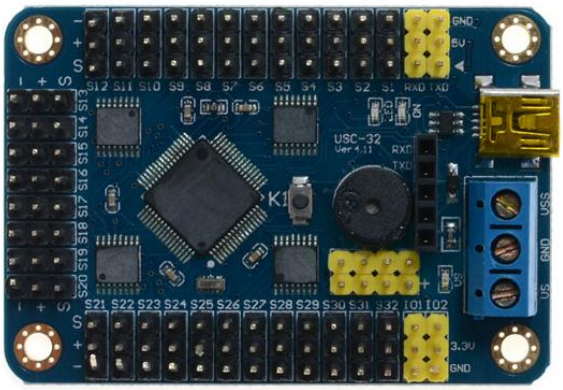

Figura 1.10. Controlador de 32 servos (Elechouse, Hoja técnica)

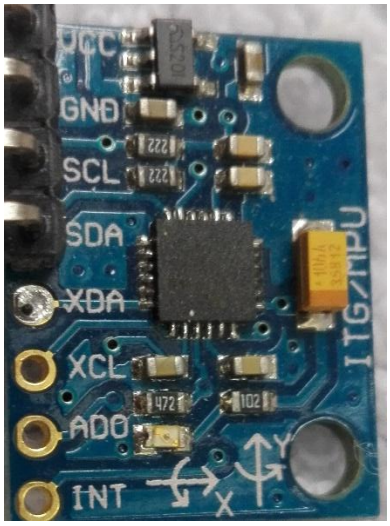

Figura 1.11. MPU-6050

Figura 1.12. PIC 16F886 de Microchip (Microcontrollers, 2007)

Una vez que se integran todos los elementos del bípedo, se procede a realizar la simulación en MATLAB, ajustando el ZMP con la trayectoria y estrategia de control para verificar su validez y posterior implementación en el bípedo real. El ZMP se utiliza como variable de control con el modelo LIPM y la implementación del controlador óptimo se detalla en la siguiente sección.

Capítulo II: MÉTODOS

En esta sección se detallan los mecanismos utilizados para la corrección y ajuste de los ángulos en la planificación de la trayectoria del bípedo. Con la trayectoria corregida, se procede a elaborar el diseño de un regulador óptimo. Se asume que el robot solo se desplaza en el plano sagital y que la

acción correctiva se aplica en el mismo plano, descartando los planos frontal y transversal. El modelo LIPM se describe junto con sus matrices de estado y las ecuaciones utilizadas para ajustar el controlador a la dinámica del sistema.

2.1 Generación de trayectoria

El centro de masa del robot se encuentra siempre en movimiento y no puede ser controlado directamente, sin embargo, se puede modificar el momento del bípedo para ajustar el ZMP. Primero, consideramos la simetría del bípedo y definimos los ángulos en la tabla 2.1 para ambas extremidades. El ciclo de caminata de la pierna derecha se aplica para la pierna izquierda con un desplazamiento temporal para cumplir las etapas mencionadas la sección 1.3. La duración del paso es de dos segundos, con 50 muestras (f=25 Hz).

Tabla 2.1. Ángulos de articulaciones del bípedo

Vínculo	**θ**	**Descripción (plano de rotación)**
O		Origen
A	θ_A	Ángulo de cadera, plano frontal
B	θ_B	Ángulo de cadera, plano sagital
C	θ_C	Ángulo de rodilla, plano sagital
D	θ_D	Ángulo de tobillo, plano sagital
E	θ_E	Ángulo de tobillo, plano frontal

Cuando el robot se desplaza, se considera el equilibrio en el plano frontal y el plano sagital. El balance en ambos planos se puede observar en la Figura 2.1. Al menos un pie se encuentra en contacto con el suelo y se utiliza para balancear el bípedo en el plano sagital y frontal.

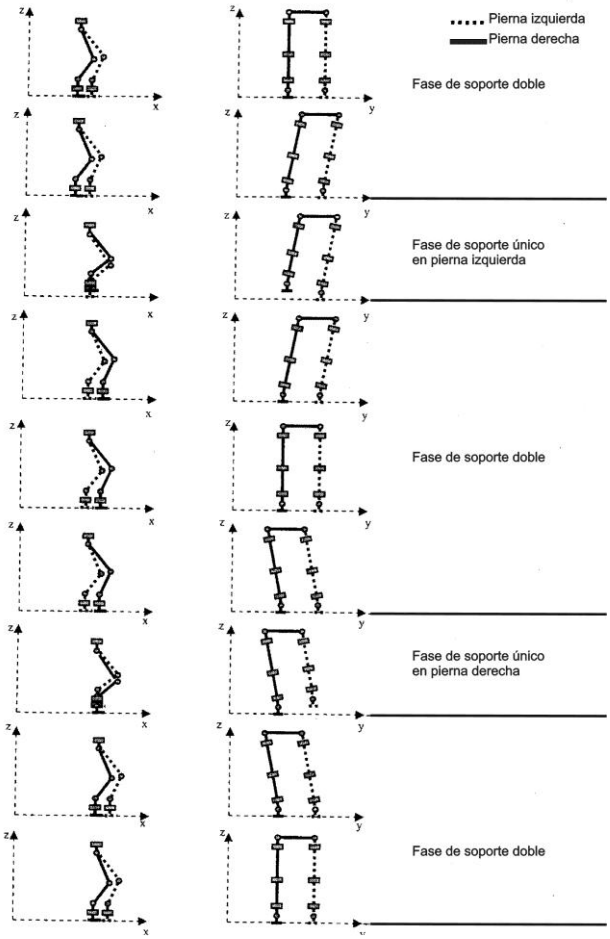

Figura 2.1. Ciclo de caminata (Weigang, 2009)

2.1.1 Equilibrio en el plano frontal

Para equilibrar el bípedo en el plano frontal, simplemente se alterna la posición de la pierna de soporte entre sus dos extremos corrigiendo los ángulos θ_A y θ_E(cadera y tobillo en el plano lateral) de la extremidad de

soporte cumpliendo la ecuación (2.1) y como se muestra en la parte derecha del ciclo de caminata de la Figura 2.1.

$$\theta_A = -\theta_E \tag{2.1}$$

En la Figura 2.2 se muestra el CoM y las transiciones de cada extremidad para cumplir con las dos fases de caminata en el plano frontal. El pie de soporte debe coincidir siempre con el CoM para la estabilización frontal.

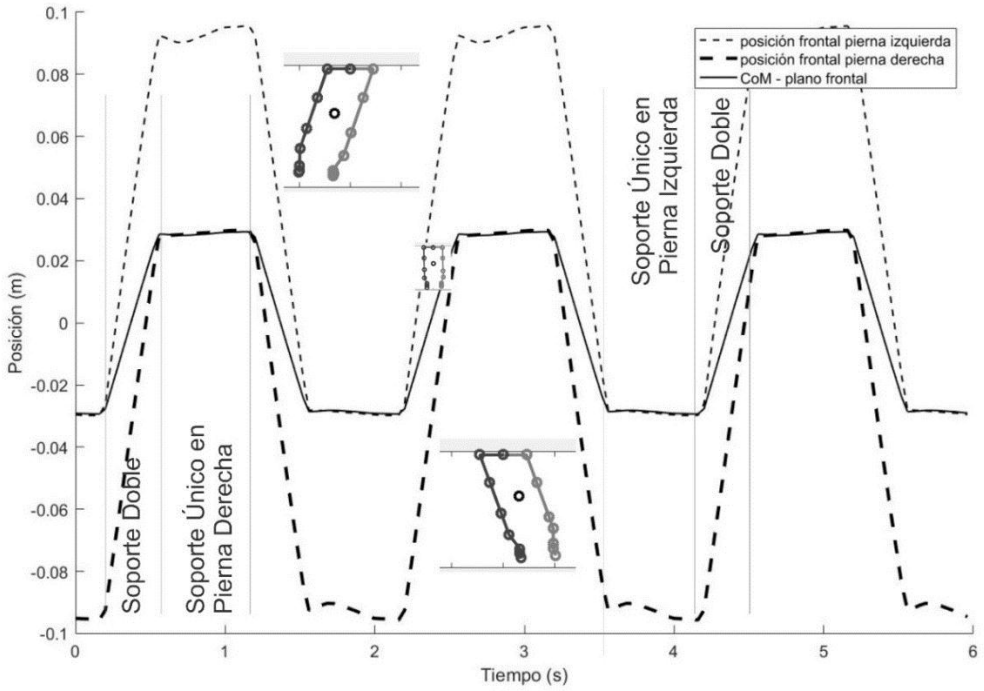

Figura 2.2. CoM y ajuste del pie de soporte en el plano frontal

Los ángulos del tobillo y cadera se van alternando con signos opuestos como se muestra en la Figura 2.3, de esta manera se estabiliza el bípedo en el plano frontal.

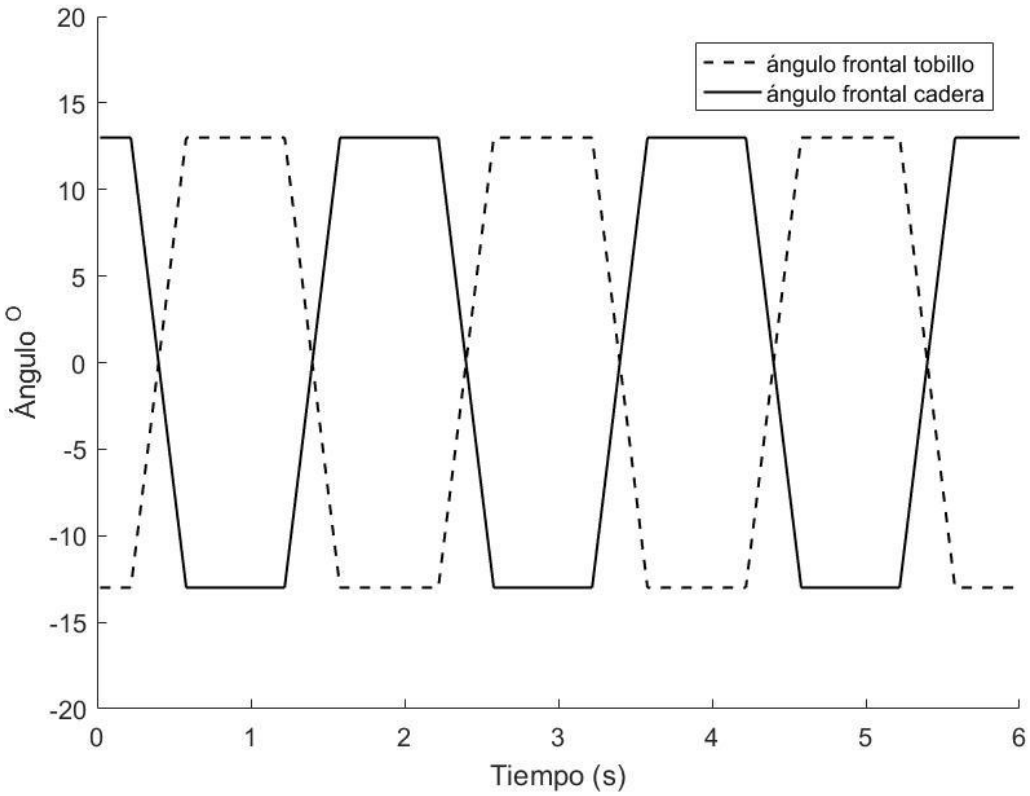

Figura 2.3. Ángulos de tobillo y cadera en el plano frontal

2.1.2 Equilibrio en el plano sagital

El equilibrio del robot en el plano sagital se logra corrigiendo primero el ángulo θ_B y después ajustando θ_D (cadera y tobillo en el plano sagital) de la pierna de soporte cumpliendo la ecuación (2.2) y ajustando la trayectoria de caminata.

$$\theta_D = -\theta_B - \theta_C \tag{2.2}$$

Para esta corrección se asume que el pie de soporte es un péndulo invertido y que se puede equilibrar el bípedo si se cambian los ángulos de cadera y

tobillo de la extremidad de soporte para alcanzar el ZMP siguiendo el modelo LIPM de la sección 1.2.

En esta sección, primero se muestran los ángulos de cadera, rodilla y tobillo en el plano sagital capturados por el DEET. Posteriormente, se muestra el CoM y ZMP antes de las correcciones respectivas. Después, se muestran ya los ángulos y trayectorias corregidas.

En la Figura 2.4 se muestran tres gráficas de los ángulos de cadera, rodilla y tobillo respectivamente antes de su modificación.

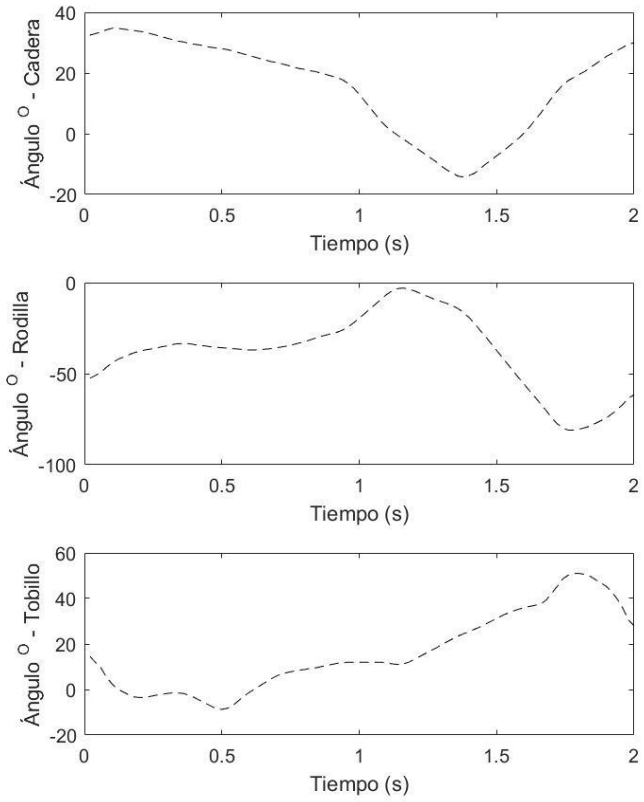

Figura 2.4. Ángulos de cadera, rodilla y tobillo antes de modificación

En la Figura 2.5 se muestra una gráfica con el CoM, el ZMP y la posición de cada pie a lo largo del ciclo de caminata. Como se observa, el centro de pie de soporte no está ni en el ZMP ni el CoM. Por esta causa, es necesario el

ajuste del momento del bípedo. Si no se realizase este ajuste, la dinámica sería más complicada de controlar.

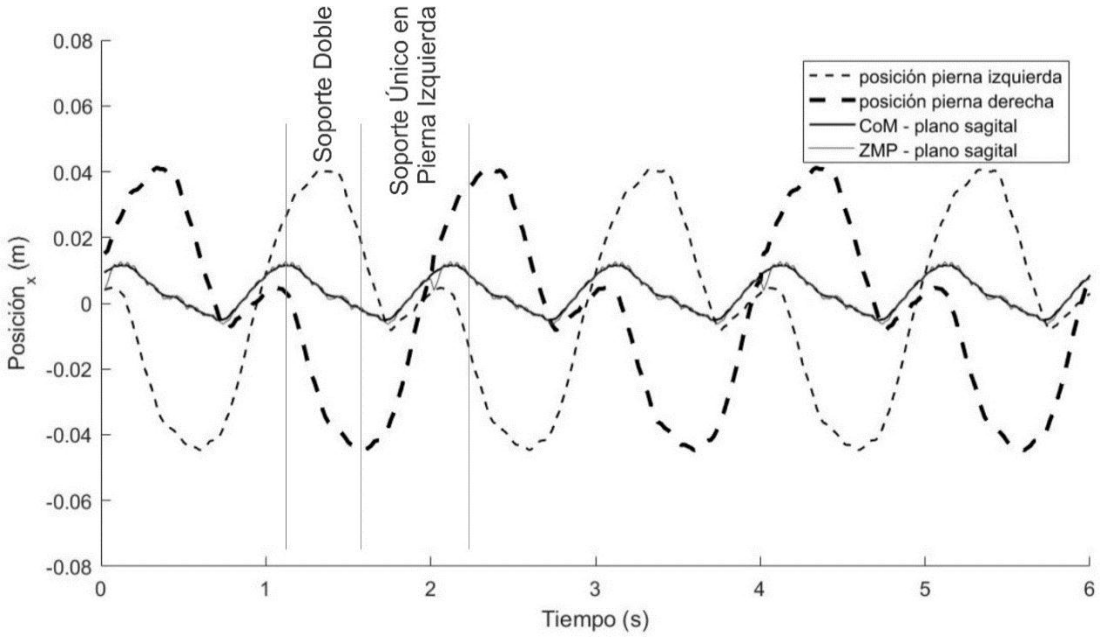

Figura 2.5. Posición sagital de pie, CoM y ZMP en un ciclo de caminata

Para ajustar la trayectoria, ambas etapas se manejan independientemente. Solo se modifican los ángulos del pie de soporte en la fase de soporte único. En la etapa de soporte doble se considera que el robot está estable con ambos pies sobre el suelo y no se realiza modificación alguna.

Los nuevos ángulos para la cadera y el tobillo en etapa de soporte único se obtuvieron a través de iteraciones considerando el menor valor del error cuadrático del ZMP esperado y el ZMP real. El valor de ZMP esperado se obtuvo siguiendo la ecuación 1.6 del modelo LIPM. Con esta consideración, los resultados de la corrección se muestran en la Figura 2.6.

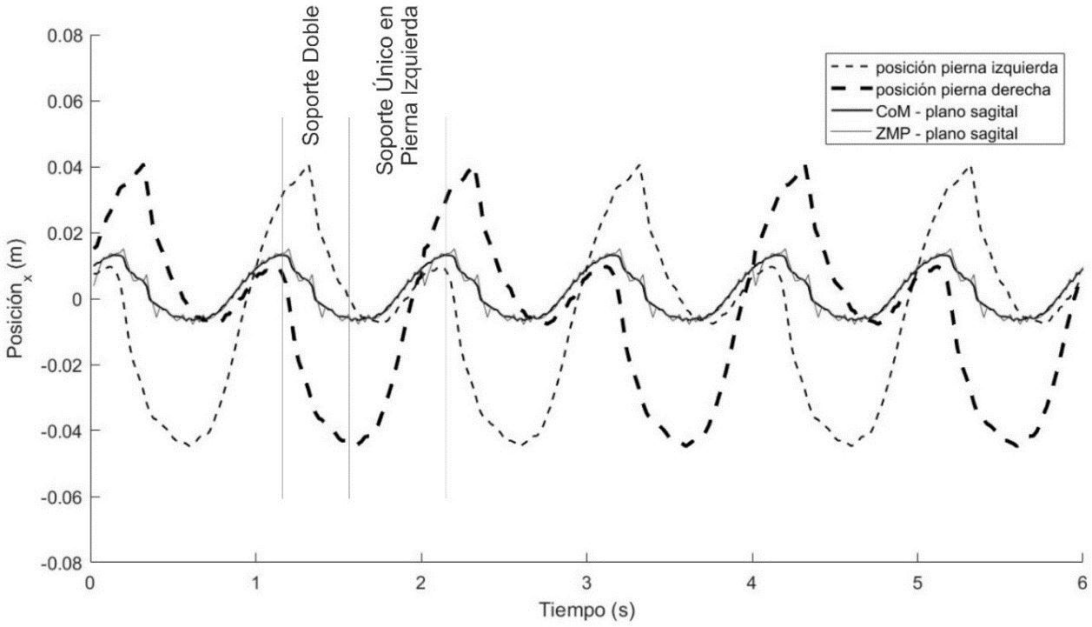

Figura 2.6. Posición sagital de pie, CoM y ZMP en un ciclo de caminata corregido

Favorablemente, con la corrección aplicada, ahora el ZMP está próximo al pie de soporte en la trayectoria. Se procede a utilizar el modelo LIPM para ajustar el controlador.

2.2 Estrategia de control

La trayectoria generada en función del ZMP y el modelo LIPM permiten estimar el comportamiento del sistema a lo largo del ciclo de operación. Para la estrategia de control, es necesario utilizar la información que se tiene de las futuras posiciones del bípedo. Además, se retroalimenta el sistema con los valores obtenidos del sensor utilizando un observador.

Para el controlador, se ha optado por utilizar un método de diseño basado en un servomecanismo para sistemas de tiempo discreto, sujeto una demanda que varía en el tiempo (trayectoria del ciclo de caminata). Este controlador tiene un integrador, retroalimentación y actuación que se anticipa el

comportamiento del bípedo. Estos parámetros son derivados aplicando la técnica de IQL (Integrador cuadrático lineal).

En el sistema, una regulación completa es posible en con el lazo cerrado incluso en la presencia de pequeñas perturbaciones, eliminando el efecto no deseado de la perturbación.

Un diagrama simplificado del sistema se puede ver en la Figura 2.7.

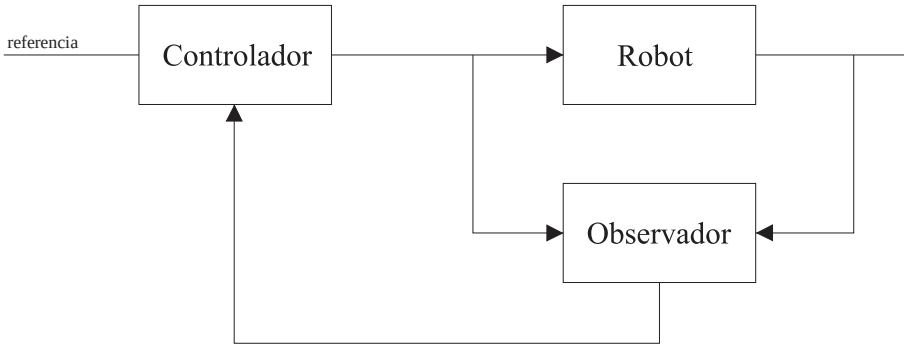

Figura 2.7 Esquema simplificado del sistema

2.2.1 Espacio de estados

El modelo LIPM puede reescribirse en su espacio de estados. La dinámica del sistema se obtiene directamente con las ecuaciones (1.3) y (1.4) y se muestran a continuación:

$$\frac{d}{dt}\begin{pmatrix} x \\ x' \\ p_x \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ \frac{g}{z_d} & 0 & -\frac{g}{z_d} \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} x \\ x' \\ p_x \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} v_x \tag{2.3}$$

$$A = \begin{pmatrix} 0 & 1 & 0 \\ \frac{g}{z_d} & 0 & -\frac{g}{z_d} \\ 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, C = (0\ \ 0\ \ 1), D = (0) \tag{2.4}$$

El vector de estados $(x\ x'\ p_x)^T$ contiene la posición actual del CoM y la posición estimada del ZMP $(\boldsymbol{p_x})$. La trayectoria planificada desde ahora se la llamará $\boldsymbol{p_{plan}}$ y la posición del ZMP medida por el sensor es $\boldsymbol{p_{sensor}}$. Cada ciclo de caminata tiene una duración de 2 segundos.

Con la simulación en MATLAB fue posible encontrar un valor medio para z_d, que es la posición del CoM desde el suelo (valor considerado positivo por el modelo LIPM). Se discretiza el sistema con un tiempo de muestreo de 40 ms, siendo los nuevos estados A_d, B_d. La variable de control es directamente p'_x. El vector real del sistema es $x(k)$ y el vector estimado es $x^*(k)$.

$$A_d = \begin{pmatrix} 1.0494 & 0.0407 & -0.0494 \\ 2.4902 & 1.0494 & -2.4902 \\ 0 & 0 & 1 \end{pmatrix}, B_d = \begin{pmatrix} -0.007 \\ -0.0494 \\ 0.04 \end{pmatrix}, C = (0\ \ 0\ \ 1), D = (0) \quad (2.5)$$

2.2.2 El observador y controlador óptimo

Una descripción detallada de este controlador se encuentra en (Katayama, Ohki, Inoue, & Kato, 1985). Para la retroalimentación y el observador se utiliza un método similar al descrito por (Czarnetzki, Kerner, & Urbann, 2010). El sistema completo se encuentra en la Figura 2.8.

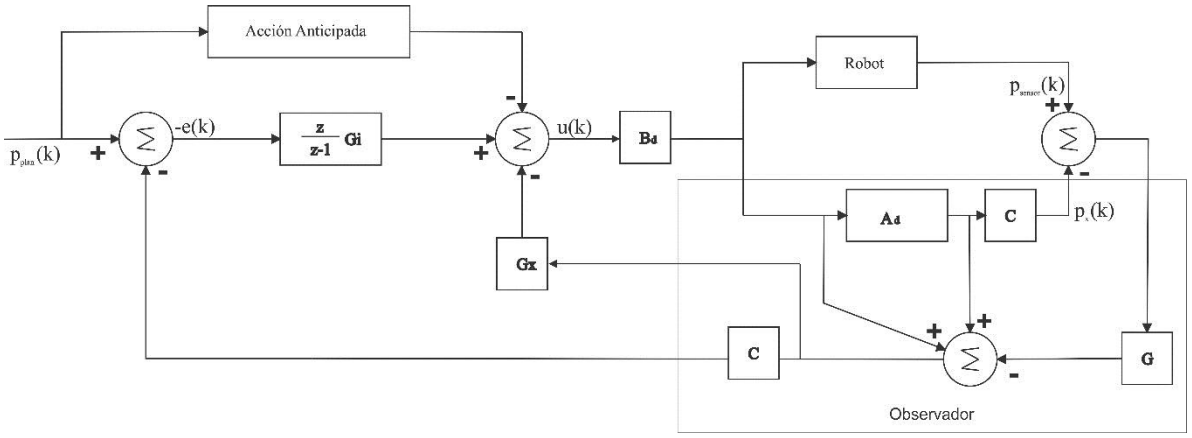

Figura 2.8. Diagrama de bloques del sistema

La extremidad en fase de soporte único debe moverse de manera que el ZMP sensado coincida con el ZMP planificado, de esta manera, el k-ésimo error está dado se muestra en la ecuación (2.6).

$$e(k) = p_{sensor}(k) - p_{plan}(k) \tag{2.6}$$

La ley de control $u(k)$ está dada por el controlador óptimo diseñado siguiendo (Katayama, Ohki, Inoue, & Kato, 1985) y se encuentra en la ecuación (2.7).

$$u(k) = -G_i \sum_{i=0}^{k} e(i) - G_x x^*(k) - \sum_{l=1}^{Nl} G_d(l)\, p_{plan}(k+l) \tag{2.7}$$

Para el controlador, Nl es el valor de ventana de las futuras predicciones, G_i, G_d, G_x son valores obtenidos de manera que optimicen el índice de rendimiento J expresado por la ecuación (2.8) y conocida como la ecuación de Ricatti. La matriz Q_e es diagonal simétrica positiva, el valor de R es positivo y Q_x es una matriz diagonal simétrica no negativa definida (que puede ser cero). En esta aplicación, los valores utilizados para R, Q_x y Q_e son $1x10^{-3}$, 0.00 y $1x10^{-1}$ respectivamente (una sola variable de control).

$$J = \sum_{i=1}^{\infty} e^T(i)\, Q_e e(i) + \Delta x^T(i)\, Q_x \Delta x(i) + R v^2(i) \tag{2.8}$$

De la ecuación (2.8), $\Delta x(i)$ es un vector incremental definido por:

$$\Delta x(i) = x(i) - x(i-1) \tag{2.9}$$

El valor del integrador para corregir el error de seguimiento está en la ganancia G_i multiplicada por la sumatoria del error, que en tiempo discreto se expresa como:

$$Integrador = \frac{z}{z-1} G_i \tag{2.10}$$

La ganancia proporcional para la retroalimentación del estado del modelo LIPM es G_x. La acción anticipada del controlador es la suma ponderada del ZMP planificado en función de la demanda futura. La ponderación variable

óptima de G_d se obtiene de la mejor solución de la ecuación de Ricatti. Las predicciones futuras están dadas por:

$$Acción\ Anticipada = \sum_{l=1}^{Nl} G_d(l)\, p_{plan}(k+l) \tag{2.11}$$

La ecuación del sistema completo se muestra a continuación:

$$x^*(k+1) = A_d x^*(k) + B_d u(k) + G(p_{sensor}(k) - Cx^*(k)) \tag{2.12}$$

El valor de G se encontró a través de una búsqueda lineal durante la etapa de simulación. La expresión $Cx^*(k)$ es el mismo $p_{plan}(k)$.

Para obtener las ganancias, primero se construyen las matrices del controlador óptimo siguiendo el método propuesto por (Katayama, Ohki, Inoue, & Kato, 1985).

Consideraciones iniciales:

- Vector de estado: $x(k)_{nx1}$ $[3x1]$
- Vector de control: $u(k)_{rx1}$ $[1x1]$
- Vector de salida: $p(k)_{sx1}$ $[1x1]$
- Matriz Identidad: I_s $[1x1]$

Construcción de Matrices:

$$B_p = \begin{bmatrix} C * B_d \\ B_d \end{bmatrix} \tag{2.13}$$

$$I_p = \begin{bmatrix} I_s \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.14}$$

$$F_p = \begin{bmatrix} C * A_d \\ A \end{bmatrix} \tag{2.15}$$

$$Q_p = \begin{bmatrix} Q_e & 0 \\ 0 & Q_x \end{bmatrix} \tag{2.16}$$

$$A_p = [I_p \quad F_p] \tag{2.17}$$

Los valores de R, Q_x y Q_e ya son conocidos. Una vez construidas las matrices, se soluciona la ecuación de Ricatti (K_p) que se muestra en la ecuación (2.16)

$$K_p = A_p^T K_p A_p - A_p^T K_p B_p \left[R + B_p^T K_p B_p\right]^{-1} B_p^T K_p A_p + Q_p \tag{2.18}$$

La solución fue obtenida con métodos numéricos convencionales. Ahora, con K_p ya es posible encontrar los valores de G_i, G_d, G_x con las ecuaciones (2.19), hasta la (2.25).

$$G_i = \left[R + B_p^T K_p B_p\right]^{-1} B_p^T K_p I_p \tag{2.19}$$

$$G_x = \left[R + B_p^T K_p B_p\right]^{-1} B_p^T K_p F_p \tag{2.20}$$

$$A_c = A_p - B_p \left[R + B_p^T K_p B_p\right]^{-1} B_p^T K_p A_p \tag{2.21}$$

$$x_p(1) = -A_c^T K_p I_p \tag{2.22}$$

$$x_p(l) = -A_c^T x_p(l-1) \; dónde \; l = 2,3,\dots,Nl \tag{2.23}$$

$$G_d(1) = -G_i \tag{2.24}$$

$$G_d(l) = \left[R + B_p^T K_p B_p\right]^{-1} x_p(l-1) \quad dónde \; l = 2,3,\dots,Nl \tag{2.25}$$

Los valores obtenidos de las ganancias se muestran en la tabla 2.1

Tabla 2.1 Ganancias G_i, G_x

Ganancia	Valor
G_i	(-4.6424)
G_x	$(-70.3755 \quad -8.9922 \quad 29.5628)$

El valor de G_d para el observador bajo demanda futura es variable y se presenta en la Figura 2.9. El número de futuras predicciones que se utilizó es 18 (Nl). No existe mejora alguna con un número más alto de predicciones.

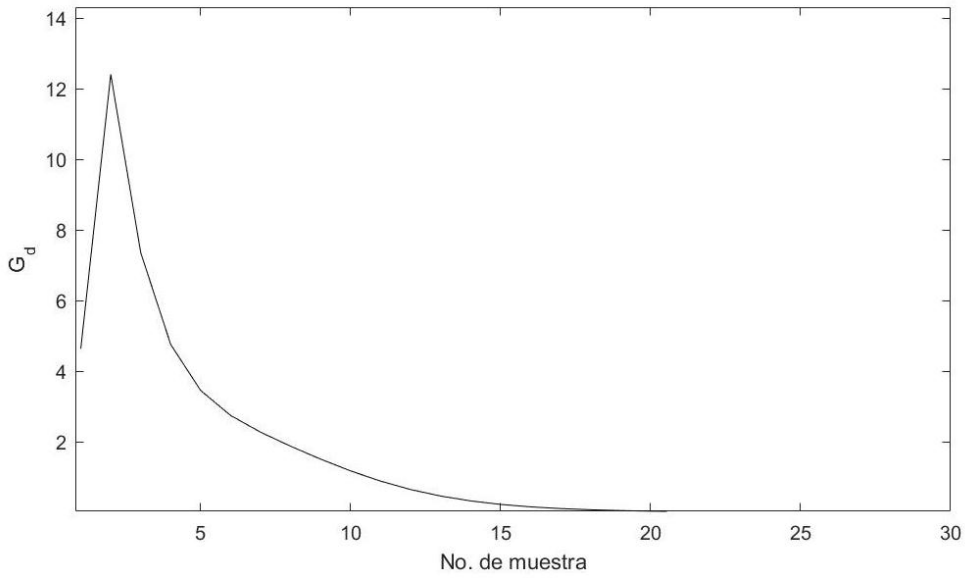

Figura 2.9. Valores de G_d vs No. de muestra.

Con los valores de las ganancias y el controlador sintonizado al modelo LIPM, se procede a la implementación, tanto en simulación en MATLAB como en el bípedo mostrado en la sección 1.4. Previo a esto, se tiene la trayectoria final basada en el ZMP en la figura 2.10. Esta trayectoria es una versión que pasa por un filtro de promediado móvil con ventana de 3 muestras. El objetivo de este filtrado es suavizar la señal y eliminar el posible ruido existente.

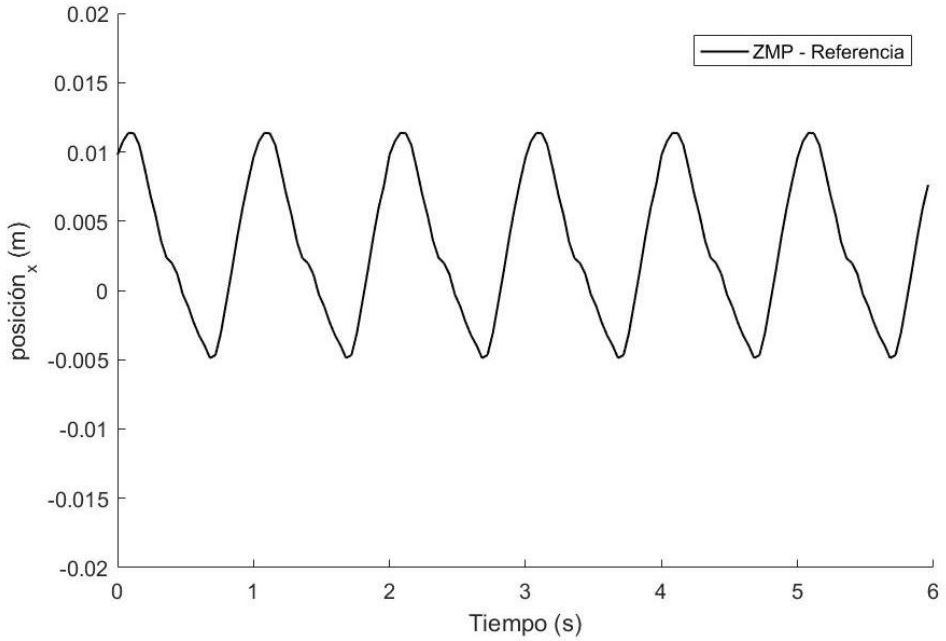

Figura 2.10. ZMP de referencia

La nueva referencia se utiliza para toda la implementación de aquí en adelante.

Capítulo III: IMPLEMENTACIÓN

La implementación se realiza en dos etapas: software y hardware. Primeramente, se explica la construcción del programa de simulación y los parámetros necesarios para predecir el comportamiento del bípedo a lo largo de la trayectoria en MATLAB. Después, con la simulación funcional, la configuración de hardware se detalla.

3.1 Simulación en MATLAB

Los parámetros conocidos del bípedo permitieron desarrollar una interfaz que simule el ciclo de caminata del robot y así obtener los valores fundamentales para su control: CoM y ZMP. Las variables utilizadas para cada articulación en la simulación son las siguientes:

- Parámetros D-H.
- Masa del vínculo.
- Posición relativa del CoM.

El centro de la cadera se considera el origen coordenado del sistema global S_o. A continuación, se describe la cinemática del robot utilizando las variables mencionadas.

3.1.1 Cinemática directa

Cada vínculo tiene una matriz de transformación que surge de los parámetros D-H, a esta matriz la llamaremos M_i^{i-1} para el i-ésimo vínculo.

Además, construimos la matriz de transformación con la posición del CoM de cada vínculo relativa a su propio sistema coordenado, cuyo asignación será C_i^{i-1}.

Utilizamos la ecuación (3.1) para encontrar la orientación O_i (matriz de 3x3) y la posición P_i (vector de 3x1) del i-ésimo vínculo desde el origen del sistema coordenado S_o.

$$T_i = M_o M_1^o M_2^1 \dots M_i^{i-1} \tag{3.1}$$

De la expresión (3.1) tomamos el vector de posición P_i que se muestra en la transformada T_i de la expresión (3.2).

$$T_i = \begin{bmatrix} O_i & P_i \\ 0 & 1 \end{bmatrix} \tag{3.2}$$

La posición del i-ésimo vínculo está dado por las coordenada $(x_i, y_i, z_i)^T$.

Similarmente, la posición del CoM de cada vínculo desde el origen del sistema coordenado S_o es:

$$C_i = M_o M_1^o M_2^1 \dots C_i^{i-1} \tag{3.3}$$

De la expresión (3.3) tomamos el vector de posición P_i^C que se muestra en la transformada T_i^C de la expresión (3.2).

$$C_i = \begin{bmatrix} O_i^C & P_i^C \\ 0 & 1 \end{bmatrix} \tag{3.4}$$

La posición del i-ésimo CoM está dado por las coordenada $(x_i^c, y_i^c, z_i^c)^T$.

Es importante recordar que después del vínculo ‘E’ mencionado en la sección (1.4), también se considera la punta del pie con su propio parámetro D-H para mantener el formato y realizar la gráfica en la simulación. Por otro lado, los ángulos $\theta_A, \theta_B, \theta_C, \theta_D, \theta_F$ requeridos para cada paso en la solución cinemática fueron capturados por el DEET y posteriormente modificados con la ayuda de la simulación creada en esta sección. La modificación se detalla

el capítulo II de este documento y la solución a la cinemática inversa para las trayectorias se muestra más adelante en este capítulo.

Debido a la simetría del bípedo, un índice k es añadido a la nomenclatura para diferencia las matrices de transformación de los vínculos de la pierna izquierda y derecha. Se crea una función que devuelva la posición de cada vínculo, utilizando los ángulos cómo valor de entrada, de la siguiente manera:

$$\left(T_A^k, T_B^k, T_C^k, T_D^k, T_E^k, T_{PP}^k\right) = pierna(\theta_A, \theta_B, \theta_C, \theta_D, \theta_F, k)$$

$$donde \begin{cases} k = 1 \text{ para pierna derecha} \\ k = -1 \text{ para pierna izquierda} \end{cases} \tag{3.5}$$

La posición del CoM viene de reemplazar la ecuación (3.1) en (3.3). La multiplicación resultante sería:

$$C_i^k = T_i^k C_i^{i-1} \tag{3.6}$$

La visualización del ciclo de caminata se realiza en los tres planos representados en la figura (3.1) junto con el CoM en cada posición.

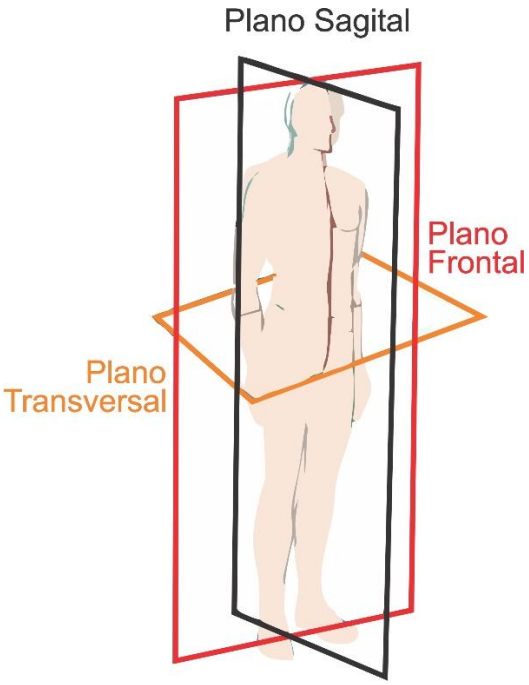

Figura 3.1. Planos en la simulación

A continuación, se muestran capturas de los resultados de simulación en las dos etapas de caminata.

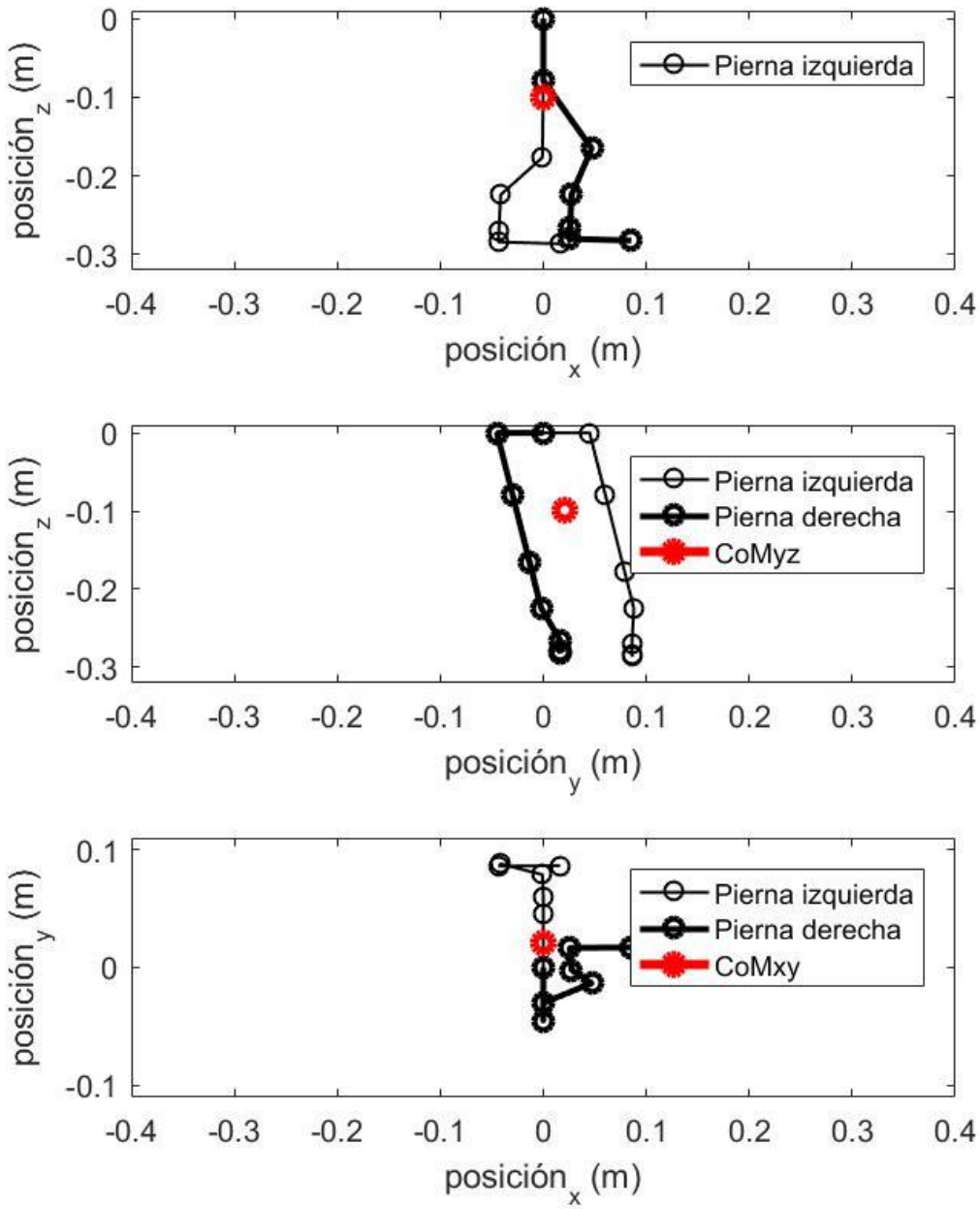

Figura 3.2. Simulación, pierna izquierda como soporte único

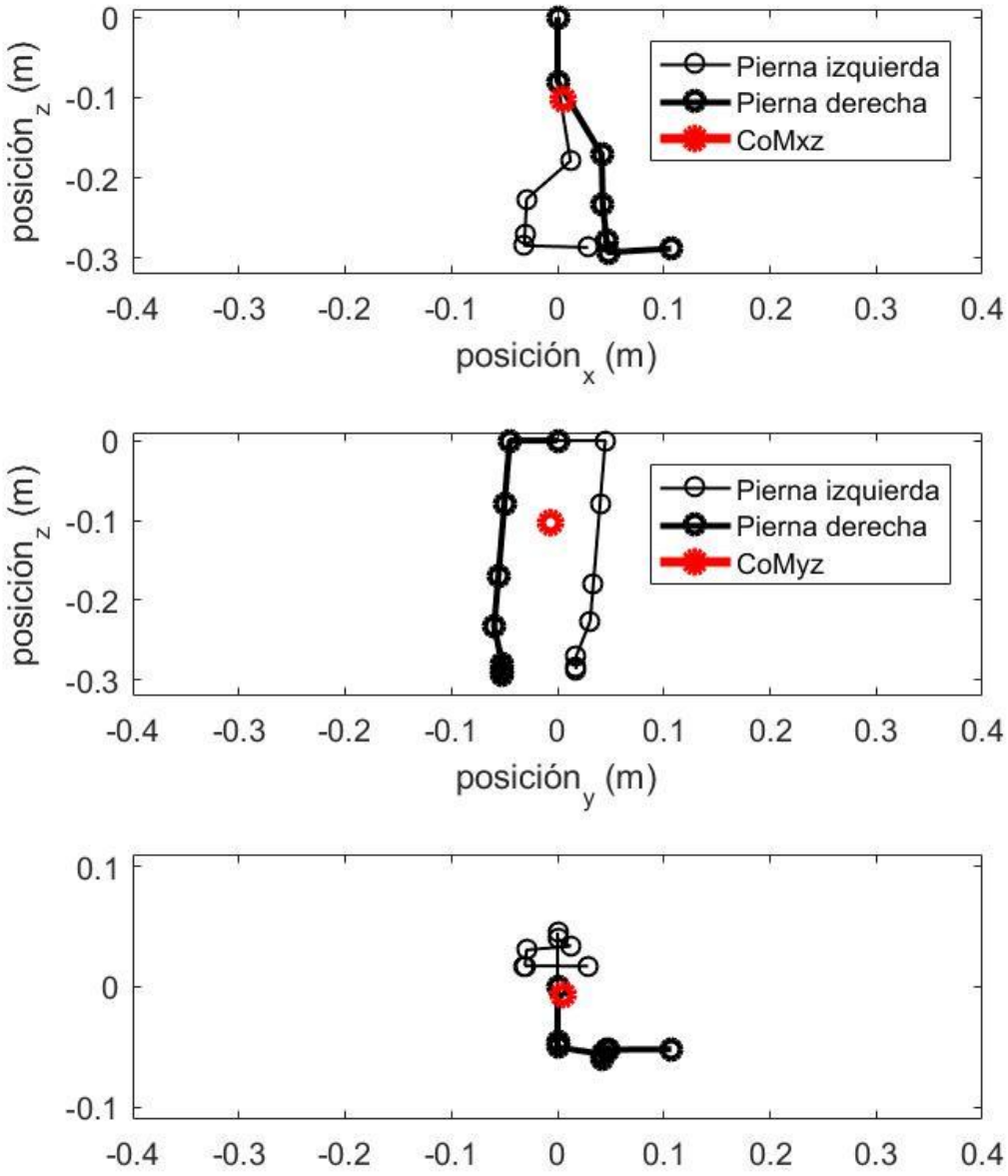

Figura 3.3. Simulación, soporte doble

Con la simulación se puede observar las posiciones del actuador final, obtener el CoM y calcular el ZMP a partir de los ángulos. Las modificaciones de la trayectoria para acercar el ZMP al CoM y cumplir con el ZMP dentro del área del polígono soporte se realizaron en base a los datos obtenidos de la simulación con la cinemática directa. Sin embargo, para obtener los nuevos ángulos a partir de las nuevas trayectorias generadas, se utilizan métodos numéricos en la solución de la cinemática inversa, como se detalla a continuación.

3.1.2 Cinemática inversa

En esta sección se explica el procedimiento utilizado para resolver el problema de cinemática inversa. Sin una correcta solución, no sería posible seguir la trayectoria planificada o corregir las posiciones de la pierna en etapa de soporte único, de manera que la proyección CoM esté dentro del polígono de soporte. Debido a la extensión de las ecuaciones, se optó por utilizar métodos numéricos para encontrar una solución.

A continuación, se muestran las matrices de transformación $M_o, M_A^o, \dots, M_E^D$ para los vínculos del robot, considerando $k = 1$ para pierna derecha, $k = -1$ para pierna izquierda, $c = \text{coseno}$ y s= seno .

$$M_o = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & -0.045k \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$M_A = \begin{bmatrix} c\theta_A & 0 & -s\theta_A & -0.08c\theta_A \\ s\theta_A & 0 & c\theta_A & -0.08s\theta_A \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

$$M_B = \begin{bmatrix} c\theta_B & -s\theta_B & 0 & -0.1c\theta_B \\ s\theta_B & c\theta_B & 0 & -0.1s\theta_B \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.9)$$

$$M_C = \begin{bmatrix} c\theta_C & -s\theta_C & 0 & -0.063c\theta_C \\ s\theta_C & c\theta_C & 0 & -0.063s\theta_C \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.10)$$

$$M_D = \begin{bmatrix} c\theta_D & 0 & s\theta_D & -0.045c\theta_D \\ s\theta_D & 0 & -c\theta_D & -0.045s\theta_D \\ 0 & 1 & 1 & -0.015k \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.11)$$

$$M_E = \begin{bmatrix} c\theta_E & -s\theta_E & 0 & -0.014c\theta_E \\ s\theta_E & c\theta_E & 0 & -0.014s\theta_E \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.12)$$

Las coordenadas (x_E^k, y_E^k, z_E^k) de la posición del final del actuador E se convierten en elemento de interés, ya que es el punto de soporte que se considera como soporte en la planificación de la trayectoria.

Debido a que en nuestro análisis para el control se limita al plano sagital, las variables de entrada para la nueva asignación de ángulos $\theta_{B,}\theta_{C,}\theta_D$ son las coordenadas x_E^k y z_E^k del pie.

Las ecuaciones para ambas piernas (ecuaciones (3.13) y (3.14)) que deben ser resultas para satisfacer x_E^k y z_E^k son iguales debido a que no se considera y_E^k y que el bípedo es simétrico.

$$x_E^k = 0.1 * sin(\theta_B) +$$

$$0.059 * cos(\theta_D) * \big(cos(\theta_B) * sin(\theta_C) + cos(\theta_C) * sin(\theta_B)\big) +$$

$$0.059 * sin(\theta_D) * \big(cos(\theta_B) * cos(\theta_C) - 1.0 * sin(\theta_B) * sin(\theta_C)\big) +$$

$$0.063 * cos(\theta_B) * sin(\theta_C) + 0.063 * cos(\theta_C) * sin(\theta_B) \quad (3.13)$$

$$z_E^k = 0.059 * sin(\theta_D) * \left(cos(\theta_B) * sin(\theta_C) + cos(\theta_C) * sin(\theta_B)\right) -$$

$$0.1 * cos\left(\theta_{B,}\right) -$$

$$0.059 * cos(\theta_D) * \left(cos(\theta_B) * cos\left(\theta_{C,}\right) - 1.0 * sin(\theta_B) * sin(\theta_C)\right) -$$

$$0.063 * cos(\theta_B) * cos(\theta_C) + 0.063 * sin(\theta_B) * sin(\theta_C) - 0.08 \qquad (3.14)$$

3.1.3 Algoritmo de Levenberg–Marquardt

Con el sistema de ecuaciones de cinemática de la sección anterior, se procede a utilizar métodos numéricos para encontrar la solución. El algoritmo utilizado es el de Levenberg-Marquardt (LM). Las operaciones y el tiempo de convergencia del algoritmo no presentan inconvenientes para la simulación, sin embargo, para el bípedo real se implementó un mapa de soluciones x_E^k y z_E^k ordenadas y accesibles por búsqueda binaria.

El algoritmo de Levenberg-Marquardt es un mecanismo que en general se utiliza para ajustar curvas a través mínimos cuadrados con la combinación de dos algoritmos: algoritmo Newton-Euler y método de descenso del gradiente (Lourakis, 2005).

Las ventajas de utilización de este método en la solución numérica de la cinemática inversa se detallan en (Sugihara, 2009) y propone su utilización para la marcha humana en robots bípedos. Entre estas ventajas tenemos:

- Las singularidades y la solvencia de la cinemática inversa no interfieren con la aplicación del método.
- En el caso de que no haya solución o existan múltiples soluciones, el método converge a la solución óptima.

En el caso de esta aplicación, se intenta encontrar una solución a un sistema no lineal a través de iteraciones sucesivas que localizan el mínimo de una función expresada como una suma de cuadrados. La solución encontrada es un mínimo local de esta función, pero no necesariamente el mínimo global.

Las restricciones de los ángulos $\theta_{B,}\theta_{C,}\theta_{D}$ utilizadas para la convergencia del algoritmo son:

$$-20 < \theta_B < 40$$
$$-86 < \theta_C < 0$$
$$-15 < \theta_D < 60$$

3.1.4 Control por fases de caminata

El siguiente paso para controlar la trayectoria es la fragmentación de una de las etapas de caminata. En el capítulo 1 se describieron las fases de soporte único y soporte doble. Sin embargo, para la aplicación del controlador a lo largo del ciclo, es necesario dividir la fase en soporte doble en etapa de soporte mayormente trasero y etapa de soporte mayormente delantero.

Para controlar el bípedo en la etapa de soporte único, la acción correctiva se realiza sobre el pie de soporte. Por otra parte, para el control en la etapa de soporte doble, primero se tiene las siguientes consideraciones para cada paso que da el bípedo:

- Al iniciar la etapa de soporte doble (línea A, Figura 3.4), la pierna que estuvo en soporte único (pie izquierdo, Figura 3.4) recibe mayor peso debido a que el bípedo se sostenía sobre esa articulación. Sin embargo, la pierna contraria (pie derecho, Figura 3.4) está avanzando con un paso hacia adelante, mientras que la que estuvo en soporte

único se empieza a colocar en el eje negativo de x (por detrás de la cadera). Por tanto, esta primera sección se llama soporte mayormente posterior.

- Transcurrido aproximadamente el 50% de la etapa de soporte doble (línea B, Figura 3.4), la pierna que se desplazaba para dar el paso (pie derecho, Figura 3.4) se empieza a acercar al CoM y el bípedo tiene más soporte en esa pierna. Por esta razón, esta segunda sección se llama soporte mayormente delantero.

- Completado el ciclo de soporte doble (línea C, Figura 3.4), el bípedo pasa a soporte único nuevamente.

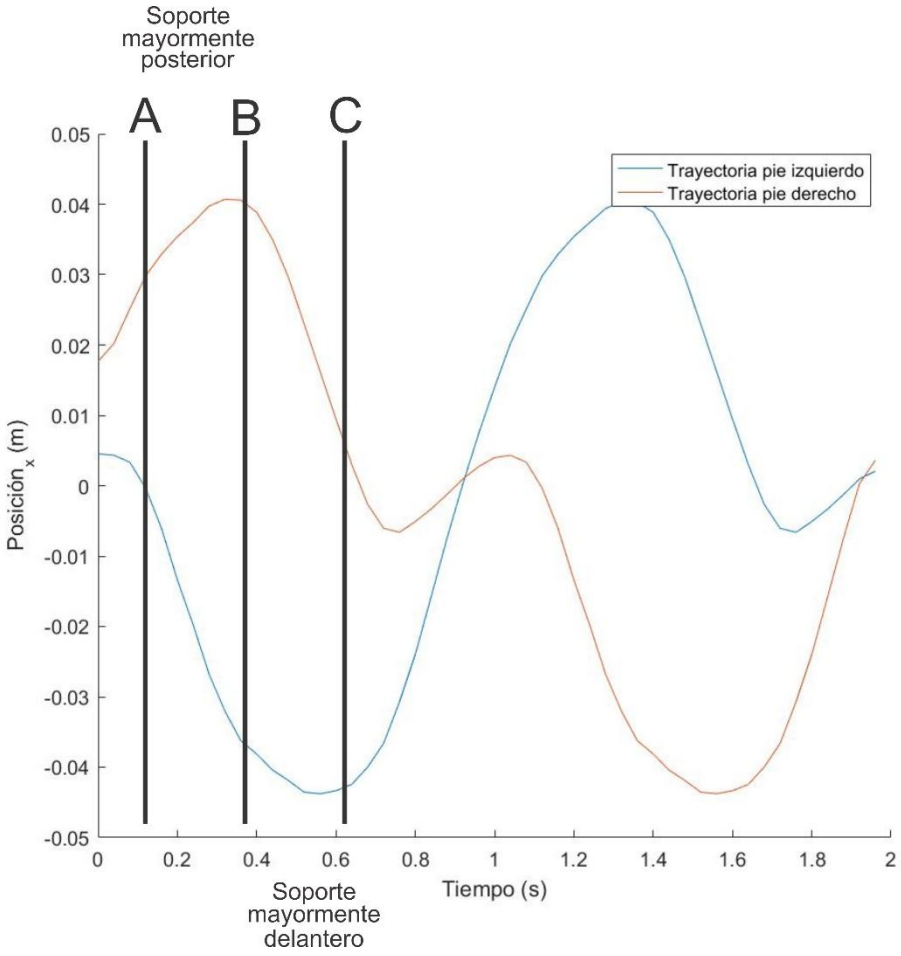

Figura 3.4. Trayectorias de pie derecho e izquierdo

Con las dos subsecciones de la etapa de soporte doble, la acción correctiva está primeramente en el pie de soporte mayormente posterior y después en el pie de soporte mayormente delantero.

Finalmente, la simulación con las ecuaciones de cinemática y los elementos involucrados en el control con las estrategias correctivas mencionadas hasta este momento se programaron en MATLAB.

3.2 Implementación funcional del bípedo

Los elementos utilizados para la implementación del bípedo se describieron en la sección 1.4, sin embargo, no se mencionó detalle alguno sobre las configuraciones. En esta sección se muestra cómo se puso en funcionamiento el bípedo y las consideraciones necesarias para el correcto desempeño del robot.

Un diagrama general de operación se muestra en la Figura 3.5.

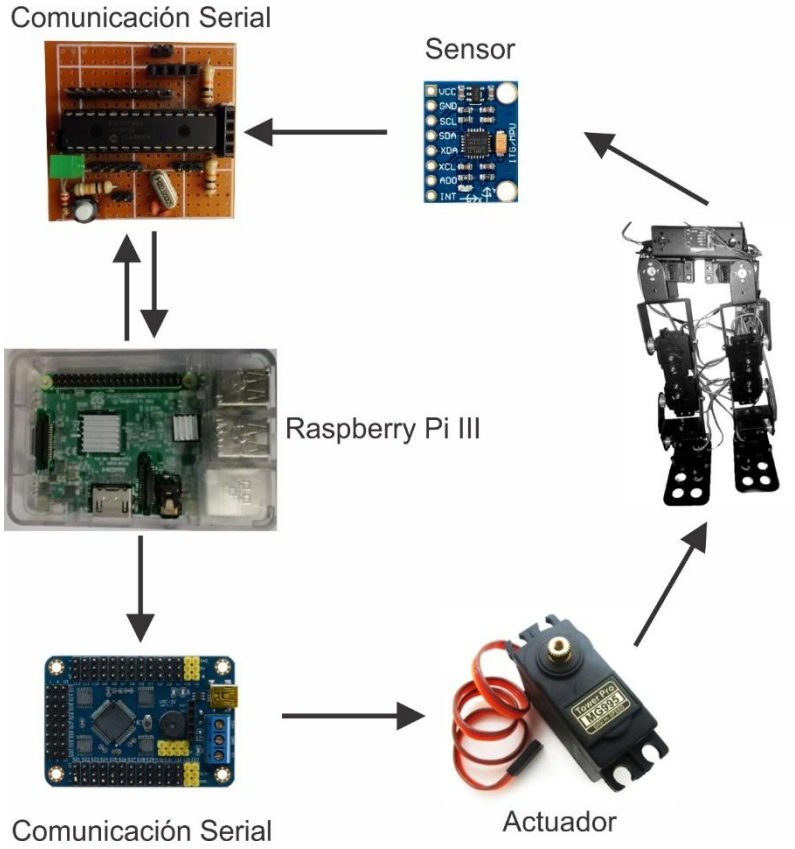

Figura 3.5. Esquema general de operación

El lenguaje de programación que se utilizó para integrar todos los elementos es Python™ con el uso de interfaces seriales. El programa de control se ejecuta desde el Raspberry Pi III.

El primer requerimiento para operar el robot es controlar los servos. Para ubicar el acanalado en la posición angular deseada, se genera un PWM con frecuencia de 50 Hz y un ciclo de trabajo que habitualmente varía desde 1 ms para 0^o a 2 ms para 180^o (Figura 3.6).

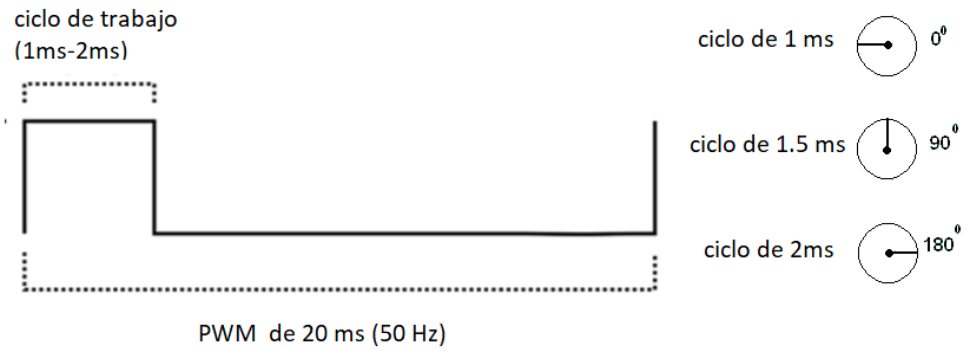

Figura 3.6. PWM de cada servo

La tarea de generar este PWM para operar los servos fue realizada por el driver de 32 canales que recibe comandos por comunicación serial. Para el caso de esta aplicación, todos los servos se inicializaron con una rotación de 90^o y después se colocaron en el bípedo en posición firmes, mapeando 90^o a un ángulo de 0^o en la articulación. Esto quiere decir que cuando un comando genera una rotación de 45^o en realidad está rotando de 90^o+45^o. De esta manera es posible tener ángulos positivos y negativos para pasar de manera directa los ángulos de la simulación a los servos del bípedo real. A continuación, se explica cómo fueron generados los comandos.

3.2.1 Comandos para el driver de 32 canales

El controlador permite rotar uno o varios servos a la vez en un mismo comando. Para generar un comando, se definen los siguientes parámetros que actuarán en el n-ésimo canal (ch_n):

- ct_n – ciclo de trabajo en ms.
- td_n – tiempo de rotación en ms.

- α_n – ángulo.
- r_n – rotación inicial en ms.

Otros parámetros globales necesarios son:

- δ – factor de mapeo.
- tg – tiempo de rotación global en ms.
- tl – ciclo de trabajo mínimo en ms que representa 0^o .
- tu – ciclo de trabajo máximo en ms que representa 180^o.

Primero, se calcula el factor de mapeo:

$$\delta = \frac{tu - tl}{2 * 90} \tag{3.15}$$

Después, se encuentra el ciclo de trabajo:

$$ct_n = r_n + \alpha_n * \delta \tag{3.16}$$

El comando para rotar el n-esimo servo sería:

$$\#\boldsymbol{ch_n}\mathrm{P}\boldsymbol{ct_n}\mathrm{T}\boldsymbol{td_n}\backslash r\backslash n$$

Si se desea rotar varios servos con un mismo tiempo de rotación, entonces el comando sería:

$$\#\boldsymbol{ch_n}\mathrm{P}\boldsymbol{ct_n}\#\boldsymbol{ch_m}\mathrm{P}\boldsymbol{ct_m}\#\boldsymbol{ch_o}\mathrm{P}\boldsymbol{ct_o}\ \ldots\ \mathrm{T}\boldsymbol{tg}\backslash r\backslash n$$

En el caso de esta aplicación, se utilizaron los siguientes valores globales:

Tabla 3.1. Valores globales

Parámetro	Valor
tl_n	900
tu_n	2100
δ	6.66
tg	40

Por ejemplo, si se deseara mover el servo 2 y 3 un ángulo de 20° y -30° respectivamente en un intervalo de 40 ms, entonces el comando sería el siguiente (considerando $r_n = 1500$):

$$\#2\mathrm{P}1633\#3\mathrm{P}1300\mathrm{T}40\backslash r\backslash n$$

3.2.2 Cálculo del error con giroscopio

Conocer los parámetros utilizados en simulación permitieron reducir el número de sensores que se necesitan para el cálculo del error.

Para encontrar el error, se mostró en la sección 2.2.2 que se necesita saber la posición del ZMP para restarla de la referencia. Con esto en mente, se utilizó la ecuación (1.6) del modelo LIPM. Esta ecuación es el resultado de la suma de dos miembros (L1 y L2):

$$L1 = x_{CoM} \tag{3.17}$$

$$L2 = -\frac{z_d}{g}\frac{d^2 x_{CoM}}{dt^2} \tag{3.18}$$

L1 se conoce por el modelo en simulación para todos los ángulos que resuelvan la cinemática directa del bípedo. Por otro lado, se utiliza un sensor para estimar la aceleración del CoM y resolver L2.

El sensor utilizado (MPU-6050) permite conocer la inclinación y la velocidad angular en x, y, z. Esto es favorable porque se puede utilizar la velocidad angular para conocer la aceleración del CoM.

Para encontrar la aceleración, se obtiene ωs_i, que es la i-ésima muestra del giroscopio. Si el tiempo de muestreo es T_s, entonces la velocidad angular es:

$$\omega_i = \omega s_i * T_s \tag{3.19}$$

Después, se encuentra la aceleración angular dada por:

$$\alpha_i = (\omega_i - \omega_{i-1})/T_s \tag{3.20}$$

Ahora, se encuentra la aceleración del CoM:

$$\frac{d^2 x_{CoM}}{dt^2} = \alpha_i * R \tag{3.21}$$

En (3.21), R es la longitud del péndulo.

Para encontrar ωs_i, el PIC16F886 se comunica con el MPU-6050 utilizando el protocolo I2C a una velocidad de 400 Kbit/s. El registro 43 en el sensor es el que devuelve los 8 bits superiores del giroscopio en el eje x. En el microcontrolador se normalizan los valores a un rango que va desde 63 para un ángulo de -90^o hasta 191 para un ángulo de 90^o. A este valor llamaremos wm_i. Después, el Raspberry Pi III se comunica con el microcontrolador para recibir el dato, y en Python calcula ωs_i siguiendo la ecuación (3.22).

$$\omega s_i = \frac{pi}{2} * \frac{wm_i - 127}{90} \tag{3.22}$$

El circuito construido se muestra en la Figura 3.8.

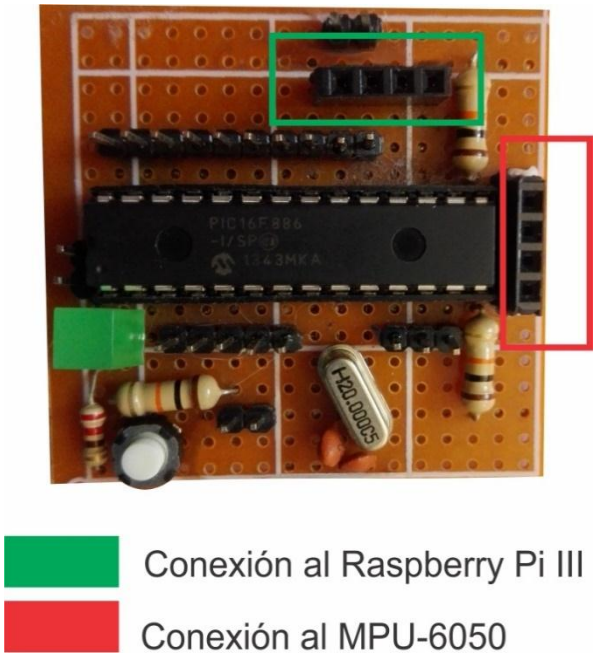

Figura 3.8. Circuito de comunicación, PIC16F886

El microcontrolador tiene un cristal de 20 MHz. Para revisar la programación, ver el Anexo B.

3.2.3 Python y el Raspberry PI III

La integración del algoritmo de control con el driver de servos y el microcontrolador que procesa las señales del giroscopio se realizó en Python 3.6. El programa se ejecuta desde un Raspberry Pi III con el sistema operativo basado en Debian propio para el Raspberry Pi, llamado Raspbian (versión Jessie).

La única librería añadida a Python es la librería pyserial. Esta librería permite comunicar el Raspberry con los demás dispositivos utilizando interfaces seriales con módulos USB a Serial TTL. Las velocidades de comunicación son 115200 Bd y 38400 Bd para el controlador de servos y el microcontrolador respectivamente.

El Raspberry Pi III se conecta a una red WiFi y es accedido desde una computadora a través de una conexión SSH. Desde la computadora se inicializa el programa principal del Raspberry Pi III.

Para instalar la librería pyserial después de la instalación de Raspbian Jessie, solo se ejecutan los siguientes comandos desde el terminal (asumiendo que hay una conexión a internet):

sudo apt-get install python3-dev python3-pip

sudo python3 -m pip install pyserial

sudo pip3 install --upgrade distribute

sudo pip3 install ipython

Con la librería instalada, se procede a escribir el algoritmo de control. Para más detalle del programa, revisar el Anexo C.

Dos baterías con 3.6 V cada una se conectan en serie para alimentar el controlador de servos (Figura 3.9, derecha), mientras que una sola independiente alimenta el Raspberry Pi III (Figura 3.9, izquierda). Estas baterías fueron extraídas de una Laptop. Las celdas permiten al robot tener una autonomía de hasta 22 minutos aproximadamente.

Figura 3.9. Baterías para alimentar el bípedo

La estructura final con el sensor, circuito de comunicación, controlador de servos y Raspberry Pi III se muestran en la Figura 3.10.

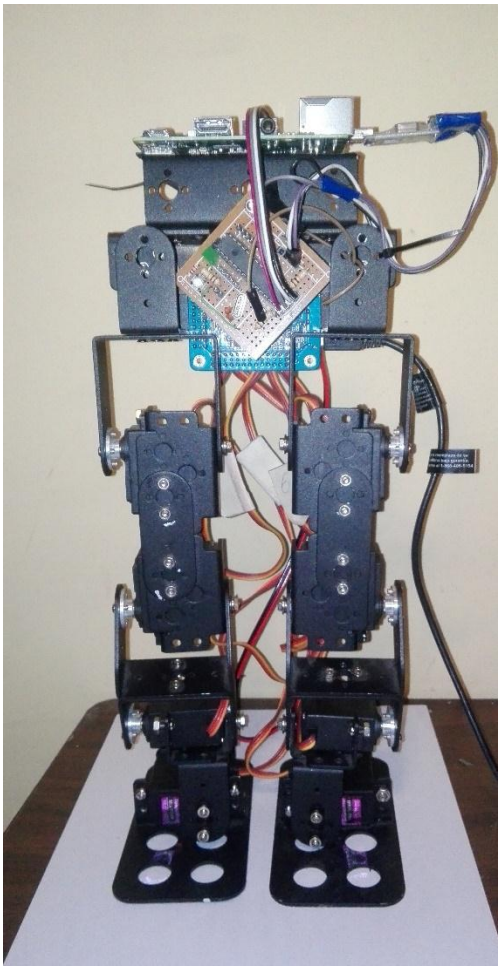

Figura 3.10. Estructura completa

Capítulo IV: RESULTADOS

En esta sección se muestran los resultados de simulación. Primero, se presenta el seguimiento de la trayectoria sin ninguna perturbación (Figura 4.1). Después, se añaden perturbaciones en diferentes momentos del ciclo de caminata y se evidencia la respuesta correctiva del controlador junto con la eliminación del error.

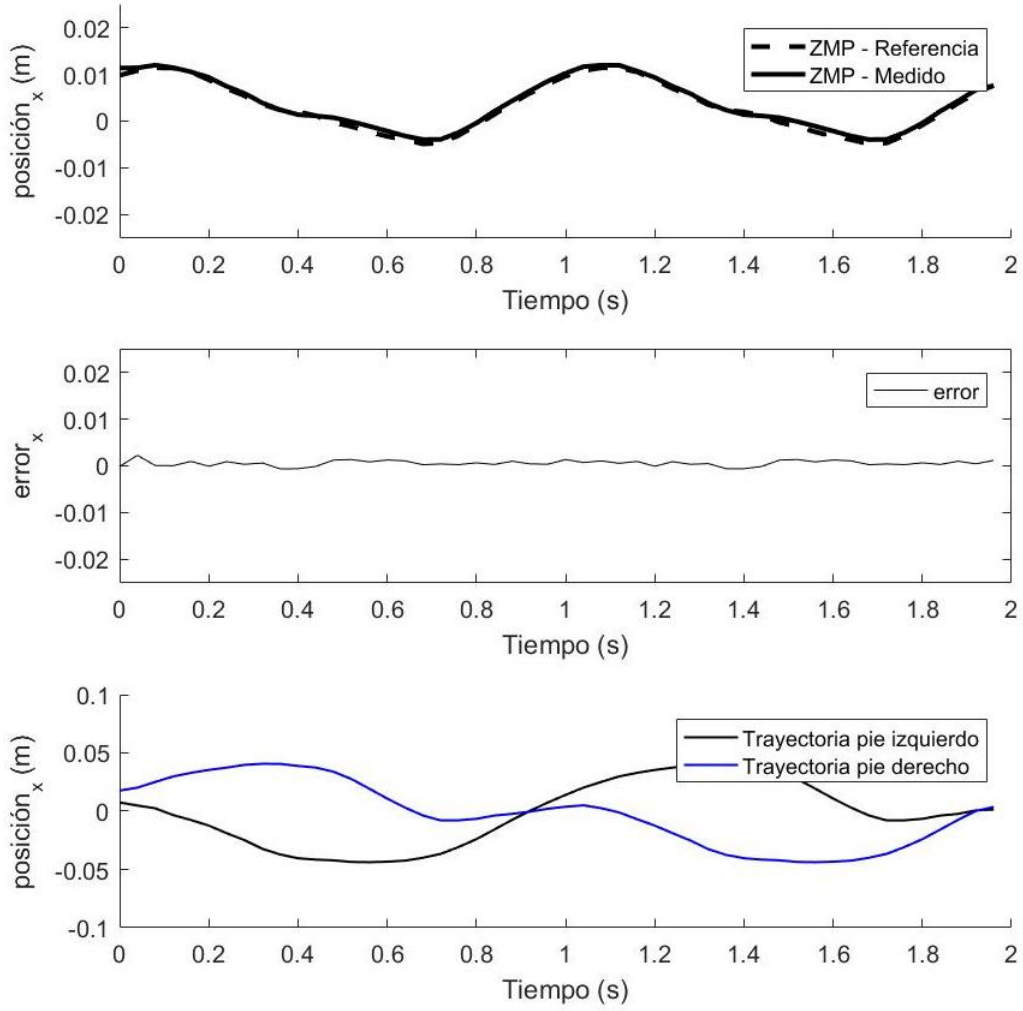

Figura 2.1 Seguimiento de trayectoria sin perturbación

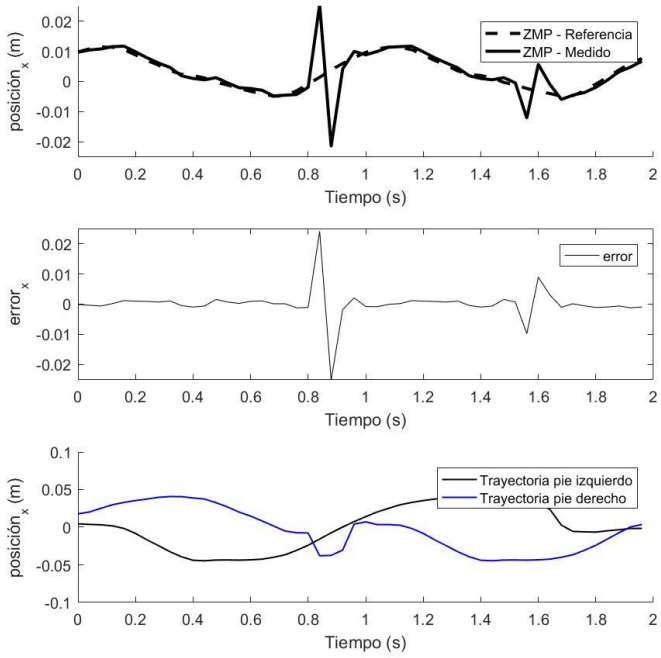

Figura 2.2. Respuesta ante perturbación en t=0.84 s y t =1.56 s

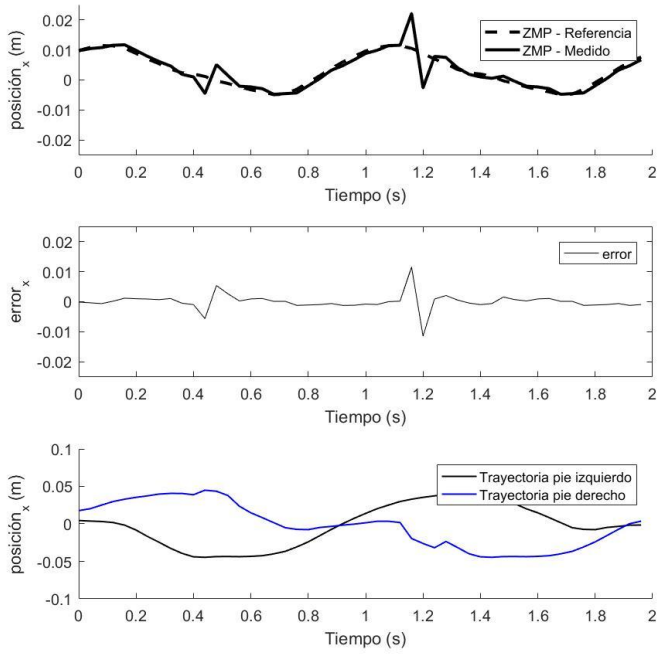

Figura 2.3. Respuesta ante perturbación en t=0.44 s y t =1.16 s

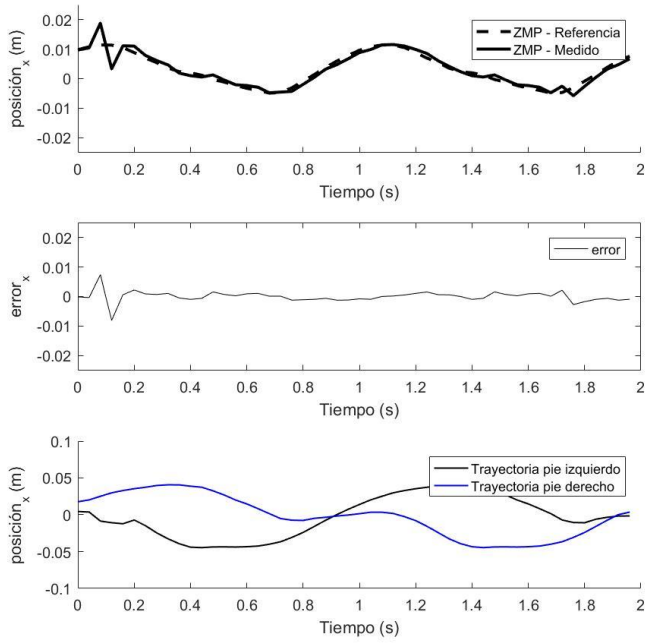

Figura 2.4. Respuesta ante perturbación en t=0.08 s y t =1.72 s

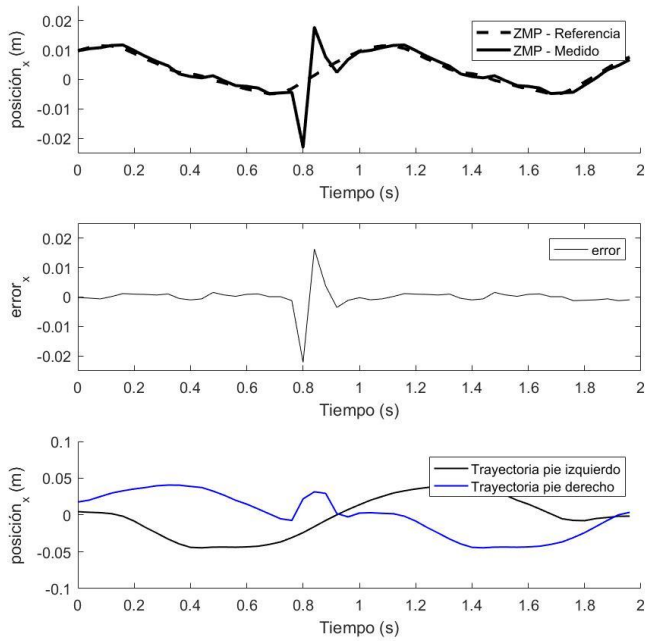

Figura 2.5. Respuesta ante perturbación en t=0.80 s

A continuación, se muestra la trayectoria del bípedo real, calculando el ZMP con los valores del sensor MPU-6050. Primero, sin perturbación y después con perturbaciones constantes.

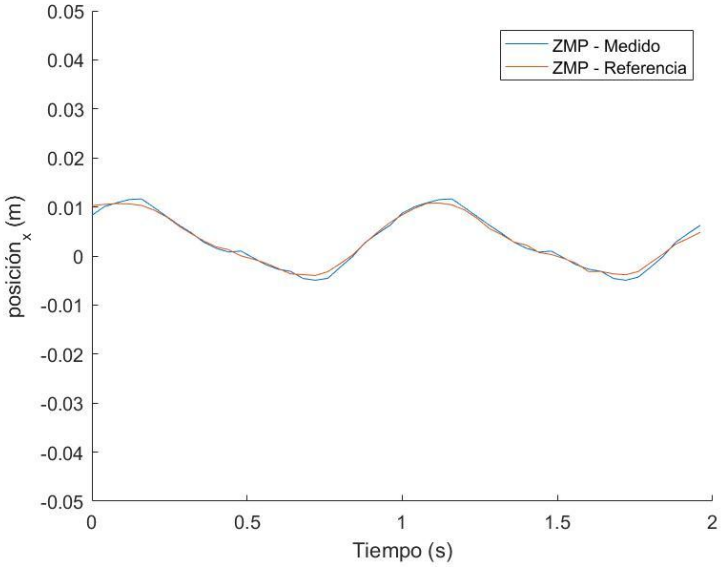

Figura 2.6. Trayectoria sin perturbación.

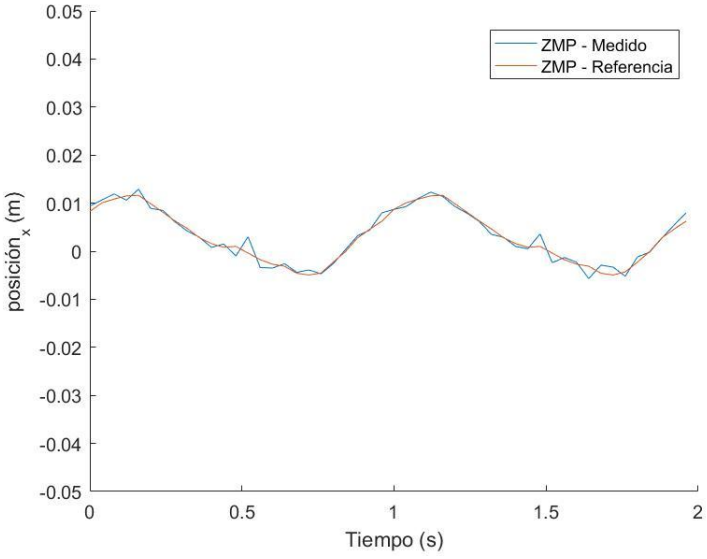

Figura 2.7. Trayectoria con perturbaciones

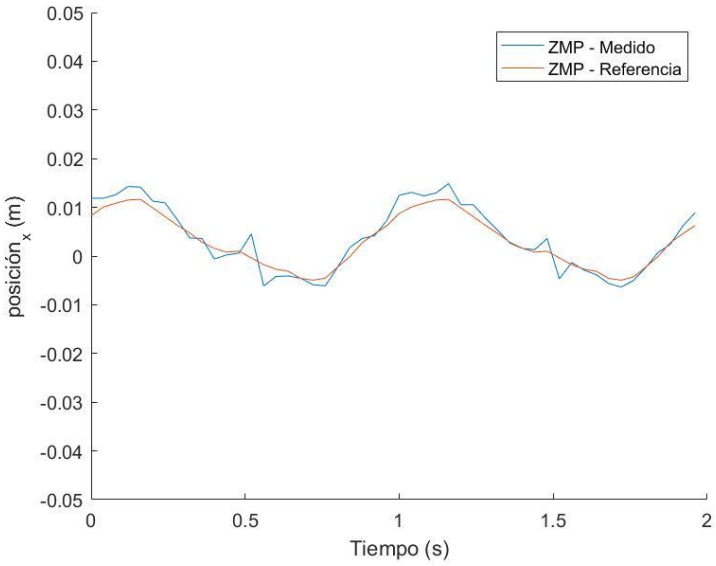

Figura 2.8. Trayectoria con perturbaciones

CONCLUSIÓN Y DISCUSIÓN

En esta tesis se presentó un procedimiento para generar y controlar la trayectoria de caminata de un bípedo con 10DOF. Primeramente, se desarrolló la implementación por simulación y posteriormente se corroboraron los resultados en el prototipo real.

Para crear una trayectoria natural en el plano sagital, se partió de los ángulos de cadera, rodilla y tobillo capturados por sensores en un ser humano en marcha bípeda en el DEET de la Universidad de Cuenca y se utilizó un bípedo con 10 DOF como prototipo para la implementación. El robot fue parametrizado con D-H y se determinó la masa y la posición del CoM para cada vínculo. Posteriormente, se escogió el modelo LIPM, cuya simplicidad redujo significativamente el modelo dinámico del sistema para avanzar con el ajuste de trayectoria y con la estrategia de control.

Con la parametrización del robot, se elaboró una simulación que permita predecir la posición del CoM y aplicar el modelo LIPM para calcular el ZMP al controlar la trayectoria. El ajuste de trayectoria se realizó dividiendo el ciclo en etapas, teniendo al menos un pie de soporte sobre el cual se pueda realizar la corrección.

Para la corrección del ZMP, se utilizó en un observador con un controlador óptimo basado en un servomecanismo. Este controlador utilizó ventanas de 18 predicciones futuras, así como un integrador y retroalimentación proporcional de estados. Un número mayor de predicciones no generó mejora alguna en la acción del controlador.

La respuesta del bípedo ante perturbaciones y la acción correctiva del controlador demostraron la validez de los métodos aplicados, siendo estos sencillos con respuesta robusta. El mapeo de soluciones a la cinemática inversa por LM redujo la carga computacional en el bípedo. La disminución

del número de sensores necesarios es un factor importante en el momento del diseño, ya que la retroalimentación solo se dio por el giroscopio en el prototipo real, esto también redujo el costo de implementación. Durante las pruebas en el prototipo, se pudo observar una región inestable en las transiciones de pierna de soporte, sin embargo, el controlador satisfactoriamente corrige la irregularidad.

Finalmente, se recomienda el método planteado para su utilización futura en bípedos y exoesqueletos.

TRABAJO FUTURO

El trabajo futuro que se espera realizar está en generar trayectorias no solo en el plano sagital, sino también en el frontal y aplicar las ecuaciones de LIPM para el control frontal. Además, se buscará tener movilidad con libertad en el plano XY. También se considera utilizar motores servo más robustos, ya que los utilizados se deterioran pronto y un sobrepeso en pruebas quema el motor.

En una futura implementación, la construcción del hardware se podría reducir casi completamente si se utilizar un robot bípedo de fábrica, como el caso de DARwIn-OP, un robot humanoide con sofisticados sensores, alta capacidad de carga y movilidad.

BIBLIOGRAFÍA

Ao, S.-I., Amouzegar, M., & Rieger, B. B. (Eds.). (2011). *Intelligent Automation and Systems Engineering* (Vol. 103, pp. 15-16). New York, NY: Springer New York.

Craig, J. J. (2006). *Robótica*. México: Pearson Prentice Hall.

Crisóstomo, M., Coimbra, A. P., & Ferreira, J. (2009). ZMP trajectory reference for the sagittal plane control of a biped robot based on a human CoP and gait. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on* (pp. 1588–1593).

Czarnetzki, S., Kerner, S., & Urbann, O. (2010). Applying Dynamic Walking Control for Biped Robots. In *RoboCup* (pp. 69–80). Springer.

Czarnetzki, S., Kerner, S., & Urbann, O. (2009). Observer-based dynamic walking control for biped robots. *Robotics and Autonomous Systems*, 57(8), 839–845.

Dang, D. (2012). *Humanoid manipulation and locomotion with real-time footstep optimization* (PhD Thesis).

Elechouse, “32-Channel Servo Controller Manual”. Hoja de especificaciones técnicas. URL: https://www.robotshop.com/letsmakerobots/files/32_Servo_Controller_Manual.pdf .Consulta: 17-diciembre-2017

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Yokoi, K., & Hirukawa, H. (2002). A realtime pattern generator for biped walking. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on* (Vol. 1, pp. 31–37).

Katayama, T., Ohki, T., Inoue, T., & Kato, T. (1985). Design of an optimal controller for a discrete-time system subject to previewable demand. *International Journal of Control*, 41(3), 677–699.

Lourakis, M. I. (2005). A brief description of the Levenberg-Marquardt algorithm implemented by levmar. *Foundation of Research and Technology*, *4*(1), 1–6.

Merchant, H. K., & Ahire, D. D. (2017). Industrial Automation using IoT with Raspberry Pi. *Quark*, *1000*, 400MH.

Microcontrollers, U. S. B. (2007). 28/40/44-Pin, High-Performance, Enhanced Flash, USB Microcontrollers with nanoWatt Technology.

Mummolo, C., Mangialardi, L., & Kim, J. H. (2013). Quantifying dynamic characteristics of human walking for comprehensive gait cycle. *Journal of Biomechanical Engineering*, *135*(9), 091006.

Muscolo, G. G., Recchiuto, C. T., Laschi, C., Dario, P., Hashimoto, K., & Takanishi, A. (2011). A method for the calculation of the effective Center of Mass of humanoid robots. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on* (pp. 371–376).

Secretaría Nacional de Planificación y Desarrollo. (2014). Agenda Nacional para la Igualdad en Discapacidades 2013-2017. URL: http://www.planificacion.gob.ec/wp-content/uploads/downloads/2014/09/Agenda-Nacional-para-Discapacidades.pdf .(pp. 77). Descargado: 14-diciembre-2017.

Sugihara, T. (2009). Solvability-unconcerned inverse kinematics based on Levenberg-Marquardt method with robust damping. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on* (pp. 555–560). IEEE.

TowerPro, “MG995 High Speed Metal Gear Dual Bearing Servo”. Hoja de especificaciones técnicas. URL: https://www.mpja.com/download/31150mp.pdf . Consulta: 17-diciembre-2017

Vicepresidencia de la República. (2012). Ecuador producirá alrededor de 300 prótesis mensuales y cubrirá la demanda interna hasta diciembre de 2013. URL: http://www.vicepresidencia.gob.ec/ecuador-producira-alrededor-de-300-protesis-mensuales-y-cubrira-la-demanda-interna-hasta-diciembre-de-2013-2/ . Consulta: 14-diciembre-2017.

Vukobratović, M., & Borovac, B. (2004). Zero-moment point—thirty five years of its life. *International Journal of Humanoid Robotics*, *1*(01), 157–173.

Weigang, G. (2009). Design and Control of a 10 DOF Biped Robot. *Lakehead University, Faculty of Engineering*, Thunder Bay, Ontario. (pp. 22)

ANEXOS

Anexo A – Código de simulación en MATLAB

```
function []=Biped_Walk()
clear all
close all
clc
%*******************************************************%
%   Universidad de Cuenca                               %
%   Facultad de Ingeniería                              %
%   Escuela de Electrónica y Telecomunicaciones         %
%*******************************************************%
%   Autor : Daniel Astudillo                            %
%   Fecha : 03-14-2018                                  %
%*******************************************************%


% Variables para LM
global PxR PzR PxL PzL;
% Valores iniciales para LM
x0 = [-30,-86,-25];


% Ecuaciones Cinemática Pierta Derecha
funR = @RightLegEquations;
% Ecuaciones Cinemática Pierta Izquierda
funL = @LeftLegEquations;


% cargar ángulos plano sagital
load('angles_sagital.mat')
% cargar ángulos plano frontal
load('angles_frontal.mat')
% cargar referencia
load('x_reference.mat')

% Tamaño de la muestra
Nm=50;
% # de repeticiones del ciclo
rep=1;
max_rep=1;

% ************************************************************

% Genera matriz de Transformación para cada vinculo
cf=@getPosition;



mass_joints=[100.5 99.5 91.5 90.5 114.5 (184.0+42.0+40.0+110.0)/2.0];%184
mass_total=sum(mass_joints);

center_circuitR=[1 0 0 0;0 1 0 -0.01;0 0 1 0.03;0 0 0 1];
center_circuitL=[1 0 0 0;0 1 0 0.01;0 0 1 0.03;0 0 0 1];

x_trayectoryR=[];
x_trayectoryRF=[];
```

```
y_trayectoryR=[];
z_trayectoryR=[];

x_trayectoryL=[];
x_trayectoryLF=[];
y_trayectoryL=[];
z_trayectoryL=[];


x_com=[];
y_com=[];
z_com=[];

errors=zeros(1,(max_rep)*Nm);
x_com=zeros(1,Nm);
y_com=zeros(1,Nm);
z_com=zeros(1,Nm);



while (rep<=max_rep)
    i=1;
    rlegA=[];
    rlegB=[];
    rlegC=[];
    rlegD=[];
    rlegE=[];
    rlegF=[];
    rP=[];

    llegA=[];
    llegB=[];
    llegC=[];
    llegD=[];
    llegE=[];
    llegF=[];
    lP=[];



    while(i<=Nm)


[RF,P_rA,P_rB,P_rC,P_rD,P_rE,P_rF]=cf(angles_frontal(i),angles_sagital(1,i),angles_sa
gital(2,i),angles_sagital(3,i),-angles_frontal(i),1); %Right Leg

[LF,P_lA,P_lB,P_lC,P_lD,P_lE,P_lF]=cf(angles_frontal(i),angles_sagital(4,i),angles_sa
gital(5,i),angles_sagital(6,i),-angles_frontal(i),-1); %Left Leg


        % ************************************************************
        % Calcular CoM
        % ************************************************************
        rlegA_CM=RF*[1 0 0 0;0 1 0 -0.0105;0 0 1 0;0 0 0 1];
        rlegB_CM=P_rA*[1 0 0 0;0 1 0 0.;0 0 1 0;0 0 0 1];
        rlegC_CM=P_rB*[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
        rlegD_CM=P_rC*[1 0 0 0;0 1 0 0.0105;0 0 1 0;0 0 0 1];
        rlegE_CM=P_rD*[1 0 0 0;0 1 0 0.0105;0 0 1 0;0 0 0 1];

        llegA_CM=LF*[1 0 0 0;0 1 0 0.0105;0 0 1 0;0 0 0 1];
        llegB_CM=P_lA*[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
        llegC_CM=P_lB*[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
        llegD_CM=P_lC*[1 0 0 0;0 1 0 0.0105;0 0 1 0;0 0 0 1];
        llegE_CM=P_lD*[1 0 0 0;0 1 0 -0.0105;0 0 1 0;0 0 0 1];


RL_CM=[rlegA_CM(1:3,4)';rlegB_CM(1:3,4)';rlegC_CM(1:3,4)';rlegD_CM(1:3,4)';rlegE_CM(1
:3,4)';center_circuitR(1:3,4)'];

LL_CM=[llegA_CM(1:3,4)';llegB_CM(1:3,4)';llegC_CM(1:3,4)';llegD_CM(1:3,4)';llegE_CM(1
:3,4)';center_circuitL(1:3,4)'];
        joints=1;
        X_center_R=0;
        Y_center_R=0;
```

```
        Z_center_R=0;
        X_center_L=0;
        Y_center_L=0;
        Z_center_L=0;
        while joints<=6
            X_center_R=X_center_R+RL_CM(joints,1)*mass_joints(joints);
            Y_center_R=Y_center_R+RL_CM(joints,2)*mass_joints(joints);
            Z_center_R=Z_center_R+RL_CM(joints,3)*mass_joints(joints);
            X_center_L=X_center_L+LL_CM(joints,1)*mass_joints(joints);
            Y_center_L=Y_center_L+LL_CM(joints,2)*mass_joints(joints);
            Z_center_L=Z_center_L+LL_CM(joints,3)*mass_joints(joints);
            joints=joints+1;
        end
        X_center_R=X_center_R/mass_total;
        Y_center_R=Y_center_R/mass_total;
        Z_center_R=Z_center_R/mass_total;
        X_center_L=X_center_L/mass_total;
        Y_center_L=Y_center_L/mass_total;
        Z_center_L=Z_center_L/mass_total;
        X_center=(X_center_R+X_center_L)/2;
        Y_center=(Y_center_R+Y_center_L)/2;
        Z_center=(Z_center_R+Z_center_L)/2;

        % ************************************************************
        x_com(i)=X_center;
        y_com(i)=Y_center;
        z_com(i)=Z_center;



        rP=[rP;RF(1:3,4)'];
        rlegA=[rlegA;P_rA(1:3,4)'];
        rlegB=[rlegB;P_rB(1:3,4)'];
        rlegC=[rlegC;P_rC(1:3,4)'];
        rlegD=[rlegD;P_rD(1:3,4)'];
        rlegE=[rlegE;P_rE(1:3,4)'];
        rlegF=[rlegF;P_rF(1:3,4)'];

        lP=[lP;LF(1:3,4)'];
        llegA=[llegA;P_lA(1:3,4)'];
        llegB=[llegB;P_lB(1:3,4)'];
        llegC=[llegC;P_lC(1:3,4)'];
        llegD=[llegD;P_lD(1:3,4)'];
        llegE=[llegE;P_lE(1:3,4)'];
        llegF=[llegF;P_lF(1:3,4)'];



        npr_x=[0 rP(i,1) rlegA(i,1) rlegB(i,1) rlegC(i,1) rlegD(i,1) rlegE(i,1)
rlegF(i,1)];
        npr_y=[0 rP(i,2) rlegA(i,2) rlegB(i,2) rlegC(i,2) rlegD(i,2) rlegE(i,2)
rlegF(i,2)];
        npr_z=[0 rP(i,3) rlegA(i,3) rlegB(i,3) rlegC(i,3) rlegD(i,3) rlegE(i,3)
rlegF(i,3)];

        npl_x=[0 lP(i,1) llegA(i,1) llegB(i,1) llegC(i,1) llegD(i,1) llegE(i,1)
llegF(i,1)];
        npl_y=[0 lP(i,2) llegA(i,2) llegB(i,2) llegC(i,2) llegD(i,2) llegE(i,2)
llegF(i,2)];
        npl_z=[0 lP(i,3) llegA(i,3) llegB(i,3) llegC(i,3) llegD(i,3) llegE(i,3)
llegF(i,3)];

        x_trayectoryR=[x_trayectoryR rlegE(i,1)];
        x_trayectoryRF=[x_trayectoryRF rlegF(i,1)];
        y_trayectoryR=[y_trayectoryR rlegE(i,2)];
        z_trayectoryR=[z_trayectoryR rlegE(i,3)];

        x_trayectoryL=[x_trayectoryL llegE(i,1)];
        x_trayectoryLF=[x_trayectoryLF llegF(i,1)];
        y_trayectoryL=[y_trayectoryL llegE(i,2)];
        z_trayectoryL=[z_trayectoryL llegE(i,3)];
```

```
        pause(0.0001);
        subplot(311)
        title('plano sagital')
        plot(npl_x,npl_z, '-o','Linewidth',1,'color','black')
        axis([-0.4 0.4 -0.32 0.01])
        hold on
        plot(npr_x,npr_z, '-o','Linewidth',2,'color','black')
        plot(X_center,Z_center,'-o','Linewidth',3,'color','red')
        %legend('Pierna izquierda','Pierna derecha','CoMxz')
        %xlabel('posición_x')
        %ylabel('posición_z')
        hold off

        subplot(312)
        title('plano frontal')
        plot(npl_y,npl_z, '-o','Linewidth',1,'color','black')
        axis([-0.4 0.4 -0.32 0.01])
        hold on
        plot(npr_y,npr_z, '-o','Linewidth',2,'color','black')
        plot(Y_center,Z_center,'-o','Linewidth',3,'color','red')
        %legend('Pierna izquierda','Pierna derecha','CoMyz')
        %xlabel('posición_y')
        %ylabel('posición_z')
        hold off

        subplot(313)
        title('plano transversal')
        plot(npl_x,npl_y, '-o','Linewidth',1,'color','black')
        axis([-0.4 0.4 -0.11 0.11])
        hold on
        plot(npr_x,npr_y ,'-o','Linewidth',2,'color','black')
        plot(X_center,Y_center,'-o','Linewidth',3,'color','red')
        %legend('Pierna izquierda','Pierna derecha','CoMxy')
        %xlabel('posición_x')
        %ylabel('posición_y')
        hold off

        i=i+1;


    end
    rep=rep+1;

end


vector_plot=[0:1:(max_rep)*Nm-1]*0.04;

figure
subplot(211)
hold on
plot([vector_plot],[x_reference(1:(max_rep)*Nm)],'--','color','black','Linewidth',2)
plot([vector_plot],[x_com],'color','black','Linewidth',2)
legend('ZMP - Referencia','ZMP - Medido')
xlabel('Tiempo (s)')
ylabel('posición_x (m)')
axis([0 2.0*(max_rep) -0.05 0.05])
hold off

subplot(212)
hold on
plot([vector_plot],[x_trayectoryL],'Linewidth',1,'color','black')
plot([vector_plot],[x_trayectoryR],'Linewidth',1,'color','blue')
xlabel('Tiempo (s)')
ylabel('posición_x (m)')
legend('Trayectoria pie izquierdo','Trayectoria pie derecho')
axis([0 2*(max_rep) -0.1 0.1])
hold off
end
```

```
function FR = RightLegEquations(x)
global PxR PzR;

FR(1) = PxR-(0.1*sin(0.017*x(1)) +
0.059*cos(0.017*x(3))*(cos(0.017*x(1))*sin(0.017*x(2)) +
cos(0.017*x(2))*sin(0.017*x(1))) +
0.059*sin(0.017*x(3))*(cos(0.017*x(1))*cos(0.017*x(2)) -
1.0*sin(0.017*x(1))*sin(0.017*x(2))) + 0.063*cos(0.017*x(1))*sin(0.017*x(2)) +
0.063*cos(0.017*x(2))*sin(0.017*x(1)));
FR(2) = PzR-(0.059*sin(0.017*x(3))*(cos(0.017*x(1))*sin(0.017*x(2)) +
cos(0.017*x(2))*sin(0.017*x(1))) - 0.1*cos(0.017*x(1)) -
0.059*cos(0.017*x(3))*(cos(0.017*x(1))*cos(0.017*x(2)) -
1.0*sin(0.017*x(1))*sin(0.017*x(2))) - 0.063*cos(0.017*x(1))*cos(0.017*x(2)) +
0.063*sin(0.017*x(1))*sin(0.017*x(2)) - 0.08);
FR(3)=x(2)+x(3)+x(1);
end

function FR = LeftLegEquations(x)
global PxL PzL;

FR(1) =PxL-(0.1*sin(0.017*x(1)) +
0.059*cos(0.017*x(3))*(cos(0.017*x(1))*sin(0.017*x(2)) +
cos(0.017*x(2))*sin(0.017*x(1))) +
0.059*sin(0.017*x(3))*(cos(0.017*x(1))*cos(0.017*x(2)) -
1.0*sin(0.017*x(1))*sin(0.017*x(2))) + 0.063*cos(0.017*x(1))*sin(0.017*x(2)) +
0.063*cos(0.017*x(2))*sin(0.017*x(1)));
FR(2) =PzL-(0.059*sin(0.017*x(3))*(cos(0.017*x(1))*sin(0.017*x(2)) +
cos(0.017*x(2))*sin(0.017*x(1))) - 0.1*cos(0.017*x(1)) -
0.059*cos(0.017*x(3))*(cos(0.017*x(1))*cos(0.017*x(2)) -
1.0*sin(0.017*x(1))*sin(0.017*x(2))) - 0.063*cos(0.017*x(1))*cos(0.017*x(2)) +
0.063*sin(0.017*x(1))*sin(0.017*x(2)) - 0.08);
FR(3)=x(2)+x(3)+x(1);
end
function [matrix]=DHP(th,d,a,al)
matrix=[cos(pi*th/180) -cos(pi*al/180)*sin(pi*th/180) sin(pi*al/180)*sin(pi*th/180)
a*cos(pi*th/180);sin(pi*th/180) cos(pi*al/180)*cos(pi*th/180) -
sin(pi*al/180)*cos(pi*th/180) a*sin(pi*th/180);0 sin(pi*al/180) cos(pi*al/180) d;0 0
0 1];
end
function [Ti,A,B,C,D,E,F]=getPosition(th_1,be_1,ga_1,al_1,ep_1,leg)
Z_f=0;
X_f=-0.014;
Z_p=0.06;
Ti=DHP(0,0,0,90)*DHP(90,0.045*(leg),0,90);
T1=DHP(th_1,0,-0.08,-90);
T2=DHP(be_1,0,-0.10,0);
T3=DHP(ga_1,0,-0.063,0);
T4=DHP(al_1,-0.0105*(leg),-0.045,90);
T5=DHP(ep_1,Z_f,X_f,0);
Tp=DHP(0,Z_p,0,0);

A=Ti*T1;
B=A*T2;
C=B*T3;
D=C*T4;
E=D*T5;
F=E*Tp;

end
```

Anexo B – Código en MicroC PRO para PIC

```
unsigned short int MPU6050_R;
unsigned short int MPU6050_W;
unsigned short int acex;
unsigned short int acey;
unsigned short int acez;
unsigned short int girox;
unsigned short int giroy;
unsigned short int giroz;
unsigned short int auxRCREG;

void configUSART()   // Configurar USART
{
        TRISC7_bit=1;
        TRISC6_bit=1;
        SPBRG=32; //38400 baud @20MHz
        TXSTA.SYNC=0;
        TXSTA.BRGH=1;
        TXSTA.TXEN=1;
        RCSTA.SPEN=1;
        RCSTA.CREN=1;

        PIR1.RCIF=0;
        PIR1.TXIF=0;
        }

void configINT() // Habilitar Interrupciones
{
        PIE1.RCIE=1; // Habilitar Interrupción nueva recepción USART

        INTCON.PEIE=1; // Habilitar Interrupciones periféricas
        INTCON.GIE=1; // Habilitar Interrupciones globales

}


getValGiro(unsigned short int fr)  // Recibir valor del Giroscopio
{

I2C1_Start();
I2C1_Wr(MPU6050_W);
I2C1_Wr(fr);
I2C1_Repeated_Start();
I2C1_Wr(MPU6050_R);
fr=I2C1_Rd(0u);
I2C1_Stop();

return fr;
}
void configGiro() // Configurar Giroscopio
{
  MPU6050_R=209; // direccion para lectura
  MPU6050_W=208;  // direccion para escritura
  I2C1_Init(400000); //  Inicializar I2C

  //  Bytes de configuración e inicialización del MPU-6050
  I2C1_Start();
  I2C1_Wr(MPU6050_W);
  I2C1_Wr(0x68);
  I2C1_Wr(0x07);
  I2C1_Stop();

  I2C1_Start();
```

```
  I2C1_Wr(MPU6050_W);
  I2C1_Wr(0x6B);
  I2C1_Wr(0x00);
  I2C1_Stop();


  I2C1_Start();
  I2C1_Wr(MPU6050_W);
  I2C1_Wr(0x1A);
  I2C1_Wr(0x00);
  I2C1_Stop();

  I2C1_Start();
  I2C1_Wr(MPU6050_W);
  I2C1_Wr(0x6C);
  I2C1_Wr(0x00);
  I2C1_Stop();

  I2C1_Start();
  I2C1_Wr(MPU6050_W);
  I2C1_Wr(0x1B);
  I2C1_Wr(0x00);
  I2C1_Stop();

}
void regulateValues()
{


if  (acex<127)
{acex=acex+127;}
else
{acex=acex-127;}

if  (acey<127)
{acey=acey+127;}
else
{acey=acey-127;}

if  (acez<127)
{acez=acez+127;}
else
{acez=acez-127;}

if  (girox<127)
{girox=girox+127;}
else
{girox=girox-127;}

if  (giroy<127)
{giroy=giroy+127;}
else
{giroy=giroy-127;}

if  (giroz<127)
{giroz=giroz+127;}
else
{giroz=giroz-127;}

}
void interrupt() iv 0x04 ics ICS_OFF{ // Vector de Interrupciones

if (PIR1.RCIF) // Se recibe un byte
{PIR1.RCIF=0;
auxRCREG=RCREG;

// Se deshabilita interrupciones mientras se envian los valores
INTCON.PEIE=0;
INTCON.GIE=0;

 while (~PIR1.TXIF);
 //Transmitir valor pedido al puerto Serial
 if (auxRCREG=='A'){TXREG=acex;}
 if (auxRCREG=='B'){TXREG=acey;}
```

```
 if (auxRCREG=='C'){TXREG=acez;}
 if (auxRCREG=='D'){TXREG=girox;}
 if (auxRCREG=='E'){TXREG=giroy;}
 if (auxRCREG=='F'){TXREG=giroz;}
 INTCON.PEIE=1;
 INTCON.GIE=1;
 // Se vuelven a habilitar las interrupciones
}
}

void main() {

TRISA0_bit=0;
PORTA.F0=1;


configGiro();
configUSART();
configINT();



while  (1)
{
// Se deshabilita interrupciones mientras se procesan los valores
INTCON.PEIE=0;
INTCON.GIE=0;

acex=getValGiro(0x3B);
acey=getValGiro(0x3D);
acez=getValGiro(0x3F);
girox=getValGiro(0x43);
giroy=getValGiro(0x45);
giroz=getValGiro(0x47);
regulateValues();
// Se vuelven a habilitar las interrupciones
INTCON.PEIE=1;
INTCON.GIE=1;
PORTA.F0=~PORTA.F0;
delay_ms(1);

}

}
```

Anexo C – Código en Python 3.6 para Raspberry Pi III

```
import time
import serial
from math import sin as sin
from math import cos as cos
from math import tan as tan
from math import atan as atan
from math import pi as pi

center_servos=[1500,1500,1500,1500,1500,1500,1500,1500]
n_cycles=3

angles=[[31.5,32.5,33,33,32.5,30.5,28.5,26.5,24.5,23,23.5,25,26,27,27,25.5,23.5,21.5,
20,20.5,21.5,22,23,24,24.5,25,25,24,22.5,19.5,15.5,10.5,6,1,-1.5,-2,-2,-2,-
0.5,0.5,2.5,4.5,5.5,8.5,12,16,20,24,27,29.5],[-66.5,-63,-59,-54.5,-49.5,-43.5,-36.5,-
29.5,-23.5,-20.5,-22,-29,-38,-46,-53,-56.5,-57.5,-59,-60,-60,-61,-61,-61,-61,-60.5,-
60,-59,-58,-57.5,-56.5,-52.5,-46.5,-39.5,-33.5,-29.5,-29.5,-31.5,-35,-39.5,-45,-50,-
54,-57.5,-61,-65,-68.5,-70.5,-71,-70.5,-
69.5],[35.25,31.1875,26.8125,22.375,17.875,13.375,9.25,5.8125,3.625,2.875,3.75,6,9.37
5,13.75,18.75,23.875,28.3125,31.8125,34.25,35.75,36.5625,36.75,36.8571,36.6667,36.4,3
5,34,33.6667,33.7143,33.5625,33.25,32.9375,32.625,32.375,32.3125,32.8125,33.9375,35.5
625,37.8125,40.375,43,45.4375,47.375,48.5625,48.9375,48.5,47.1875,45.1875,42.4375,39.
0625],[25,25,24,22.5,19.5,15.5,10.5,6,1,-1.5,-2,-2,-2,-
0.5,0.5,2.5,4.5,5.5,8.5,12,16,20,24,27,29.5,31.5,32.5,33,33,32.5,30.5,28.5,26.5,24.5,
23,23.5,25,26,27,27,25.5,23.5,21.5,20,20.5,21.5,22,23,24,24.5],[-60,-59,-58,-57.5,-
56.5,-52.5,-46.5,-39.5,-33.5,-29.5,-29.5,-31.5,-35,-39.5,-45,-50,-54,-57.5,-61,-65,-
68.5,-70.5,-71,-70.5,-69.5,-66.5,-63,-59,-54.5,-49.5,-43.5,-36.5,-29.5,-24,-20.5,-
22,-29,-38,-46,-53,-56.5,-57.5,-59,-60,-60,-61,-61,-61,-61,-
60.5],[35,34,33.6667,33.7143,33.5625,33.25,32.9375,32.625,32.375,32.3125,32.8125,33.9
375,35.5625,37.8125,40.375,43,45.4375,47.375,48.5625,48.9375,48.5,47.1875,45.1875,42.
4375,39.0625,35.25,31.1875,26.8125,22.375,17.875,13.375,9.25,5.8125,3.625,2.875,3.75,
6,9.375,13.75,18.75,23.875,28.3125,31.8125,34.25,35.75,36.5625,36.75,36.8571,36.6667,
36.4],[-14,-14,-14,-14,-14,-12.2778,-9.3889,-6.5,-3.6111,-
0.72222,2.1667,5.0556,7.9444,10.8333,13.5,14,14,14,14,14,14,14,14,14,14,14,14,14,14,1
4,12.2778,9.3889,6.5,3.6111,0.72222,-2.1667,-5.0556,-7.9444,-10.8333,-13.5,-14,-14,-
14,-14,-14,-14,-14,-14,-14,-14],[-14,-14,-14,-14,-14,-12.2778,-9.3889,-6.5,-3.6111,-
0.72222,2.1667,5.0556,7.9444,10.8333,13.5,14,14,14,14,14,14,14,14,14,14,14,14,14,14,1
4,12.2778,9.3889,6.5,3.6111,0.72222,-2.1667,-5.0556,-7.9444,-10.8333,-13.5,-14,-14,-
14,-14,-14,-14,-14,-14,-14,-14],]
angles_position=[[-28,1,26,-0.075],[-27,0,26,-0.074],[-28,2,25,-0.073],[-25,-5,29,-
0.072],[-27,0,25,-0.071],[-26,-2,26,-0.07],[-24,-5,28,-0.069],[-23,-7,29,-0.068],[-
22,-10,30,-0.067],[-21,-12,31,-0.066],[-20,-14,32,-0.065],[-18,-16,33,-0.064],[-17,-
18,33,-0.063],[-16,-19,34,-0.062],[-16,-21,35,-0.061],[-15,-22,35,-0.06],[-14,-
23,36,-0.059],[-13,-25,36,-0.058],[-12,-26,37,-0.057],[-11,-27,37,-0.056],[-11,-
28,37,-0.055],[-10,-29,38,-0.054],[-9,-30,38,-0.053],[-8,-31,38,-0.052],[-8,-32,38,-
0.051],[-7,-33,39,-0.05],[-6,-34,39,-0.049],[-6,-35,39,-0.048],[-5,-36,39,-0.047],[-
4,-36,39,-0.046],[-4,-37,39,-0.045],[-3,-38,39,-0.044],[-2,-41,41,-0.043],[-1,-
41,41,-0.042],[0,-42,41,-0.041],[0,-42,41,-0.04],[1,-43,41,-0.039],[1,-43,41,-
0.038],[2,-44,40,-0.037],[2,-44,40,-0.036],[3,-45,40,-0.035],[3,-45,40,-0.034],[4,-
46,40,-0.033],[4,-46,40,-0.032],[5,-46,40,-0.031],[5,-47,40,-0.03],[6,-47,40,-
0.029],[7,-47,39,-0.028],[7,-48,39,-0.027],[8,-48,39,-0.026],[8,-49,39,-0.025],[9,-
49,39,-0.024],[9,-49,39,-0.023],[10,-49,38,-0.022],[10,-50,38,-0.021],[11,-50,38,-
0.02],[11,-50,38,-0.019],[11,-51,38,-0.018],[12,-51,37,-0.017],[12,-51,37,-
0.016],[13,-51,37,-0.015],[13,-51,37,-0.014],[14,-52,36,-0.013],[14,-52,36,-
0.012],[15,-52,36,-0.011],[15,-52,35,-0.01],[16,-52,35,-0.009],[15,-51,34,-
0.008],[16,-51,33,-0.007],[16,-51,33,-0.006],[17,-51,33,-0.005],[17,-51,32,-
0.004],[17,-51,32,-0.003],[18,-51,32,-0.002],[18,-51,31,-0.001],[19,-51,31,0],[19,-
```

```
51,30,0.001],[19,-51,30,0.002],[20,-51,30,0.003],[20,-51,29,0.004],[21,-
51,29,0.005],[21,-51,28,0.006],[21,-51,28,0.007],[22,-51,27,0.008],[22,-
51,27,0.009],[22,-50,27,0.01],[23,-50,26,0.011],[23,-50,26,0.012],[23,-
50,25,0.013],[24,-50,25,0.014],[24,-50,24,0.015],[24,-50,24,0.016],[25,-
49,23,0.017],[25,-49,23,0.018],[25,-49,22,0.019],[26,-49,22,0.02],[26,-
48,21,0.021],[26,-48,20,0.022],[27,-48,20,0.023],[27,-48,19,0.024],[27,-
47,19,0.025],[27,-47,18,0.026],[28,-47,18,0.027],[28,-46,17,0.028],[28,-
46,16,0.029],[28,-46,16,0.03],[29,-45,15,0.031],[29,-45,15,0.032],[29,-
44,14,0.033],[29,-44,13,0.034],[30,-44,13,0.035],[30,-43,12,0.036],[30,-
43,11,0.037],[30,-42,11,0.038],[30,-42,10,0.039],[31,-41,9,0.04],[31,-
41,9,0.041],[31,-40,8,0.042],[31,-40,7,0.043],[31,-39,6,0.044],[31,-39,6,0.045],[31,-
38,5,0.046],[32,-37,4,0.047],[32,-37,3,0.048],[32,-36,3,0.049],[32,-35,2,0.05],[32,-
35,1,0.051],[32,-34,0,0.052],[32,-33,0,0.053],[31,-29,-3,0.054],[32,-32,-
2,0.055],[31,-27,-5,0.056],[32,-30,-4,0.057],[31,-25,-8,0.058],[30,-23,-
9,0.059],[33,-28,-6,0.06],[30,-20,-11,0.061],[30,-19,-12,0.062],[30,-17,-
14,0.063],[33,-24,-10,0.064],[29,-14,-17,0.065],[28,-11,-18,0.066],[28,-9,-
20,0.067],[27,-7,-22,0.068],[26,-4,-24,0.069],[26,-1,-26,0.07],[25,2,-
28,0.071],[23,6,-31,0.072],[25,4,-30,0.073],[29,-6,-24,0.074],[28,-4,-26,0.075]]
position=[-0.075,-0.074,-0.073,-0.072,-0.071,-0.07,-0.069,-0.068,-0.067,-0.066,-
0.065,-0.064,-0.063,-0.062,-0.061,-0.06,-0.059,-0.058,-0.057,-0.056,-0.055,-0.054,-
0.053,-0.052,-0.051,-0.05,-0.049,-0.048,-0.047,-0.046,-0.045,-0.044,-0.043,-0.042,-
0.041,-0.04,-0.039,-0.038,-0.037,-0.036,-0.035,-0.034,-0.033,-0.032,-0.031,-0.03,-
0.029,-0.028,-0.027,-0.026,-0.025,-0.024,-0.023,-0.022,-0.021,-0.02,-0.019,-0.018,-
0.017,-0.016,-0.015,-0.014,-0.013,-0.012,-0.011,-0.01,-0.009,-0.008,-0.007,-0.006,-
0.005,-0.004,-0.003,-0.002,-
0.001,0,0.001,0.002,0.003,0.004,0.005,0.006,0.007,0.008,0.009,0.01,0.011,0.012,0.013,
0.014,0.015,0.016,0.017,0.018,0.019,0.02,0.021,0.022,0.023,0.024,0.025,0.026,0.027,0.
028,0.029,0.03,0.031,0.032,0.033,0.034,0.035,0.036,0.037,0.038,0.039,0.04,0.041,0.042
,0.043,0.044,0.045,0.046,0.047,0.048,0.049,0.05,0.051,0.052,0.053,0.054,0.055,0.056,0
.057,0.058,0.059,0.06,0.061,0.062,0.063,0.064,0.065,0.066,0.067,0.068,0.069,0.07,0.07
1,0.072,0.073,0.074,0.075]
x_reference=[0.0098,0.0108,0.0114,0.0113,0.0105,0.0088,0.0069,0.0054,0.0036,0.0024,0.
002,0.0012,-0.0003,-0.0012,-0.0023,-0.0033,-0.004,-0.0049,-0.0047,-0.0031,-
0.0008,0.0014,0.0038,0.006,0.0079,0.0096,0.0108,0.0114,0.0113,0.0105,0.0088,0.0069,0.
0054,0.0035,0.0023,0.002,0.0012,-0.0003,-0.0012,-0.0023,-0.0033,-0.004,-0.0049,-
0.0047,-0.0031,-
0.0008,0.0014,0.0038,0.006,0.0076,0.0098,0.0108,0.0114,0.0113,0.0105,0.0088,0.0069,0.
0054,0.0036,0.0024,0.002,0.0012,-0.0003,-0.0012,-0.0023,-0.0033,-0.004,-0.0049,-
0.0047,-0.0031,-
0.0008,0.0014,0.0038,0.006,0.0079,0.0096,0.0108,0.0114,0.0113,0.0105,0.0088,0.0069,0.
0054,0.0035,0.0023,0.002,0.0012,-0.0003,-0.0012,-0.0023,-0.0033,-0.004,-0.0049,-
0.0047,-0.0031,-0.0008,0.0014,0.0038,0.006,0.0076]
Gd=[4.7204,12.4673,7.5926,5.0417,3.731,2.9972,2.4918,2.0655,1.674,1.3179,1.0082,0.752
7,0.5517,0.3999,0.2886,0.2088,0.1521,0.1118,0.0829,0.0619,0.0464,0.0349,0.0262,0.0196
,0.0146,0.0109,0.0081,0.006,0.0044,0.0033,0.0024,0.0018,0.0013,0.001,0.0007,0.0006,0.
0004,0.0003,0.0002,0.0002,0.0001,0.0001,0.0001,0.0001,0,0,0,0,0,0]
y_reference=[]

# ************************************************************
# Controll Variables
# ************************************************************
Nl=50
Zh=0.1785
g=9.8
A=[[1.0494,0.0407,-0.0494],[2.4902,1.0494,-2.4902],[0,0,1]]
B=[[-0.0007],[-0.0494],[0.040]]
Gi= -18.2541
Gx=[[-193.5137,-26.1167,50.9588]]
Ts=-0.04
cycle_samples=50 # 50 samples
starting_point=17


lrf=-0.038
anla=13
max_a_lateral=20

cambiofaseA=9
cambiofaseB=36


Lx=-0.007
Ly=-0.00005
```

```
delta_time_transition=40

def zeros(o,p):
    matrix=[]
    for i in range(o):
        matrix.append([])
        for j in range(p):
            matrix[i].append(0)
    return matrix
def MM(X,Y): ## Matrix Multiplication
    f=len(X)
    c=len(Y[0])
    result=zeros(f,c)
    for i in range(f):
       for j in range(c):
           for k in range(len(Y)):
               result[i][j] += X[i][k] * Y[k][j]
    return result
def SM(X,scalar): ## Scalar Multiplication
    f=len(X)
    c=len(X[0])
    result=zeros(f,c)
    for i in range(f):
        for j in range(c):
            result[i][j] = X[i][k] *scalar
    return result
def CL(matrix, i): ## Get Column
    return [[row[i]] for row in matrix]


def busBin(vector, elemento):
    encontrado = False
    inicio = 0
    fin = len(vector)-1
    while inicio<=fin and not encontrado:
        puntomedio = (inicio + fin)//2
        if vector[puntomedio] == elemento:
            encontrado  = True
        else:
            if elemento < vector[puntomedio]:
                fin = puntomedio-1
            else:
                inicio = puntomedio+1
    return puntomedio
def getLateral(pos):
    if pos<=cambiofaseA or pos>cambiofaseB:
        angleLateral=-anla
    else:
        angleLateral=anla
    if pos==cambiofaseA or pos==cambiofaseB:
        angleLateral=0

    return angleLateral
def getLateralreference(pos):
    if pos<=cambiofaseA or pos>cambiofaseB:
        reference=-1*lrf
    else:
        reference=1*lrf
    if pos==cambiofaseA or pos==cambiofaseB:
        reference=0

    return reference
def generateLateralreference():
    for r in range (2):
        for s in range(50):
            y_reference.append(getLateralreference(s))

def getZMP(sensor,previous_velocity):
    avg_data_a=0
    avg_data_g=0
    nt=5
    for sd in range(nt):
        avg_data_a=avg_data_a+int.from_bytes(getDataGiroscope(vector_data[sensor]),
byteorder='big')
```

```
        avg_data_g=avg_data_g+int.from_bytes(getDataGiroscope(vector_data[sensor+3]),
byteorder='big')
    sensor_data[sensor]=avg_data_a/nt
    sensor_data[sensor+3]=avg_data_g/nt


    actual_inclination=sensor_data[sensor]-127
    actual_velocity=sensor_data[sensor+3]-127

    proyection=Zh*(tan((actual_inclination/64)*(pi/2)))

    if proyection>0.10:
        proyection=0.10
    elif proyection<-0.10:
        proyection=-0.10


    aceleration=(((actual_velocity-previous_velocity)*(pi/2)/64)*Zh)
    real_com=proyection
    actual_zmp=real_com-(aceleration*Zh/g)
    return [actual_zmp,actual_velocity,real_com]



def startSerialController():
    serController = serial.Serial(
        port='COM4',
#       port='/dev/ttyS0',
        baudrate=115200,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=2.0)
    return serController
def sendCommand(command):
   serController.write(bytes(command+ '\r\n', 'UTF-8'))
   return

def startSerialGiroscope():
    serGiroscope = serial.Serial(
    # port='/dev/ttyS0',
        port='COM6',
        baudrate=38400,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=2.0)
    return serGiroscope
def getDataGiroscope(val):
    # send the character to the device
    serGiroscope.write(bytes(val,'ASCII'))
    out = ''
    out=serGiroscope.read(1)
    return out



def generateCommands():
    commands=[]
    for i in range(50):
        new_angles=[]
        for j in range(8):
            if j==6 or j==7:
                angles[j][i]=getLateral(i)

            new_angles.append(angles[j][i])
        commands.append(generateAngles(new_angles,i))
    return commands

def walkingCycle():

    # ************************************************************
    # Controll Variables
    # ************************************************************
```

```
rep=0
start_cycle=True
starting_point=17

angle_h=0
angle_k=0
angle_t=0
previous_velocity_x=0
previous_velocity_y=0

integral_x=0
integral_y=0

x=[[0],[0],[0]]
xa=[[0],[0],[0]]

y=[[0],[0],[0]]
ya=[[0],[0],[0]]

nzmp_1_x=0
nzmp_1_y=0
e_x=0
e_y=0


while rep<n_cycles:
    i=0
    while i<cycle_samples:


        initial_time=time.time()
        if rep==0 and start_cycle:
            i=starting_point
            start_cycle=False

        if (i<=35 and i>=11):
            k=1# pierna derecha
        elif (i>=0 and i<=10)or (i<=49 and i>=36):
            k=-1# pierna izquierda

        x=xa
        e_x=x[2][0]-x_reference[i]
        integral_x=integral_x+e_x


        y=ya
        e_y=y[2][0]-y_reference[i]
        integral_y=integral_y+e_y

        l=0
        sumG=0
        while l<=18:
            sumG=sumG+(Gd[l]*x_reference[i+l])
            l=l+1;
        sumG_x=sumG

        l=0
        sumG=0
        while l<=18:
            sumG=sumG+(Gd[l]*y_reference[i+l])
            l=l+1;
        sumG_y=sumG


        operacionA=MM(Gx,x)
        u_x=[[-Gi*integral_x-operacionA[0][0]+sumG_x]]

        operacionA=MM(Gx,y)
        u_y=[[-Gi*integral_y-operacionA[0][0]+sumG_y]]

        nzmp_0_x=u_x[0][0]*Ts+nzmp_1_x
        nzmp_1_x=nzmp_0_x

        nzmp_0_y=u_y[0][0]*Ts+nzmp_1_y
```

```
        nzmp_1_y=nzmp_0_y



        new_lateral_angle=(atan(nzmp_0_y/Zh)*180/pi)
        if nzmp_0_x>0.07:
            nzmp_0_x=0.07
        elif nzmp_0_x<-0.07:
            nzmp_0_x=-0.07

        if new_lateral_angle>max_a_lateral:
             new_lateral_angle=max_a_lateral
        elif new_lateral_angle<-max_a_lateral:
             new_lateral_angle=-max_a_lateral

        posicionMapeo=busBin(position,nzmp_0_x)

        angle_h=angles_position[posicionMapeo][0]
        angle_k=angles_position[posicionMapeo][1]
        angle_t=angles_position[posicionMapeo][2]

        auxCommandR=walking_commands[i][0]
        auxCommandL=walking_commands[i][1]
        new_angles=[angle_h,angle_k,angle_t,new_lateral_angle]



        if k==1:

            new_angles=[angles[0][i],angles[1][i],angles[2][i],new_lateral_angle]

            control_command=mapAngle(new_angles,'r',i)
            control_command=control_command+ auxCommandL
        elif k==-1:

            new_angles=[angles[3][i],angles[4][i],angles[5][i],new_lateral_angle]


            control_command=mapAngle(new_angles,'l',i)
            control_command=control_command+ auxCommandR
        sendCommand(auxCommandR+auxCommandL)
        sendCommand(control_command)
        print(new_lateral_angle)
        while(time.time()-initial_time<=(0.03)):
            pass


        sense_zmp_x=getZMP(0,previous_velocity_x)
        actual_zmp_x=sense_zmp_x[0]
        previous_velocity_x=sense_zmp_x[1]

        sense_zmp_y=getZMP(1,previous_velocity_y)
        actual_zmp_y=sense_zmp_y[0]
        previous_velocity_y=sense_zmp_y[1]

        operacionB=MM(A,x)
        operacionC=MM(B,u_x)
        feedback_x=actual_zmp_x-x[2][0]
        xa[0][0]=operacionB[0][0]+operacionC[0][0]-Lx*(feedback_x)
        xa[1][0]=operacionB[1][0]+operacionC[1][0]-Lx*(feedback_x)
        xa[2][0]=operacionB[2][0]+operacionC[2][0]-Lx*(feedback_x)


        operacionB=MM(A,y)
        operacionC=MM(B,u_y)
        feedback_y=actual_zmp_y-y[2][0]
        ya[0][0]=operacionB[0][0]+operacionC[0][0]-Ly*(feedback_y)
        ya[1][0]=operacionB[1][0]+operacionC[1][0]-Ly*(feedback_y)
        ya[2][0]=operacionB[2][0]+operacionC[2][0]-Ly*(feedback_y)
        i=i+1
    rep=rep+1
```

```
def generateAngles(new_angles,pos):

    mapped_angles_servoR=""
    mapped_angles_servoL=""
    rleg=[0,1,2,6]
    lleg=[3,4,5,7]
    if (pos==cambiofaseA+1 or pos==cambiofaseB+1):
        delta_time_aux=delta_time_transition
    else:
        delta_time_aux=delta_time

    for j in rleg:

mapped_angles_servoR=mapped_angles_servoR+"#"+str(n_servos[j])+"P"+str(round(center_s
ervos[j]+((operation_servos[j]*new_angles[j]*delta_angle[j])/90)))


    for k in lleg:

mapped_angles_servoL=mapped_angles_servoL+"#"+str(n_servos[k])+"P"+str(round(center_s
ervos[k]+((operation_servos[k]*new_angles[k]*delta_angle[k])/90)))

    mapped_angles_servoR=mapped_angles_servoR+"T"+str(delta_time_aux)
    mapped_angles_servoL=mapped_angles_servoL+"T"+str(delta_time_aux)
    return mapped_angles_servoR,mapped_angles_servoL

def mapAngle(new_angles,l,pos):
    if l=='r':
        leg=[0,1,2,6]
    elif l=='l':
        leg=[3,4,5,7]
    mapped_angles_servo=""
    count=0
    if (pos==cambiofaseA+1 or pos==cambiofaseB+1):
        delta_time_aux=delta_time_transition
    else:
        delta_time_aux=delta_time
    for j in leg:

mapped_angles_servo=mapped_angles_servo+"#"+str(n_servos[j])+"P"+str(round(center_ser
vos[j]+((operation_servos[j]*new_angles[count]*delta_angle[j])/90)))
        count=count+1
    mapped_angles_servo=mapped_angles_servo+"T"+str(delta_time_aux)
    return mapped_angles_servo

def mapAngleD(new_angles,l,delta):
    if l=='r':
        leg=[0,1,2,6]
    elif l=='l':
        leg=[3,4,5,7]
    mapped_angles_servo=""
    count=0
    for j in leg:

mapped_angles_servo=mapped_angles_servo+"#"+str(n_servos[j])+"P"+str(round(center_ser
vos[j]+((operation_servos[j]*new_angles[count]*delta_angle[j])/90)))
        count=count+1
    mapped_angles_servo=mapped_angles_servo+"T"+str(delta)
    return mapped_angles_servo


def generateCenter():
    center_CommandR=mapAngleD([0,0,0,0],'r',1500)
    center_CommandL=mapAngleD([0,0,0,0],'l',1500)
    return center_CommandR+center_CommandL



def generateDuck():

    alpha=32
```

```
    beta=-67
    gamma=35
    duck_CommandR=mapAngleD([alpha,beta,gamma,0],'r',1500)
    duck_CommandL=mapAngleD([alpha,beta,gamma,0],'l',1500)
    return duck_CommandR+duck_CommandL
def startPosition():

    alpha=32
    beta=-67
    gamma=35
    duck_CommandR=mapAngleD([alpha,beta,gamma,18],'r',1500)
    duck_CommandL=mapAngleD([alpha,beta,gamma,18],'l',1500)
    return duck_CommandR+duck_CommandL




print("Communicating with Servo Controller")
serController=startSerialController()
serController.isOpen()
print("Communicating with MPU 6050")
serGiroscope=startSerialGiroscope()
serGiroscope.isOpen()

# pos x, pos y, pos z, ace x, ace y, ace z
vector_data=['A','B','C','D','E','F']
sensor_data=[0,0,0,0,0,0]



delta_time=40
#- Cycle Generation and Servo Mapping
#- # of Servo [r_h,r_k,r_t,l_h,l_k,l_t,r_t,l_t]
n_servos=[4,3,2,10,9,11,1,7]
#  Operation
#- Right Leg: -+- -
#- Left Leg: +-+ -

print("Generating reference for Frontal Stability")
generateLateralreference()



operation_servos=[-1.0,1.0,-1.0,1.0,-1.0,1.0,1.0,1.0]

fc=600
delta_angle=[fc,fc,fc,fc,fc,fc,fc,fc]


print("Generating  Commands")
center_command=generateCenter()
duck_command=generateDuck()
walking_commands=generateCommands()
center_command=generateCenter()
start_position=startPosition()
sendCommand(center_command)
time.sleep(2.0)


while 1 :
    Tinput = input('>> ')
    if Tinput == 'salir':
        serSer.close()
        exit()
    elif Tinput =='stand': #stand position
        sendCommand(center_command)
    elif Tinput =='duck': #duck
        sendCommand(duck_command)
    else:
        sendCommand(start_position)
        time.sleep(1.5)
```

```
        walkingCycle()
        sendCommand(duck_command)
        time.sleep(1.5)
        sendCommand(center_command)
```