UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Estabilización Orbital: Aplicación en el prototipo Butterfly Robot

T E S I S

Que para obtener el título de

Ingeniero Eléctrico-Electrónico

P R E S E N T A

Luis Eduardo Silva Franco

DIRECTOR DE TESIS

Dr. Rafael Iriarte Vivar-Balderrama

Ciudad Universitaria, Cd. Mx., 2024

Jurado asignado

Presidente:	Dr. Leonid Fridman
Secretario:	Dr. Hoover Mujica Ortega
Vocal:	Dr. Rafael Iriarte Vivar-Balderrama
1^{er} suplente:	M.I. Ulises Arturo Pérez Ventura
2^{do} suplente:	Dr. Juan Gustavo Rueda Escobedo

Ciudad Universitaria, Departamento de Control y Robótica, Laboratorio de Control por Modos Deslizantes.

Ciudad de México.

Director de tesis

Dr. Rafael Iriarte Vivar-Balderrama

Dedicatoria

A mis padres Julieta Franco y Eduardo Silva, así como a mi segunda madre Silvia Salguero, que lo son todo.

Agradecimientos

Agradezco a la Universidad Nacional Autónoma de México por brindarme todas las herramientas técnicas, científicas y humanas para cursar y finalizar mis estudios de ingeniería.

A mi profesor y director de tesis, el Dr. Rafael Iriarte Vivar-Balderrama, por acompañarme, guiarme y aconsejarme en todo el camino correspondiente a este trabajo de tesis. Además de brindarme su amistad y su experiencia durante la última instancia de mi carrera.

A mi profesor y mentor, el Dr. Leonid Fridman por enseñarme que el camino de la investigación y la ciencia, está lleno de enseñanzas y aprendizajes que van más allá de lo técnico; y por demostrarme que es posible encontrar una familia dentro de ella.

A mi profesor, el Dr. Hoover Mujica Ortega por ser un ente principal en mi desarrollo profesional y personal, así como por su apoyo incondicional en todo el proceso de titulación.

A mis sinodales y profesores, el M.I. Ulises Arturo Pérez Ventura y el Dr. Juan Gustavo Rueda Escobedo, por aportarme su conocimiento y su maestría dentro de la realización de esta tesis.

A todos y cada uno de mis profesores durante la carrera por inculcarme siempre el deseo por aprender, conocer y disfrutar de la ingeniería; de manera especial al Ing. Raúl Puente Mancilla por infundir y cultivar en mí el deseo de ser investigador.

A mis compañeros y amigos Alejandro León y Alejandro Lagunas, por acompañarme en todo este proceso y ser pieza clave en mi carrera. También a mi amigo José Luis Bautista por su constante apoyo y atención para mejorar mi investigación.

A todos los miembros del Laboratorio de Control por Modos Deslizantes por su valioso apoyo y guía. A Andrés, Luis, José Antonio y Areli por sus enseñanzas y amistad.

A mis contribuidores, el profesor Maksim Surov por siempre mostrarme su disposición al resolver mis dudas; y a la Mtra. Taisia Medvedeva por proveerme su invaluable material y conocimiento referente a la estabilización orbital.

A toda mi familia y mis amigos, por ser los principales artifices la persona que soy ahora, al fomentarme, inculcarme y enseñarme el valor de estar acompañado.

A la Dirección General de Asuntos del Personal Académico (DGAPA) de la UNAM por el apoyo y la beca brindada en el Proyecto UNAM-PAPIME PE115224.

A todos ellos dedico el presente trabajo, porque me han demostrado que los triunfos y la superación siempre son mejores y más grandes si cuento con su apoyo.

Resumen

Dentro del presente trabajo de tesis se aplica el concepto de estabilización orbital en el control de un sistema dinámico no prensil conocido como *Buttterfly Robot*. La estabilidad orbital se destaca por la convergencia y estabilidad de una trayectoria cerrada (órbita) prescindiendo de la dependencia temporal. En la literatura actual existen pocos métodos que provean lo anterior, pero destaca uno en particular: el control por linealización de coordenadas transversales.

El controlador previamente mencionado es implementado y validado experimentalmente en el prototipo *Butterfly Robot* desarrollado por *Robotikum* dentro del Laboratorio de Modos Deslizantes de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México. El control propuesto cumple con llevar al sistema a una órbita deseada y mantenerla como un ciclo límite estable. Además, se presenta un análisis extenso del software utilizado para su implementación experimental.

Índice general

Índice de figuras

Acrónimos

LTIs Sistemas lineales invariantes en el tiempo (LTIs, por sus siglas en inglés). xi, 10, 14, 17, 20, 25, 36, 37

LTVs Sistemas lineales variantes en el tiempo (LTVs, por sus siglas en inglés). xi, xii, xiv, 14, 20, 22, 25, 31, 32, 37, 55, 56, 57, 70, 71, 76, 80

BIBO Entrada acotada - salida acotada (BIBO, por sus siglas en inglés). xi, xiii, 32, 33

TCP Protocolo de Control de Transmisión (TCP, por sus siglas en inglés). 4, 57, 60, 61, 65

LQR Regulador Lineal Cuadrático (LQR, por sus siglas en inglés). 22, 56, 79

HJB Ecuación diferencial parcial de Hamilton-Jacobi-Bellman. 23, 24

ESL Estable en el sentido de Lyapunov. 34, 37

AESL Asintóticamente estable en el sentido de Lyapunov. 34, 37

ISL Inestable en el sentido de Lyapunov. 34, 37

GAS Asintóticamente estable de manera global (GAS, por sus siglas en inglés). 34, 37

USB Bus Serie Universal (USB, por sus siglas en inglés). 60, 61

SMC Control por modos deslizantes (SMC, por sus siglas en inglés). 80

Capítulo 1

Introducción

El éxito de los robots se debe en gran medida a que automatizan tareas que para los humanos son repetitivas o peligrosas. Dentro de todos los enfoques de la robótica, el principal es la manipulación de objetos, por su aplicación en entornos comúnes como operaciones de *pick and place*, soldadura, pintura, entre otros. [Sætre, 2022]

Conforme los sistemas robóticos continuan evolucionando, se espera que realicen acciones y maniobras más complejas, esto promueve estudiar nuevos desafíos de control. Uno de los mayores retos en la actualidad son los sistemas subactuados, donde el controlador no puede actuar directamente en todos los grados de libertad; es por esto, que los métodos clásicos de planificación y control de movimientos aplicados en sistemas subactuados a menudo no son adecuados para resolver este tipo de problemas.

Para superar los desafíos que los sistemas subactuados presentan, se han desarrollado distintas metodologías para su solución. Uno de los enfoques más recientes es la estabilización orbital que se centra en lograr que las dinámicas del sistema converjan al movimiento deseado, sin importar la dependencia temporal de los estados; es decir, encontrar una ley de control que haga que el sistema admita el movimiento deseado como una órbita asintóticamente estable.

Aunque ya existen métodos para estabilizar puntos de equilibrio y órbitas periódicas, todavía existe la necesidad de crear estrategias de propósito general, aplicables a una amplia gama de sistemas dinámicos no lineales y subactuados. La estabilización orbital ofrece una solución prometedora para el control de movimientos complejos en robótica avanzada.

1.1. Motivación

El desarrollo de sistemas de control que sigan trayectorias, es una tarea que normalmente recae en el enfoque de *tracking control*, el cual funciona muy bien en la mayoria de los ámbitos aplicables. Sin embargo, existen algunas implementaciones en las cuales es de mayor interés asegurar la convergencia con la trayectoria deseada antes que medir el tiempo en que eso ocurre. Por ello, la estabilización orbital ha sido aplicada principalmente en robots bípedos ([Hamed y Grizzle, 2013]), donde es más importante lograr que el robot camine adecuadamente sin considerar el tiempo en que lo logre.

La aplicación de la estabilización orbital en los sistemas de control no se acota únicamente a los robots bípedos, si no también, a otros sistemas subactuados. En esta tesis, las aplicaciones que generan especial interés son aquellas en las que el actuador no puede sostener al objeto a controlar, y que se conocen como sistemas no prensiles, [Lynch y Mason, 1999]. En la vida cotidiana aplicamos la acción no prensil comúnmente, por ejemplo cuando empujamos algo con nuestra mano para moverlo; por lo que se espera que si un robot manipulador quiere imitar a los movimientos de un humano, también pueda realizar movimiento no prensil.

Dentro del Laboratorio de Modos Deslizantes de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México, se encuentra un prototipo real que presenta un movimiento no prensil, conocido como *Butterfly Robot*, el cual fue desarrollado por *Robotikum* y será el eje de estudio de este trabajo de tesis.

El artículo que aplica la metodología del control por linealización de coordenadas transversales para el *Butterfly Robot* fue desarrollado por [Surov*, et al.*, 2015], y aunque ofrece la solución teórica-experimental, no brinda un estudio profundo del problema; de esta manera, esta tesis también servirá como un documento complementario para el artículo antes mencionado.

1.2. Antecedentes

Los sistemas no prensiles presentan una dificultad alta en cuanto a su entendimiento práctico y teórico; ya que, al no sujetar el objeto que controlan, los controladores no pueden actuar directamente en todos los grados de libertad, sobre todo porque en ocasiones estos grados de libertad incluso pueden desaparecer y reaparecer durante el movimiento, [Surov*, et al.*, 2015].

A lo largo de la historia han existido múltiples aplicaciones de sistemas no prensiles, como el *Ball and Beam* ([Hauser*, et al.*, 1992]), el *Ball-on-disk* ([Ryu*, et al.*, 2013]), el *Acrobot* ([Spong, 1994]) o el *Pendubot* ([Zhang y Tarn, 2002]). Todos estos sistemas han sido controlados a partir de distintas metodologías. Sin embargo, de entre todos ellos existe uno que resalta, pues la primer solución experimental se consiguió 16 años después de su concepción, estamos hablando del ***Butterfly Robot*** (Figura 1.1).

El *Butterfly Robot* fue introducido por [Lynch*, et al.*, 1998], bajo la premicia de realizar un *malabarismo* con la mano. Presentar un modelo exacto de este sistema es una tarea sumamente complicada, pues por el tamaño y su dinámica, no es posible linealizarlo en ningún punto de equilibrio. Esta complejidad en el modelado originó algunas aproximaciones importantes pero no completas de su representación matemática, [Surov*, et al.*, 2015].

En el artículo presentado por [Cefalo*, et al.*, 2006], se hacen dos consideraciones que no son triviales: la primera es que la pelota es tratada como un punto de masa, desconsiderando así los efectos que el radio de la pelota provoca en el modelado; y, en segundo lugar, se desprecia la energía cinética que la bola obtiene al girar. Lo anterior no permite tener un modelo completo del sistema, pues para generar un movimiento en la pelota forsozamente se provoca un giro; no considerar la energía que el movimiento rotacional de la bola imprime al sistema puede acarrear errores potencialmente graves

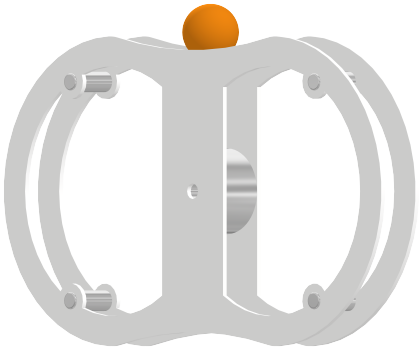


Figura 1.1 *Butterfly Robot.*

en su comportamiento en lazo cerrado. En el artículo previamente mencionado, aunque se consiguen resultados teóricos, la implementación experimental no es presentada.

Un segundo enfoque fue realizado por [Morales, *et al.*, 2013] donde, a partir de las consideraciones hechas en el artículo anterior, generan una parametrización de la curva en el plano (x, y), con la limitante debida a la aparición de discontinuidades dentro de los lóbulos más pronunciados de la mariposa e imposibilitando la implementación de un lazo de control en tiempo real.

No fue hasta que [Surov, *et al.*, 2015] presentó finalmente la implementación experimental, y un modelo matemático más completo que los anteriores. En ese trabajo se considera la rotación de la pelota en la mariposa, y se redefine la parametrización utilizando una única variable polar, que evita las discontinuidades en los lóbulos más pronunciados de la mariposa, y permite su implementación en tiempo real. Además, resuelve el problema de control bajo el esquema presentado por [Shiriaev, *et al.*, 2005], el cual, hace uso de un enfoque de control basado en estabilidad orbital. Cabe destacar que el profesor Maksim Surov, desarrolló un ambiente completo de programación que permite la ejecución real del algoritmo de control, por lo cual su artículo será una de las principales referencias para este trabajo de tesis.

1.3. Objetivos

Los objetivos de esta tesis son los siguientes:

Objetivo General

Desarrollar el algoritmo de control por *linealización de coordenadas transversales* basado en la estabilización orbital para el prototipo conocido como *Butterfly Robot* y evaluar experimentalmente su comportamiento.

Objetivos Particulares

- Deducir el modelo matemático que representa al sistema dinámico.

- Explicar el *software* correspondiente a la implementación experimental.
- Evaluar experimentalmente al controlador.
- Evidenciar la ventaja que presenta el enfoque de estabilización orbital con respecto al *tracking control*.

1.4. Contribuciones

Antes se mencionó que una de las principales motivaciones de este trabajo es complementar los resultados de [Surov*, et al.*, 2015]. Por ello, las contribuciones son las siguientes:

- Se dedujo el modelo matemático completo para el sistema *Butterfly Robot* presentado en [Surov*, et al.*, 2015], a partir del algoritmo de modelado utlizando ecuaciones de Euler-Lagrange.
- Se fundamentó el uso del algoritmo de control por linealización de coordenadas transversales (propuesto por [Shiriaev*, et al.*, 2005]) como una alternativa para resolver el problema de estabilización orbital.
- Se implementó computacionalmente el algoritmo previamente mencionado a través del entorno de *Linux* y *C++*, utilizando dos computadoras comunicadas mediante el protocolo TCP.
- Se presentó una explicación detallada del *software* que permite la implementación experimental.
- Se elaboró un manual de uso de las herramientas computacionales.
- Se optimizó la programación para la implementación computacional del sistema de control en lazo cerrado.
- Se realizaron pruebas con perturbaciones controladas para el prototipo *Butterfly Robot*, y se presentaron resultados experimentales.
- Se desarrolló un modelo 3D de la planta *Butterfly Robot*, con el fin de representar a los elementos del sistema detalladamente y ofrecer una herramienta de manipulación virtual.

1.5. Organización de la tesis

La presente tesis se encuentra dividida en cinco capítulos, este primero compete a la **Introducción**. El contenido de los siguientes cuatro capítulos se describe a continuación:

En el **Capítulo 2** se presenta el marco teórico correspondiente al concepto de estabilización orbital. Para ello, se muestra un desarrollo completo sobre sistemas dinámicos, su modelado y representaciones más comúnes; posteriormente, se estudian los sistemas de control por retroalimentación de estados, centrándonos en los objetivos de control, y los enfoques lineales y no lineales que se utilizarán en el presente documento, de los cuales el **control por linealización de coordenadas transversales** será el más importante. Finalmente, se explica el concepto de estabilidad, partiendo desde su forma más general hasta el concepto específico de estabilidad orbital.

Dentro del **Capítulo 3** se presenta la implementación del algoritmo de **control por linealización de coordenadas transversales** en el prototipo *Butterfly Robot*. En primer lugar se expone el planteamiento del problema a resolver, se introduce al prototipo desarrollado por *Robotikum*, y se realiza el modelado matemático de forma detallada. En la segunda sección se propone el esquema de control general, y se encuentra la entrada de control que cumple con el problema de estabilización orbital. Por último, la tercer sección desarrolla el ámbito computacional necesario para el correcto funcionamiento experimental de la planta.

Con el fin de validar experimentalmente el funcionamiento del controlador, en el **Capítulo 4** se muestran los resultados de la implementación desarrollada, bajo dos escenarios. El primer escenario evalúa al sistema sin presencia de perturbaciones, con el propósito de valorar el desempeño del controlador. El segundo escenario presenta los mismos experimentos pero con presencia de perturbaciones, a fin de comprobar la robustez del sistema de control.

Por último, en el **Capítulo 5** se exponen las conclusiones generales de la tesis y se comentan los posibles trabajos futuros que extiendan el tema.

Los códigos desarrollados en *C++*, correspondientes a la implementación computacional del **Capítulo 3**, se presentan en el **Apéndice A**.

Capítulo 2

Marco Teórico

Dentro de este trabajo de tesis se abordará principalmente la noción de órbitas y su estabilización mediante algoritmos de control por realimentación de estados, aplicado en un prototipo conocido como *Butterfly Robot*. A continuación se presentan todos los conceptos teóricos necesarios para su desarrollo y comprensión.

2.1. Sistemas dinámicos

2.1.1. Definición

Un sistema dinámico describe el comportamiento de fenómenos mecánicos, eléctricos, hidráulicos y térmicos mediante el uso de ecuaciones diferenciales, álgebra matricial y física elemental, [Lobontiu, 2010].

El comportamiento dinámico de los sistemas puede ser obtenido mediante descripciones matemáticas, [Antsaklis y Michel, 2007]. Lo anterior se puede reducir a encontrar las ecuaciones diferenciales que gobiernan la conducta del sistema con el fin de obtener su respuesta en el tiempo. De manera general las ecuaciones diferenciales de movimiento son del tipo:

$$y^{(n)} + a_1 y^{n-1} + \ldots + a_{n-1} y + a_n = x^{(n)} + b_1 x^{n-1} + \ldots + b_{n-1} x + b_n. \tag{2.1}$$

Donde a_i y b_i son parámetros, x es la variable independiente, y es la variable dependiente y n es el orden de la derivada temporal máxima.

Sin embargo, conocer los parámetros y el orden real de la ecuación diferencial para un fenómeno específico, no es un tarea trivial; no obstante, es posible idear una serie de pasos para obtener una representación matemática suficientemente fiel al comportamiento real. A este proceso se le llama **modelado**, y es una de las principales herramientas para el estudio de sistemas dinámicos (el esquema para obtener el modelado matemático de los sistemas se presenta en la figura 2.1).

2.1.2. Modelado de Sistemas Dinámicos

Existen diversas metodologías para el modelado de sistemas físicos, pero todas parten del mismo principio: identificar sus propiedades físicas fundamentales. Estas variables principales deben ser las mí-

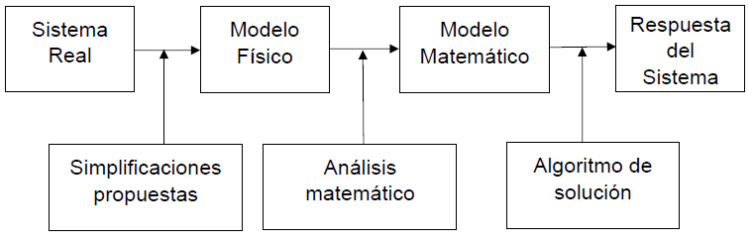


Figura 2.1 Esquema de modelado de sistemas dinámicos.

nimas necesarias para poder describir satisfactoriamente la dinámica del sistema y se les llama **estados**.

Debido a la cantidad de métodos para modelar sistemas, podrían existir distintos modelos (descripciones matemáticas) en función de lo que queremos conocer de cada caso de estudio, [Antsaklis y Michel, 2007]. Por esta razón, se acotará este marco teórico a una de las técnicas más utilizadas en la ingeniería, el **método de Euler-Lagrange**.

2.1.3. Modelado mediante las ecuaciones de Euler-Lagrange

Las ecuaciones de Euler-Lagrange permiten modelar un sistema mediante el análisis de su energía *cinética*, su energía *potencial* y la energía *disipada*, a partir de las variables de estado, [Hendricks, *et al.*, 2008].

Se puede definir una clase de **algoritmo** para obtener el modelo matemático de un sistema mecánico.

1. Identificar los grados de libertad (variables de estado) asociados al sistema. Las variables de estado deben ser **linealmente independientes** entre sí. Estas variables las denotaremos con $\mathbf{q} \in \mathbb{R}^{\mathbf{n}}$ [1], donde n es el número de estados mínimos que definen al sistema.

$$\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix}$$

2. Obtener el vector de posición de cada elemento almacenador de energía del sistema en función de las variables de estado.

$$\vec{p_1}(q_1, q_2, ..., q_n)$$
$$\vec{p_2}(q_1, q_2, ..., q_n)$$
$$\vdots$$
$$\vec{p_m}(q_1, q_2, ..., q_n)$$

[1] A lo largo de esta tesis utilizaremos la notación en **negrita** para referirnos a variables que son *vectores* o *matrices*. Para los *vectores* también será utilizada la notación $\vec{a}$.

3. Obtener el vector de velocidad y su norma (rapidez) al cuadrado de cada elemento almacenador ($v^2 = |\vec{v}|^2$).

$$\vec{v_1}(q_1, q_2, ..., q_n) = \frac{d}{dt}\vec{p_1}$$
$$\vec{v_2}(q_1, q_2, ..., q_n) = \frac{d}{dt}\vec{p_2}$$
$$\vdots$$
$$\vec{v_m}(q_1, q_2, ..., q_n) = \frac{d}{dt}\vec{p_m}$$

4. Calcular con base en cada variable de estado:

   a) *Energía cinética* de cada masa (m).

$$k = \frac{1}{2}m|\vec{v}|^2$$

   donde k es la energía cinética [J] y $|\vec{v}|$ es la rapidez $[\frac{m}{s}]$.

   b) *Energía potencial* de cada masa (en campo gravitatorio o anclada a un resorte).

$$E_p = mgh$$
$$E_k = K\Delta x$$

   donde E_p es la energía potencial [J], g es la constante de aceleración debido a la gravedad ($9.81[\frac{m}{s^2}]$), h la altura, E_k la energía potencial de un resorte [J], K la constante del resorte $[\frac{J}{m}]$ y Δx la elongación del resorte [m].

   c) *Disipación de Rayleigh* de cada elemento disipador.

$$P_{b_i} = \frac{1}{2}b_i(\Delta v)^2$$
$$P_{b_\omega} = \frac{1}{2}b_\omega(\Delta\omega)^2$$

   Donde P_{b_i} es la energía disipada en cada elemento disipador lineal [J], b_i es la constante de disipación de cada elemento disipador lineal $[\frac{Js}{m}]$, y Δv es la diferencia de velocidad en cada extremo del elemento disipador $[\frac{m}{s}]$. Los elementos ω son los similes para elementos de disipación angulares.

5. Obtener la energía cinética, potencial y de disipación total.

$$k_T = k_1 + k_2 + \ldots + k_m$$
$$E_{P_T} = E_{P_1} + E_{P_2} + \ldots + E_{P_m}$$
$$P_{b_T} = P_{b_1} + P_{b_2} + \ldots + P_{b_m}$$

6. Calcular el **Lagrangiano** del sistema.

$$\mathcal{L} = K_T - E_{P_T} \tag{2.2}$$

7. Calcular las **ecuaciones de movimiento** asociadas a cada variable de estado, con base en las **ecuaciones de Euler-Lagrange**.

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} + \frac{\partial P_{b_T}}{\partial \dot{q}_i} = \sum f_i \tag{2.3}$$

donde q_i y $\dot{q}_i$ son las variables de estado y sus respectivas derivadas. y f_i son las fuerzas que actuan sobre el sistema [N]. Cabe recalcar que f_i es positiva (+) si va en sentido de la variable de estado analizada, y negativa (-) si va en sentido contrario; además, en caso de que la variable asociada sea de **traslación**, f_i es una **fuerza**, mientras que si es de **rotación** es un **par de fuerzas**.

8. Obtener las ecuaciones de estado a partir de las ecuaciones de movimiento.

Se elige el vector de estados como $x = \begin{bmatrix} q(t) \\ \dot{q}(t) \end{bmatrix}$, que implica $\dot{x} = \begin{bmatrix} \dot{q}(t) \\ \ddot{q}(t) \end{bmatrix}$. Se obtiene $\ddot{q}(t)$ a partir de las ecuaciones de movimiento.

2.1.4. Representación mediante la función de transferencia

Una vez que contamos con el modelo dinámico del sistema, es momento de adentrarnos en sus representaciones típicas, de las cuales el primer acercamiento es **la función de transferencia**. Las funciones de transferencia son de gran utilidad para representar sistemas lineales [2], del tipo invariantes en el tiempo (LTIs, por sus siglas en inglés), lo que significa que los coeficientes de la ec.(2.1), que multiplican a las variables de estados son constantes.

Este tipo de sistemas se pueden transformar fácilmente del dominio temporal al dominio en el espacio de las frecuencias mediante el uso de la **transformada de Laplace**, con el fin de convertir la ecuación diferencial en un simple problema algebraico, [Nise, 2015]. De la ecuación diferencial general descrita en la ec.(2.1), podemos encontrar su transformada de Laplace:

$$\begin{gathered} s^n Y(s) + a_1 s^{n-1} Y(s) + \ldots + a_n + \text{términos por condiciones iniciales} \\ = \\ s^n X(s) + b_1 s^{n-1} X(s) + \ldots + b_n + \text{términos por condiciones iniciales} \end{gathered}$$

Con el fin de simplicar las ecuaciones, suponemos que todas las condiciones iniciales son nulas, encontrando así una transformación de la siguiente manera:

$$(s^n + a_1 s^{n-1} + \ldots + a_n) Y(s) = (s^n + b_1 s^{n-1} + \ldots + b_n) X(s). \tag{2.4}$$

Como se estudió en la sección 2.1.1, nos interesa conocer el comportamiento del sistema dinámico y obtener su respuesta en el tiempo; aunque es posible analizar la evolución interna del sistema, también es posible analizar su salida[3] a partir de una entrada propuesta, para conocer detalles de su conducta temporal. En la ec.(2.4), notamos que existen 2 polinomios: uno multiplica a la variable dependiente o salida ($Y(s)$) y otro que multiplica a la variable independiente o entrada ($X(s)$); como es necesario

[2] Los sistemas lineales son aquellos que siempre cumplen con *homogeneidad* ($y = f(u) \implies f(a \cdot u) = a \cdot y$ con $a = cte \in \mathbb{R}$) y *aditividad* ($f(u_1) = y_1, f(u_2) = y_2 \implies f(u_1 + u_2) = y_1 + y_2$), que al final garantizan *superposición* ($f(a \cdot u_1 + b \cdot u_2) = a \cdot y_1 + b \cdot y_2$ con $a, b = cte \in \mathbb{R}$), [Chen, 1999].

[3] La salida de un sistema es la variable o el conjunto de variables que nos interesa medir o conocer.

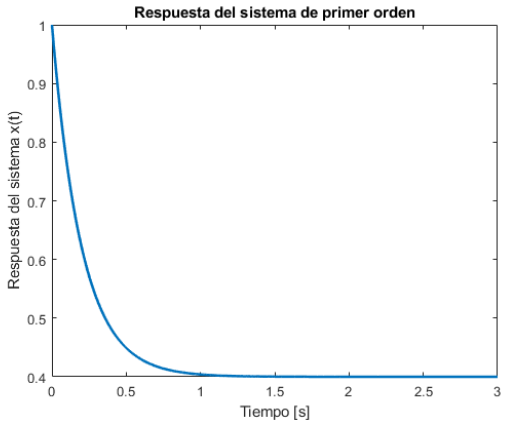
conocer la salida en relación de su entrada, es posible reescribir nuestra igualdad de polinomios de la siguiente manera:

$$\frac{Y(s)}{X(s)} = \frac{s^n + b_1 s^{n-1} + \ldots + b_n}{s^n + a_1 s^{n-1} + \ldots + a_n} \tag{2.5}$$

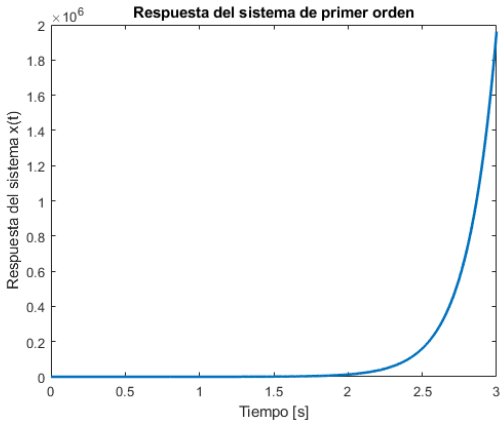
A la ec.(2.5) se le conoce como la **función de transferencia**, y no es más que la relación entre la entrada ($X(s)$) y la salida ($Y(s)$) de una sistema. Esta representación es de vital importancia para el análisis de los sistemas dinámicos; ya que, permite obtener la salida (respuesta) del sistema en función de una entrada conocida. Sin embargo, lo más importante para aprovechar de la función de transferencia, es precisamente el hecho de que se tiene un polinomio tanto en el numerador como en el denominador.

Recordando conceptos básicos del álgebra, todos los polinomios tienen raíces, es decir, valores para los cuales si el polinomio es evaluado su resultado será cero. En los sistemas dinámicos, las raices de los polinomios de la función de transferencia tienen una denotación importante, pues ayudan a identificar características fundamentales del sistema. Si las raíces son del polinomio del numerador, se les llama **ceros**, al provocar que la función de transferencia de como resultado cero a pesar de la entrada; si las raíces son del denominador, se les llama **polos** y provocan que la función de transferencia se vuelva indefinida, [Nise, 2015].

Los polos de un sistema no solo dan información sobre su estabilidad, además brindan información sobre su comportamiento. En el caso de un sistema de primer orden el comportamiento de la salida siempre es parecido a los mostrado por las figuras 2.2a y 2.2b, no obstante, para sistemas de orden superior, los polos tienen especial repercusión en la salida del sistema.

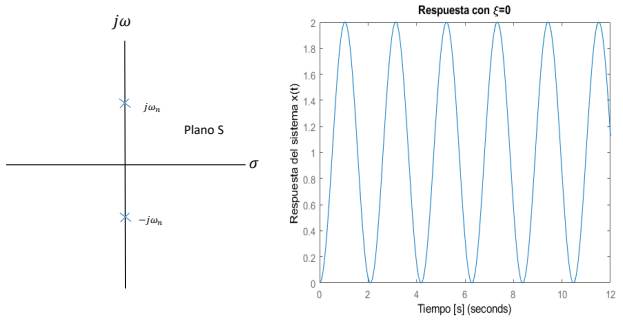


(a)

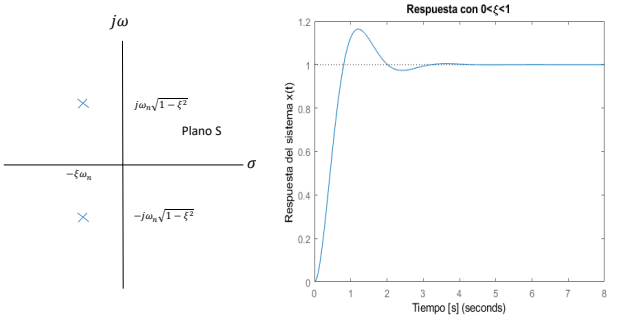


(b)

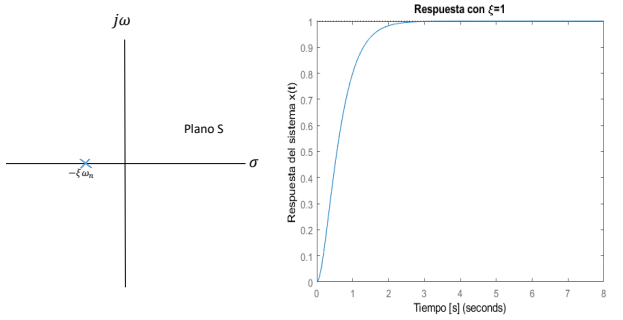
Figura 2.2 Respuesta al escalón unitario del sistema con función de transferencia: a) $\frac{X(s)}{U(s)} = \frac{s+2}{(s+5)}$ y b) $\frac{X(s)}{U(s)} = \frac{s+2}{(s-5)}$.



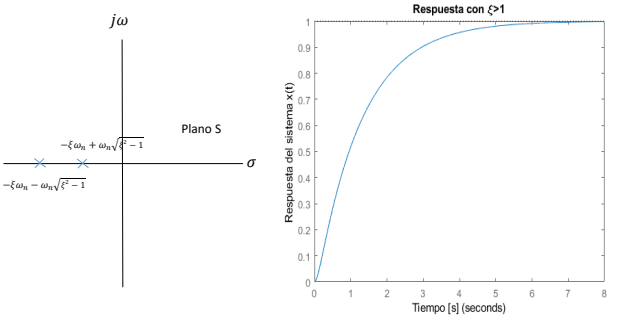
(a) Respuesta al escalón unitario con $\xi = 0$ (No Amortiguado).



(b) Respuesta al escalón unitario con $0 < \xi < 1$ (Subamortiguado).



(c) Respuesta al escalón unitario con $\xi = 1$ (Críticamente amortiguado).



(d) Respuesta al escalón unitario con $\xi > 1$ (Sobreamortiguado).

Figura 2.3 Posicionamiento de polos para un sistema de segundo orden.

2.1.4.1. Polos y ceros en sistemas de segundo orden

Exploremos el sistema de segundo orden, en el que su función de transferencia tiene la siguiente forma:

$$\frac{Y(s)}{X(s)} = \frac{{\omega_n}^2}{s^2 + 2 + 2\xi\omega_n + {\omega_n}^2}$$

Para este sistema, los polos están dados por:

$$s_{1,2} = -\xi\omega_n + -\omega_n\sqrt{\xi^2 - 1} \tag{2.6}$$

Con ξ como el factor de amortiguamiento y $\omega_n[\frac{rad}{s}]$ la frecuencia natural del sistema.

Según la naturaleza de las soluciones[4], la respuesta ante una entrada conocida cambiará su desempeño. La ec.(2.6) tiene cuatro posibles casos[5] en función del **factor de amortiguamiento** ξ:

- **No amortiguado** ($\xi = 0$): Al no contar con amortiguamiento, el sistema se mantendrá oscilando periódicamente (Figura 2.3a). Los polos tiene únicamente parte imaginaria y son conjugados, i.e. $s_1 = 0 + j\omega_n$ y $s_2 = 0 - j\omega_n$.
- **Subamortiguado** ($0 < \xi < 1$): Cuando el factor de amortiguamiento es menor a 1, el sistema cuenta con una pequeña oscilación que posteriormente desaparece, a este efecto se le llama **sobrepaso** y es muy común en sistemas de 2do orden (Figura 2.3b). Los polos son complejos conjugados con parte real negativa.
- **Críticamente amortiguado** ($\xi = 1$): En este caso, el sistema presenta un comportamiento parecido al de primer orden, esto significa que se elimina el sobrepaso (Figura 2.3c). Los polos son enteramente reales negativos e iguales ($s_1 = s_2 = -\xi\omega_n$).
- **Sobreamortiguado** ($\xi > 1$): El comportamiento es muy parecido al caso **críticamente amortiguado**, pero es más lento (Figura 2.3d). Los polos son enteramente reales negativos y distintos entre sí.

Describiendo los anteriores comportamientos, notamos que al modificar los polos de un sistema dinámico, podemos hacer que se comporte de la manera en que deseemos. Esta idea será rescatada y estudiada en la sección 2.2.4.

2.1.5. Representación en el Espacio de Estados

Aunque desde el punto de vista clásico de control es común utilizar a la **función de transferencia** como una representación de un sistema dinámico, para este caso de estudio vale la pena utilizar una representación más general.

La representación del espacio de estados, se basa en la utilización de las 3 principales variables para el estudio de un sistema: las variables de estado, las variables de entrada y las variables de salida, [Ogata, 2010]. Una de sus características esenciales, es que este tipo de representación no está limitado a sistemas lineales, si no que también se puede obtener una extensión para sistemas no lineales.

La forma general para la representación en el espacio de estados es la siguiente:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t). \end{aligned} \tag{2.7}$$

Donde:

- $\mathbf{x}(t)$ es el vector de estados, y es de dimensión n, $\mathbf{x}(t) \in \mathbb{R}^n$.
- $\dot{\mathbf{x}}(t)$ es el vector de derivadas temporales de los estados y es de dimensión n, $\dot{\mathbf{x}}(t) \in \mathbb{R}^n$.
- $\mathbf{u}(t)$ es el vector de entradas, y es de dimensión m, $\mathbf{u}(t) \in \mathbb{R}^m$.

[4]Por naturaleza me refiero a si son números enteramente reales, enteramente imaginarios o complejos.
[5]Estos casos son suponiendo una entrada escalón unitaria.

- $\mathbf{y}(t)$ es el vector de salidas, y es de dimensión r, $\mathbf{y}(t) \in \mathbb{R}^r$.

Y además, $\mathbf{f}$ y $\mathbf{g}$ son funciones vectoriales que, a su vez, se componen de funciones escalares f_i y g_i, [Hendricks*, et al.*, 2008]:

$$\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) = \begin{bmatrix} f_1(\mathbf{x}(t), \mathbf{u}(t), t) \\ f_2(\mathbf{x}(t), \mathbf{u}(t), t) \\ \vdots \\ f_n(\mathbf{x}(t), \mathbf{u}(t), t) \end{bmatrix}, \qquad \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t) = \begin{bmatrix} g_1(\mathbf{x}(t), \mathbf{u}(t), t) \\ g_2(\mathbf{x}(t), \mathbf{u}(t), t) \\ \vdots \\ g_r(\mathbf{x}(t), \mathbf{u}(t), t) \end{bmatrix}.$$

Esta representación es general para sistemas lineales o no lineales. En el desarrollo de este trabajo se utilizarán ambos tipos de sistemas dinámicos; en consecuencia, es importante mostrar sus representaciones más comunes.

El espacio de estados de un sistema lineal es un caso específico de la ec.(2.7), y matemáticamente se puede expresar de la siguiente manera,

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t), \\ \mathbf{y}(t) &= \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t). \end{aligned} \tag{2.8}$$

Con parámetros usualmente conocidos como:

- $\mathbf{A}$ es la matriz de dinámica. $\mathbf{A}(t) \in \mathbb{R}^{n \times n}$.
- $\mathbf{B}$ es la matriz de entradas. $\mathbf{B}(t) \in \mathbb{R}^{n \times m}$.
- $\mathbf{C}$ es la matriz de salidas. $\mathbf{C}(t) \in \mathbb{R}^{r \times n}$.
- $\mathbf{D}$ es la matriz de prealimentación (*feed forward*). $\mathbf{D}(t) \in \mathbb{R}^{r \times m}$.

Es importante observar que la ec.(2.8), tiene parámetros variantes en el tiempo dentro de las matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ y $\mathbf{D}$ (por lo que son llamados **sistemas lineales variantes en el tiempo** - LTVs, por sus siglas en inglés -); debido a esto, si se trabaja con un **sistema lineal invariante en el tiempo** (LTIs, por sus siglas en inglés), estas matrices pierden su dependencia temporal, es decir,

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t). \end{aligned} \tag{2.9}$$

Para los LTIs, la información contenida en la matriz $\mathbf{A}$ es de especial relevancia, pues los **eigenvalores** de dicha matriz corresponden a los polos del sistema.

Los polos en el **espacio de estados** son equivalentes a los polos obtenidos por la **función de transferencia**, e incluso, por la naturaleza de esta representación es posible obtener una matriz de funciones de transferencia; así, los fenómenos estudiados en la sección 2.1.4 también suceden para esta representación.

El espacio de estados de un sistema no lineal, puede ser representado directamente con la ec.(2.7), sin embargo, existe una representación más usada para aplicaciones de robótica, la **forma del manipulador**, y será descrita en la siguiente sección.

2.1.6. Forma del Manipulador para Sistemas Mecánicos Subactuados

Para los sistemas mecánicos se puede describir un modelo matemático en función de la ec.(2.3), la cual es reescrita en función de los estados ($\mathbf{q}$ y $\dot{\mathbf{q}}$) y sus entradas ($\mathbf{u}$).

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}(\mathbf{q},\dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial \mathcal{L}(\mathbf{q},\dot{\mathbf{q}})}{\partial \mathbf{q}} = \mathbf{B}_u\mathbf{u} \tag{2.10}$$

En la ec.(2.10), $\mathbf{B}_u \in \mathbb{R}^{n\times m}$ es una matriz de acoplamiento.

También se puede reformular la definición del lagrangiano descrito en la ec.(2.2), poniéndolo en función de los estados ($\mathbf{q}$ y $\dot{\mathbf{q}}$) y la **matriz de inercia** del sistema ($\mathbf{M}(\cdot) \in \mathbb{R}^{n\times n}$).

$$\mathcal{L} := \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - E_{P_T}(\mathbf{q}) \tag{2.11}$$

Si además definimos un **vector de fuerzas potenciales** $\mathbf{G}(\mathbf{q})$, dado matemáticamente por:

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} \frac{\partial E_{P_T}(\mathbf{q})}{\partial q_1} \\ \frac{\partial E_{P_T}(\mathbf{q})}{\partial q_2} \\ \vdots \\ \frac{\partial E_{P_T}(\mathbf{q})}{\partial q_n} \end{bmatrix}, \tag{2.12}$$

y una matriz que contenga los términos centrífugos y de **Coriolis**, denotada por $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$, y descrita en dos funciones.

$$\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{C_1}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{C_2}(\mathbf{q},\dot{\mathbf{q}}) \tag{2.13}$$

Las funciones matriciales $\mathbf{C_1}(\cdot)$ y $\mathbf{C_2}(\cdot)$ están dadas por:

$$\mathbf{C_1}(\mathbf{q},\dot{\mathbf{q}}) := \tfrac{d}{dt}\mathbf{M}(\mathbf{q}) = \textstyle\sum_{i=1}^{n} \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_i}\dot{q}_i$$

$$\mathbf{C_2}(\mathbf{q},\dot{\mathbf{q}}) := -\tfrac{1}{2}\begin{bmatrix} \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_1}\dot{\mathbf{q}} \\ \vdots \\ \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_n}\dot{\mathbf{q}} \end{bmatrix}$$

Considerando la **matriz de inercia** $\mathbf{M}(\cdot)$ de la ec.(2.11), y las **matrices de gravedad** $\mathbf{G}(\cdot)$ y **Coriolis** $\mathbf{C}(\cdot)$ descritas en las ecuaciones 2.12 y 2.13, respectivamente. Podemos expresar al sistema de la ec.(2.10) en la **forma del manipulador** [6], [Sætre, 2022].

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{B}_u\mathbf{u} \tag{2.14}$$

Los sistemas mecánicos descritos por la ec.(2.14), pueden caracterizarse en función de su grado de actuación. El sistema es *subactuado* si $m < n$, y su grado de subactuación es $n - m$, por ejemplo si $m = 2$ y $n = 3$, el grado de subactuación es 1, lo cual quiere decir que uno de los estados no puede ser alterado ni directamente, ni indirectamente por algún otro estado que sí es actuado. El

[6] A la ecuación 2.14 se le llama **forma del manipulador** ya que este tipo de modelos matemáticos son bastante usados para describir la dinámica de robots manipuladores de n grados de libertad.

sistema es *completamente actuado* si $m = n$, es decir, todos los estados pueden ser alterados directa o indirectamente. Finalmente, el sistema es *sobreactuado* si $m > n$, lo cual significa que es posible tener más alteración en los estados de la que es forsozamente necesaria.

Estos sistemas también cumplen con ciertas propiedades importantes:

1. El eigenvalor más pequeño (λ_m) y el eigenvalor más grande (λ_M) de $\mathbf{M}(\mathbf{q})$, con $\mathbf{q} \in \mathbb{R}^n$, representan su cota inferior y su cota superior, i.e. $\lambda_m \mathbf{I}_n \leq \mathbf{M}(\mathbf{q}) \leq \lambda_M \mathbf{I}_n$.
2. Para todo $\mathbf{q}, \mathbf{z}, \mathbf{y} \in \mathbb{R}^n$, $a \in \mathbb{R}$ y $k_c > 0$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ satisface las siguientes propiedades: 1) $\mathbf{C}(\mathbf{q}, a\mathbf{z}) = a\mathbf{C}(\mathbf{q}, \mathbf{z})$, 2) $\mathbf{C}(\mathbf{q}, \mathbf{z})\mathbf{y} = \mathbf{C}(\mathbf{q}, \mathbf{y})\mathbf{z}$ y 3) $\|\mathbf{C}(\mathbf{q}, \mathbf{z})\| \leq k_c \|\mathbf{z}\|$.
3. Existe un escalar g_M que representa la cota inferior y superior de $\mathbf{G}(\mathbf{q})$, tal que $\|\mathbf{G}(\mathbf{q})\| \leq g_M$ para todo $\mathbf{q} \in \mathbb{R}^n$

Las propiedades anteriores denotan que todas las matrices empleadas en la definición del sistema dado por ec.(2.14), están acotadas inferior y superiormente, lo que implica que los valores dentro de las matrices no pueden ser indeterminados o infinitos; esto es un aspecto que podría considerarse natural en los sistemas mecánicos reales, pues en su condición inherente, sus propiedades, estados y energia no tienden a ser infinitas.

2.2. Sistemas de Control

Un sistema de control es un conjunto de procesos y subsistemas que, a partir de una entrada específica, nos permiten obtener una salida y un comportamiento deseado, [Nise, 2015].

Según [Hendricks*, et al.*, 2008], para el correcto funcionamiento de un sistema de control se requieren los siguientes elementos generales (Figura 2.4):

1. **Planta u objeto a controlar**: Es el equipo de interés, y su naturaleza puede ser mecánica, eléctrica, hidráulica, térmica, biológica, etc.
2. **Actuadores**: Son los dispositivos que permiten inyectar la energía proveniente del controlador a la entrada de la planta (transductores de entrada).
3. **Sensores**: Son los dispositivos que nos permiten medir salidas o estados de la planta (transductores de salida).
4. **Controlador**: Es un sistema dinámico que genera una salida (entregada a los actuadores) en función de las mediciones dadas por los sensores, con el fin de lograr el objetivo de control.
5. **Equipo de Computo del Controlador**: Es el sistema computacional que procesa los datos de entrada y calcula la señal de control (se encuentra dentro del controlador en la figura 2.4).

Además de los elementos anteriormente descritos, en la figura 2.4 se introduce el término de **perturbaciones**. Una **perturbación** es una señal que afecta adversamente el valor de salida de un sistema, la señal puede ser interna (propia del sistema) o externa (generada fuera del sistema), [Ogata, 2010].

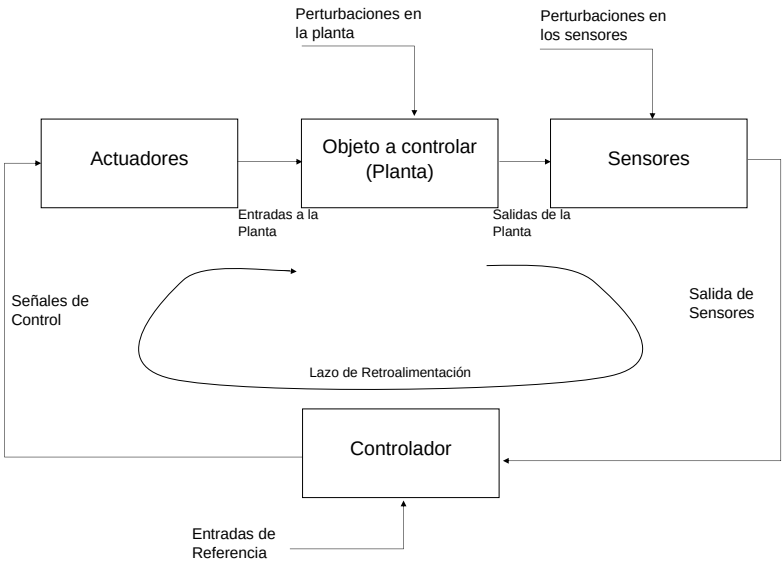


Figura 2.4 Elementos principales para un sistema de control.

Las perturbaciones no solo son generadas por malas mediciones, o comportamientos inesperados del sistema, si no que también son muy comunes en el modelado de los sistemas físicos, ya que, al generar modelos que describen el comportamiento dinámico de un sistema, es imposible tomar en cuenta todas las fuerzas que actuan sobre él, como es el caso de las fuerzas de fricción; sin embargo, es posible modelar de forma general a las perturbaciones. Para un sistema descrito por la ec.(2.7):

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t).$$

O para un LTIs:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) + \mathbf{B}_v\mathbf{v}(t).$$

Con $\mathbf{v}(\mathbf{t}) \in \mathbb{R}^p$ como el vector de perturbaciones y $\mathbf{B}_v \in \mathbb{R}^{n\times p}$.

Debido a lo anterior, las perturbaciones generalmente no pueden ser eliminadas matemáticamente en el diseño del controlador, por lo que únicamente se pueden reducir o minimizar sus efectos. Así el diseño se centra en, no solo asegurar la tarea de control, si no también en suprimir las perturbaciones del sistema, lo cual se logra con la **retroalimentación**, [Hendricks*, et al.*, 2008].

2.2.1. Control Realimentado

Para un sistema de control son posibles dos configuraciones: control en *lazo abierto* y en *lazo cerrado*, la diferencia radica en si la salida o los estados del sistema son medidos o no, [Nise, 2015].

- **Control en lazo abierto (Open-Loop Control)**. Es el sistema en el que la salida no tiene efecto en la entrada de control, es decir, no existe retroalimentación entre la salida y la entrada del sistema (Figura 2.5).

En este proceso no es posible medir un error entre la señal conseguida en la salida de la planta y la referencia. Al no contar con retroalimentación, este tipo de control no puede actuar contra perturbaciones ni internas ni externas, y no se puede asegurar que la tarea de control se lleve a cabo correctamente.

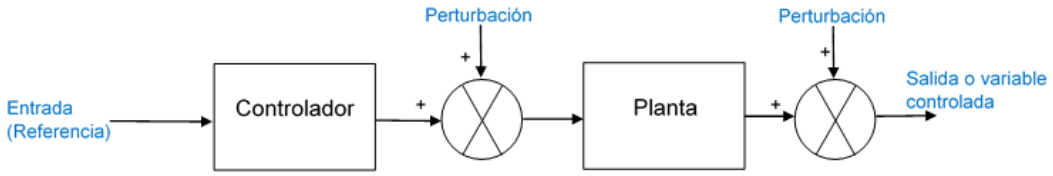


Figura 2.5 Esquema del control en lazo abierto.

- **Control en lazo cerrado (Closed-Loop Control)**. Con el fin de minimizar las perturbaciones y poder lograr la tarea de control de manera efectiva, se utiliza un término de **retroalimentación**. La **retroalimentación** se consigue al poder sensar las salidas (o estados de la **planta**), y compararla con la **señal de referencia**, para obtener una **señal de error** que finalmente ingresa al controlador.

En la figura 2.6, se puede observar que en el lazo de realimentación, la señal sensada se encuentra con signo negativo en el *sumador*, esto quiere decir que la retroalimentación es negativa, y es así debido a que este tipo de realimentación permite obtener la diferencia entre la señal medida y la deseada, permitiendo reducirla y eliminar el error conforme pasa el tiempo.

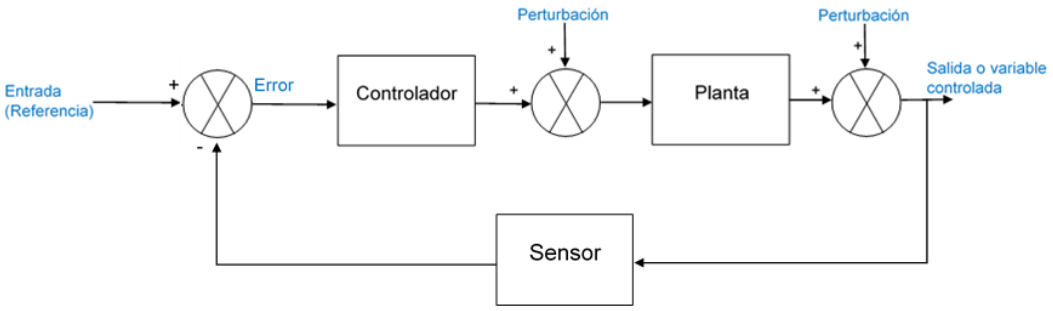


Figura 2.6 Esquema del control en lazo cerrado.

Con todo lo mencionado, podemos darnos cuenta de que la **retroalimentación** es el principal rasgo de la teoría de control, [Hendricks, *et al.*, 2008].

2.2.2. Propósitos de Control

Generalmente existen dos propóstios básicos en la teoría de control:

1. **Regulación**.
2. **Seguimiento (Tracking)**.

La **regulación** es probablemente el propósito más común del control, además de ser la noción más básica del mismo. La misión es llevar la salida de la planta a una configuración (***Set Point***) específica dentro del espacio de estados. Usualmente esta configuración es un valor dado en alguna de las salidas, o una combinación lineal de ellas.

Por otra parte, el propósito de **seguimiento** pretende que el vector de estados realice una trayectoria específica en el tiempo, esto se logra variando la entrada de referencia.

Aunque estos dos propósitos son los más comunes y estudiados por el área de control, en esta tesis introduciremos un modo extra de operación, llamado **estabilización orbital**, el cual es muy similar al control por seguimiento, pero en lugar de variar la referencia en función del tiempo, únicamente nos interesa asegurar la convergencia con una **órbita deseada**, independientemente del tiempo en el que la alcance. Este enfoque es muy útil en sistemas altamente no lineales y subactuados, [Sætre, 2022], y será estudiado más a profundidad en la sección 2.3.3.

2.2.3. Controlabilidad y Observabilidad

Antes de analizar detenidamente los sistemas de control aplicados, es de vital importancia definir las propiedades de ***alcanzabilidad***, ***controlabilidad***, ***observabilidad*** y ***constructibilidad***.

Según [Antsaklis y Michel, 2007] la alcanzabilidad se puede definir como:

Definición 2.2.1 (Alcanzabilidad) *Un sistema es **alcanzable** si existe una entrada que lo permita llevar de su estado inicial x_0 a un estado final x_1 en un intervalo de tiempo finito.*

Desde el punto de vista de [Ogata, 2010] la controlabilidad se define como:

Definición 2.2.2 (Controlabilidad) *Un sistema es **controlable** en el tiempo t_0, si existe una entrada que permita llevar a sus variables de estado, de un estado inicial $x(t_0)$ a cualquier otro estado final deseado en un intervalo de tiempo finito.*

La entrada debe ser acotada, puesto que las entradas infinitas no son útiles en su implementación, pues no es posible generar energía infinita por el principio de conservación de la energía.

Las definiciones de ***observabilidad*** y ***constructibilidad*** según [Antsaklis y Michel, 2007] son:

Definición 2.2.3 (Observabilidad) *Es la habilidad de reconstruir un estado presente $x(t_0)$ a partir del conocimiento de las entradas $u(t)$ y salidas $y(t)$ presentes y futuras del sistema, $t \geq t_0$.*

Definición 2.2.4 (Constructibilidad) *Es la habilidad de reconstruir un estado presente $x(t_0)$ a partir del conocimiento de las entradas $u(t)$ y salidas $y(t)$ presentes y pasadas del sistema, $t \leq t_0$.*

La **observabilidad** es muy útil para sistemas de los que no es posible medir (sensar) algún estado necesario para realizar una acción de control.

Los conceptos estudiados anteriormente, son complementarios y duales (Figura 2.7), es decir, analizando las definiciones podemos encontrar que para **sistemas continuos** la **alcanzabilidad** implica **controlabilidad**, así como la **observabilidad** implica **constructibilidad**.

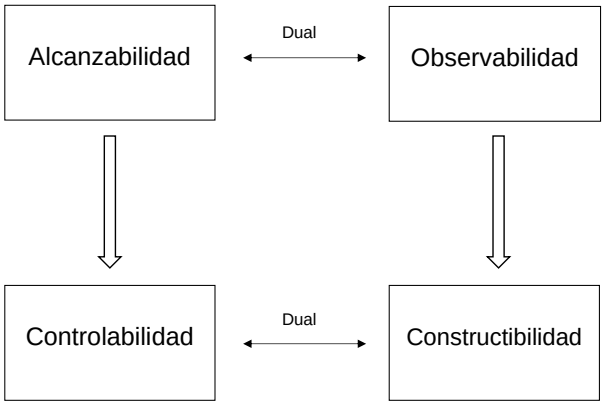


Figura 2.7 Alcanzabilidad/Controlabilidad y su relación con Observabilidad/Constructibilidad.

2.2.4. Control de sistemas lineales

Una vez entendiendo qué es la **controlabilidad** y la **observabilidad**, podemos estudiar de mejor manera el diseño de controladores. Empezaremos analizando el caso más sencillo y estudiado en los cursos de control, el diseño de controladores para sistemas dinámicos lineales, que si recordamos los temas descritos en la sección 2.1.5, tiene a su vez 2 vertientes: los sistemas lineales invariantes en el tiempo (LTIs) y los sistemas lineales variantes en el tiempo (LTVs).

2.2.4.1. Control de sistemas lineales invariantes en el tiempo - LTIs -

Partiendo de un sistema sin **prealimentación** (*feed forward*), descrito por:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t). \end{aligned} \tag{2.15}$$

Se puede comprobar **controlabilidad**, utilizando la matriz de dinámica ($\mathbf{A}$) y la matriz de entradas ($\mathbf{B}$), ya que son las matrices que contienen la información de la evolución dinámica del sistema. La expresión que nos permite comprobar controlabilidad está dada por:

$$\mathcal{C} := \left[\mathbf{B}, \mathbf{AB}, \mathbf{A^2B}, \ldots, \mathbf{A^{n-1}B}\right] \in \mathbb{R}^{n \times nm} \tag{2.16}$$

Según la ec.(2.16), el par de matrices ($\mathbf{A}$, $\mathbf{B}$) es controlable si y solo si la matriz de controlabilidad $\mathcal{C}$ es de rango completo[7], es decir, $\rho(\mathcal{C}) = n$, [Antsaklis y Michel, 2007].

También se puede comprobar **observabilidad** mediante la expresión:

[7]El rango ($\rho(\cdot)$) de una matriz es el número de columnas o renglones linealmente independientes, [Chen, 1999]. Si se tiene una matriz $M \in \mathbb{R}^{r \times c}$, se dice que M es de rango completo si $\rho(M) = min(r, c)$.

$$\mathbb{O} := \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix} \in \mathbb{R}^{pn \times n} \tag{2.17}$$

El par de matrices ($\mathbf{A}$, $\mathbf{C}$) es observable si y solo si la matriz de observabilidad $\mathcal{O}$ es de rango completo, es decir, $\rho(\mathcal{O}) = n$.

Con la controlabilidad comprobada, es posible diseñar una ley de control. En esta tesis se utilizarán **controladores basados en el modelo matemático**, pues se considera posible que la acción de control dependa de los estados medidos. De esta manera se puede definir una entrada de control general.

$$u(t) = -k_1 x_1 - k_2 x_2 - \ldots - k_n x_n + \mathbf{r(t)} \tag{2.18}$$

Dentro de la ec.(2.18), se encuentran términos k_i conocidos como **ganancias**, y un vector de señales de referencia $\mathbf{r}(t)$ que tiene la misma dimensión (m) que el vector de entradas $\mathbf{u}(t)$. Se puede definir una **matriz de ganancias** $\mathbf{K} \in \mathbb{R}^{n \times m}$, y entonces de manera general la entrada de control será descrita como:

$$u(t) = -\mathbf{K}\mathbf{x}(t) + \mathbf{r}(t). \tag{2.19}$$

A esta acción de control se le conoce como un sistema de **realimentación completa de estados** (figura 2.8), [Hendricks, *et al.*, 2008]. Hay que recordar que la acción de retroalimentación que nos conviene para el control es la **retroalimentación negativa** (ver sección 2.2.1), es por ello que en la ec.(2.19), la matriz de ganancias tiene signo negativo.

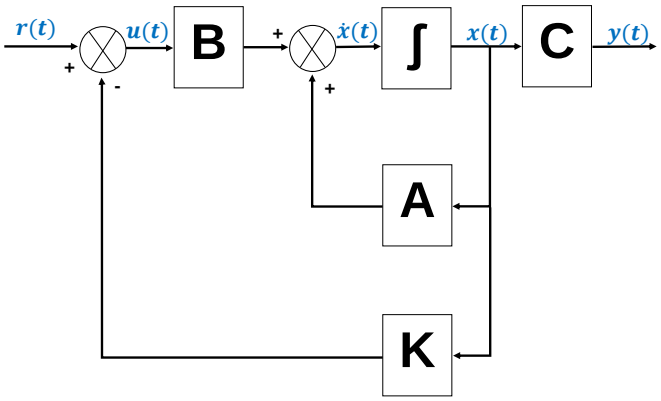


Figura 2.8 Esquema de la realimentación completa de estados.

Si sustituimos la ec.(2.19) en la ec.(2.15), encontramos el modelo del sistema realimentado.

$$\begin{aligned}\dot{\mathbf{x}}(t) &= (\mathbf{A} - \mathbf{BK})\mathbf{x}(t) + \mathbf{Br}(t),\\ \mathbf{y}(t) &= \mathbf{Cx}(t).\end{aligned} \tag{2.20}$$

Ahora como se muestra en la ec.(2.20), el término que multiplica a la dinámica natural (sin entrada) del sistema es la matriz $\mathbf{A} - \mathbf{BK}$, por lo tanto, al recapitular el tema de polos y ceros (sección 2.1.4), también se pueden obtener los eigenvalores (polos) del sistema realimentado, mediante la expresión:

$$det(s\mathbf{I} - \mathbf{A} + \mathbf{BK}) = 0 \tag{2.21}$$

El modelo del sistema realimentado permite modificar únicamente a la matriz $\mathbf{K}$, esto indica que como la ec.(2.21) depende de esta matriz, es posible modificar los polos del sistema y así garantizar un desempeño adecuado del sistema en lazo cerrado. Esta técnica se conoce como **asignación de polos**.

2.2.4.2. Control de sistemas lineales variantes en el tiempo - LTVs -

De la deducción hecha en la sección 2.2.4.1 es natural preguntarse dos cosas: ¿es posible extender los conceptos para un sistema lineal variante en el tiempo?, y si es así, ¿qué polos son los mejores por elegir? Ambas inquietudes pueden ser resueltas por un enfoque llamado **control óptimo**.

En los problemas de control toda acción realizada implica un "costo" a pagar; con el fin de que este costo no sea excesivo, es de vital interés encontrar la mejor acción de control (el mejor posicionamiento de polos). De forma matemática, lo anterior se consigue definiendo una **función de costo** que debe ser optimizada en función del espacio de estados, de modo que también se minimice la energía ejercida por el controlador hacia el sistema. Aunque se puede analizar al control óptimo de manera general para sistemas no lineales como el de la ec.(2.7), acotaremos el análisis al caso de LTVs, mediante el algoritmo conocido como **regulador lineal cuadrático** (LQR, por sus siglas en inglés).

Linear Quadratic Regulator (LQR) . La razón del nombre de este algoritmo se debe a dos principios: se aplica a modelos lineales (**linear**) y la **función de costo** es cuadrática (**quadratic**).

El análisis comienza a partir de un LTVs descrito por la ec.(2.8), la meta es encontrar una entrada $\mathbf{u}(t)$ que permita optimizar la función de costo cuadrática ($\mathbf{J}$) dada por [Hendricks, *et al.*, 2008]:

$$\mathbf{J}(\mathbf{x}, \mathbf{u}) = \int_0^\infty (\mathbf{x}^T(t)\mathbf{Q}(t)\mathbf{x}(t) + \mathbf{u}^T(t)\mathbf{R}(t)\mathbf{u}(t))dt. \tag{2.22}$$

La ec.(2.22) describe a una función de costo matricial y cuadrática, de ahí su definición con el vector de estados $\mathbf{x}(t)$ y el vector de entradas $\mathbf{u}(t)$. La matriz $\mathbf{Q}(t) \in \mathbb{R}^{n\times n}$ es conocida como **matriz de peso del error** y $\mathbf{R}(t) \in \mathbb{R}^{n\times m}$ es la **matriz de costo de control**, en conjunto son llamadas **matrices de ponderación** y tienen la forma:

$$\mathbf{Q}(t) = \begin{bmatrix} q_1(t) & q_{12}(t) & \cdots & q_{1n}(t) \\ q_{21}(t) & q_2(t) & \cdots & q_{2n}(t) \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1}(t) & q_{n2} & \cdots & q_n \end{bmatrix},$$

$$\mathbf{R}(t) = \begin{bmatrix} r_1(t) & r_{12}(t) & \cdots & r_{1n}(t) \\ r_{21}(t) & r_2(t) & \cdots & r_{2n}(t) \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1}(t) & r_{n2} & \cdots & r_n \end{bmatrix}.$$

Los términos $q_{ij}(t)$ y $r_{ij}(t)$ denotan que estas matices nos permiten saber la influencia entre las componentes individuales de los vectores de estado y de entrada. Además deben cumplir con ser positivas definidas (en el caso de $\mathbf{R}$) o positivas semidefinidas[8] (en el caso de $\mathbf{Q}$), i.e

$$Q = Q^T \geq 0,$$
$$R = R^T > 0.$$

Para entender mejor cómo actúan las **matrices de ponderación**, ejemplifiquemos los casos, si $\mathbf{Q}(t)$ tiene su primer término (q_1, q_{1n}) mayor al segundo (q_2, q_{2n}), los estados del sistema tienden a responder rápido, a expensas de aumentar la señal de control; en el caso contrario, la señal de control disminuye, lo cual ocasiona que los estados del sistema respondan más lento. Es por esto, que la elección de las matrices de ponderación nos permiten sintonizar la velocidad de la respuesta del sistema, tanto como acotar la señal de control, [Hendricks*, et al.*, 2008].

La función de costo (ecuación 2.22), actúa únicamente sobre el sistema descrito con el espacio de estados de la ec.(2.8), por ello, el espacio de estados funge como una **restricción dinámica (*Dynamic Constraint*)**. Debido a lo anterior, es necesaria una herramienta matemática conocida como **multiplicador de Lagrange** $(\lambda(t))$, que ayuda a encontrar la evolución energética del sistema bajo estas restricciones. El Hamiltoniano[9] del sistema está dado por

$$\mathcal{H} = \frac{1}{2}\mathbf{x}^T(t)\mathbf{Q}(t)\mathbf{x}(t) + \frac{1}{2}\mathbf{u}^T(t)\mathbf{R}(t)\mathbf{u}(t) + \lambda^T(t)\left(\mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)\right) \tag{2.23}$$

De la ecuación anterior, el único término que se desconoce es el multiplicador de Lagrange $(\lambda(t))$. Al tratarse de un sistema variante en el tiempo que debe proveer el comportamiento óptimo del sistema, la ec.(2.23) debe satisfacer simultáneamente a la **ecuación diferencial parcial de Hamilton-Jacobi-Bellman (HJB)**, [Poznyak, 2008], y a la **ecuación de co-estado**, descritas respectivamente por la ec.(2.24) y ec.(2.25).

$$\forall x \ \min_u \left[\frac{1}{2}\mathbf{x}^T(t)\mathbf{Q}(t)\mathbf{x}(t) + \frac{1}{2}\mathbf{u}^T(t)\mathbf{R}(t)\mathbf{u}(t) + \frac{\partial \mathbf{J}^*}{\partial \mathbf{x}}\left(\mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)\right)\right] = 0 \tag{2.24}$$

$$\dot{\lambda}^T(t) = -\frac{\partial \mathcal{H}}{\partial \mathbf{x}(t)} \tag{2.25}$$

[8]Una matriz positiva definida es aquella en la que todos sus eigenvalores son positivos (no incluye al 0, i.e. $eig(R) > 0$), mientras que es positiva semidefinida, si todos sus eigenvalores son no negativos (incluye al 0, i.e. $eig(Q) \geq 0$), [Antsaklis y Michel, 2007].

[9]El Hamiltoniano es la función que describe la evolución energética de algún sistema.

Donde $\lambda^T = \mathbf{J}^*$ es una función auxiliar de costo cuadrática y convexa descrita por $\mathbf{J}^* = \frac{1}{2}\mathbf{x}^T(t)\mathbf{P}(t)\mathbf{x}(t)$, que al ser derivada con respecto a $\mathbf{x}(t)$ da como resultado:

$$\frac{\partial \mathbf{J}^*}{\partial \mathbf{x}(t)} = \mathbf{x}^T(t)\mathbf{P}(\mathbf{t}). \tag{2.26}$$

Sustituyendo la ec.(2.26) en la ec.(2.24), se simplifica el Hamiltoniano:

$$\forall x \ \min_u \left[\frac{1}{2}\mathbf{x}^T(t)\mathbf{Q}(t)\mathbf{x}(t) + \frac{1}{2}\mathbf{u}^T(t)\mathbf{R}(t)\mathbf{u}(t) + \mathbf{x}^T(t)\mathbf{P}(\mathbf{t})\left(\mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)\right)\right] = 0 \tag{2.27}$$

De esta manera, el problema planteado por la ecuación diferencial parcial HJB (ec.2.27), radica exclusivamente en encontrar una entrada $\mathbf{u}(t)$ que minimice al Hamiltoniano del sistema; para esto se debe utilizar el criterio de la primer derivada para encontrar el mínimo de la función $\mathcal{H}$.

$$\frac{\partial \mathcal{H}}{\partial \mathbf{u}(t)} = \mathbf{u}^T(t)\mathbf{R}(t) + \mathbf{x}^T(t)\mathbf{P}(t)\mathbf{B}(t) = 0$$

Despejando $\mathbf{u}(\mathbf{t})$, se encuentra la **entrada de control óptima**

$$\mathbf{u}(\mathbf{t}) = -\mathbf{R}^{-1}(t)\mathbf{B}^T(t)\mathbf{P}(t)\mathbf{x}(t). \tag{2.28}$$

Hasta este momento el único valor que sigue siendo una incógnita es la matriz $\mathbf{P}(t)$, pero al recordar que para garantizar el comportamiento óptimo del sistema, la ec.(2.25) también debe ser cumplida y como $\lambda(t) = \mathbf{P}(t)\mathbf{x}(t)$.

$$\dot{\lambda}(t) = \dot{\mathbf{P}}(t)\mathbf{x}(t) + \mathbf{P}(t)\dot{\mathbf{x}}(t)$$

$$\dot{\lambda}(t) = -\left(\frac{\partial \mathcal{H}}{\partial \mathbf{x}(t)}\right)^T = -\mathbf{Q}(t)\mathbf{x}(t) - \mathbf{A}^T\lambda(t) = -\mathbf{Q}(t)\mathbf{x}(t) - \mathbf{A}^T(t)\mathbf{P}(t)\mathbf{x}(t)$$

Además sustituyendo la ec.(2.28) en la ec.(2.8), se obtiene

$$\dot{\mathbf{P}}(t)\mathbf{x}(t) + \mathbf{P}(t)[\mathbf{A}(t)\mathbf{x}(t) - \mathbf{B}(t)\mathbf{R}^{-1}(t)\mathbf{B}^T(t)\mathbf{P}(t)\mathbf{x}(t)] = -\mathbf{Q}(t)\mathbf{x}(t) - \mathbf{A}^T(t)\mathbf{P}(t)\mathbf{x}(t).$$

Es sencillo notar que en la ecuación anterior todos los términos multiplican al vector de estados $\mathbf{x}(t)$ por la izquierda, factorizando este término y reacomodando la ecuación, se tiene la siguiente forma

$$\left(\dot{\mathbf{P}}(t) + \mathbf{P}(t)[\mathbf{A}(t) - \mathbf{B}(t)\mathbf{R}^{-1}(t)\mathbf{B}^T(t)\mathbf{P}(t)] + \mathbf{Q}(t) + \mathbf{A}^T(t)\mathbf{P}(t)\right)\mathbf{x}(t) = 0.$$

El vector de estados no puede ser nulo, i.e. $\mathbf{x}(t) \neq 0$, es por ello que el término dentro del paréntesis debe ser igual a 0, lo cual resulta en una ecuación diferencial con la matriz $\mathbf{P}(t)$ como su solución

$$\dot{\mathbf{P}}(t) + \mathbf{A}^T(t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{A}(t) + \mathbf{Q}(t) = \mathbf{P}(t)\mathbf{B}(t)\mathbf{R}^{-1}(t)\mathbf{B}^T(t)\mathbf{P}(t). \tag{2.29}$$

La ec.(2.29) es conocida como la **ecuación diferencial matricial de Riccati** y permite encontrar la matriz $\mathbf{P}(t)$ que asegura el comportamiento óptimo de un LTVs.

Existe una versión algebraica de la ecuación de Ricatti (ecuación 2.30), la cual se obtiene mediante una deducción similar a la realizada en esta sección, y es muy útil para encontrar la matriz $\mathbf{P}$ que asegura el comportamiento óptimo de un LTIs.

$$\mathbf{Q} + \mathbf{A}^T\mathbf{P} + \mathbf{PA} - \mathbf{PBR}^{-1}\mathbf{B}^T\mathbf{P} = 0 \tag{2.30}$$

2.2.5. Control de sistemas no lineales

Al realizar el modelado de un sistema dinámico notaremos que, por lo general, las ecuaciones resultantes contienen elementos no lineales como potencias de algún estado (x_1^2, x_2^3), multiplicaciones entre estados (x_2x_4) o términos trigonométricos ($sen(x_1), cos(x_3)$), lo que demuestra que la no linealidad es inherente en los sistemas físicos.

A pesar de la naturaleza no lineal de los sistemas, es posible aproximar su comportamiento a un contexto lineal en el que se pueden aplicar los métodos descritos en la sección 2.2.4. Sin embargo, los controladores para sistemas lineales tienen dos limitaciones:

- Los sistemas lineales tienen un **correcto funcionamiento únicamente en una región limitada**. Esto significa que si esta región es superada, el modelo lineal es incapaz de predecir el comportamiento del sistema, lo que puede causar inestabilidad, y por lo tanto, un mal desempeño.
- Existen sistemas que son **altamente no lineales**, y no es posible representarlos de manera correcta por un modelo lineal.

Estas dos razones motivan a un estudio más general de los sistemas de control, que puedan compensar aquellos factores que el control lineal es incapaz; a este marco de estudio le corresponden las técnicas de **control no lineal**.

Según [Slotine y Li, 1991] los principales sucesos que afectan a los sistemas no lineales son:

A **Zonas de estabilidad**[10]: En los sistemas no lineales pueden coexistir regiones estables y regiones inestables, que da cabida a hablar de **estabilidad local** (Hay dependencia de la condición inicial - figura 2.9); caso contrario ocurre en los sistemas lineales donde se tiene estabilidad global o inestabilidad global, pero no ambos a la vez.

B **Oscilaciones**: Los sistemas no lineales generan oscilaciones internas sin importar sus parámetros.

C **Sistemas caóticos**: Debido a que los sistemas no lineales no cumplen con la propiedad de **superposición** (propia de los sistemas lineales), es posible que pequeños cambios en las condiciones iniciales provoquen un comportamiento completamente aleatorio en la salida, el cual incluso se puede volver impredecible; a este fenómeno se le conoce como **caos**.

Aunque no existe un método general para diseñar controladores de sistemas no lineales, existen una buena cantidad de técnicas que pueden ser aplicadas según el caso de estudio. Para esta tesis nos centraremos en la técnica de **control por linealización de coordenadas transversales**.

[10]El tema de estabilidad es estudiado en la sección 2.3

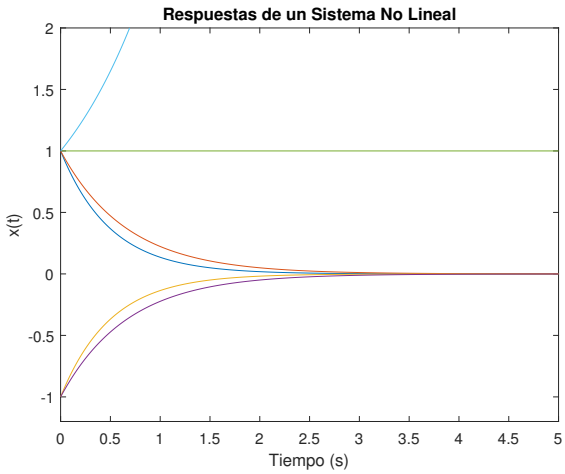


Figura 2.9 Respuestas de un sistema no lineal.

2.2.5.1. Control por linealización de coordenadas transversales

El método de **control por linealización de coordenadas transversales** nace del estudio de la estabilización órbital (explicada a fondo en la sección 2.3.3), la cual requiere encontrar una entrada **u** que pueda generar y estabilizar una órbita (Problema 2.3.1). Cabe aclarar que este algoritmo permite únicamente controlar órbitas de tipo periódicas, i.e. $\mathbf{x}^*(t+T) = \mathbf{x}^*(t)$, con periodo $T > 0$.

La ley de control propuesta fue presentada por [Shiriaev, *et al.*, 2005] para sistemas con un grado de libertad no actuado, y se extendió a más grados de libertad no actuados en [Shiriaev, *et al.*, 2010]. Para poder trabajar con este tipo de controladores es necesaria la noción de **operadores de proyección** y **coordenadas transversales**.

Operador de proyección y coordenadas transversales

En la tesis doctoral de [Sætre, 2022] se define a un operador de proyección de la siguiente manera:

Definición 2.2.5 (Operador de proyección) *Se tiene una órbita periódica $\mathcal{O}$ y una perametrización $x_s : S \to \mathcal{O}$. La función de mapeo $\mathcal{C}^1$, $p : \mathcal{N}_\mathcal{O} \to S$, que está definida dentro de una vecindad $\mathcal{N}_\mathcal{O} \subset \mathbb{R}^n$ de $\mathcal{O}$, es un* ***operador de proyección*** *respecto a $\mathcal{O}$ si $s = p(x_s(s))$ para toda $s \in S$.*

El operador de proyección de la definición 2.2.5, permite encontrar una proyección de los estados del sistema en la órbita, i.e. $\mathcal{N}_\mathcal{O} \to \mathcal{O}$, como una composición

$$x_p := (x_s \circ p)(x).$$

Y por lo tanto, también se puede definir una función de error, la cual si se aplica a órbitas periódicas es conocida como **coordenada transversal**

$$e_\perp := x - x_p(x). \tag{2.31}$$

Naturalmente cuando la coordenada transversal es 0, significa que el sistema se encuentra en la órbita forzada. De esta aseveración se puede generar una definición formal para el vector de coordenadas transversales, [Sætre, 2022]:

Definición 2.2.6 (Coordenadas Transversales) *Una función vectorial* $\mathcal{C}^2$, $\mathbf{z}_\perp : \mathbb{R}^n \to \mathbb{R}^{n-1}$ *es un* ***vector de coordenadas transversales*** *para una órbita periódica* $\mathcal{O}$ *si* $\|\mathbf{z}_\perp(y)\| \equiv 0$ *y es de rango* $n-1$ *para toda* $y \in \mathcal{O}$.

La pregunta forzosa para estos sistemas es ¿cómo encontrar operadores de proyección que puedan obtener las coordenadas transversales? Aunque existen diversas maneras de hacerlo no es una tarea sencilla, sin embargo, tampoco es una tarea forsozamente necesaria, pues la teoría indica que para lograr estabilización orbital tener información sobre el operador de proyección puede ser prescindible, siempre y cuando, conozcamos un vector de coordenadas transversales.

Dentro del algoritmo propuesto por [Shiriaev*, et al.*, 2005] se describe una manera de obtener el vector de coordenadas transversales a partir del uso de un ente matemático conocido como **restrición virtual**[11] (*Virtual Constraint*). Pero antes de entender como se obtienen, se tiene que comprender cómo funciona este procedimiento.

Para poder estabilizar una órbita periódica es necesario primero generar esta órbita y luego garantizar su estabilidad. Precisamente el método mencionado anteriormente se subdivide en dos partes: el **generador de movimientos** y la **estabilización orbital**.

Generador de movimientos

Partimos de un sistema descrito en la forma del manipulador:

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + G(\mathbf{q}) = B(\mathbf{q})\mathbf{u} \tag{2.32}$$

Para inducir rotaciones continuas al sistema descrito por la ec.(2.32), es necesario definir funciones que relacionen las variables de estado $\mathbf{q}(t)$ con las trayectorias deseadas $\phi(\theta^*)$ para generar la órbita.

$$q_1 = \phi_1(\theta^*), \cdots, q_n = \phi_n(\theta^*),\ \theta^* \in [\Theta_b, \Theta_c] \tag{2.33}$$

Donde θ^* es un parámetro de la órbita que incluso puede ser alguno de los estados q_i. Las funciones dadas por la ec.(2.33) son justamente las **restricciones virtuales**, [Shiriaev*, et al.*, 2010]. Todas las restricciones y sus derivadas temporales se pueden agrupar en vectores:

$$\begin{aligned} \mathbf{\Phi}(\theta^*) &= [\phi_1(\theta^*), \cdots, \phi_n(\theta^*)]^T, \\ \dot{\mathbf{\Phi}}(\theta^*) &= [\dot{\phi_1}(\theta^*), \cdots, \dot{\phi_n}(\theta^*)]^T, \\ \ddot{\mathbf{\Phi}}(\theta^*) &= [\ddot{\phi_1}(\theta^*), \cdots, \ddot{\phi_n}(\theta^*)]^T. \end{aligned} \tag{2.34}$$

Sustituyendo los vectores de la ec.(2.34) en el sistema de la ec.(2.32), se obtiene:

$$M(\mathbf{\Phi}(\theta^*))[\ddot{\mathbf{\Phi}}(\theta^*)\dot{\theta}^* + \dot{\mathbf{\Phi}}(\theta^*)\ddot{\theta}^*] + C(\mathbf{\Phi}(\theta^*), \dot{\mathbf{\Phi}}(\theta^*)\dot{\theta}^*)(\dot{\mathbf{\Phi}}(\theta^*)\dot{\theta}^*) + G(\mathbf{\Phi}(\theta^*)) = B(\mathbf{\Phi}(\theta^*))\mathbf{u}^*.$$

[11] Una restricción virtual es una limitación matemática que nosotros le ponemos al sistema, con el fin de acotar o ajustar su movimiento.

Considerando que se tienen m entradas de control, podemos tener una función matricial $\mathbf{B}^{\perp}(\mathbf{q}) \in \mathbb{R}^{(n-m)\times n}$. Premultiplicando la ecuación anterior por $\mathbf{B}^{\perp}(\mathbf{q})$, es posible deshacerse de la entrada de control (debido a que $\mathbf{B}^{\perp}(\mathbf{q})\mathbf{B}(\mathbf{q}) = 0$), [Shiriaev, *et al.*, 2010]

$$\mathbf{B}^{\perp}(\mathbf{\Phi}(\theta^*))\left\{M(\mathbf{\Phi}(\theta^*))[\ddot{\mathbf{\Phi}}(\theta^*)\dot{\theta}^* + \dot{\mathbf{\Phi}}(\theta^*)\ddot{\theta}^*] + C(\mathbf{\Phi}(\theta)^*, \dot{\mathbf{\Phi}}(\theta^*)\dot{\theta}^*)(\dot{\mathbf{\Phi}}(\theta^*)\dot{\theta}^*) + G(\mathbf{\Phi}(\theta^*))\right\}$$
$$= \mathbf{B}^{\perp}(\mathbf{\Phi}(\theta^*))\mathbf{B}(\mathbf{\Phi}(\theta^*))\mathbf{u}^* = 0.$$

Si se agrupan términos se obtiene:

$$\begin{aligned}
\begin{bmatrix} \alpha_1(\theta^*) \\ \vdots \\ \alpha_{n-m}(\theta^*) \end{bmatrix} &= \mathbf{B}^{\perp}(\mathbf{\Phi}(\theta^*))M(\mathbf{\Phi}(\theta^*))\dot{\mathbf{\Phi}}(\theta^*), \\
\begin{bmatrix} \beta_1(\theta^*) \\ \vdots \\ \beta_{n-m}(\theta^*) \end{bmatrix} &= \mathbf{B}^{\perp}(\mathbf{\Phi}(\theta^*))\left[M(\mathbf{\Phi}(\theta^*))\ddot{\mathbf{\Phi}}(\theta^*) + C(\mathbf{\Phi}(\theta^*), \dot{\mathbf{\Phi}}(\theta^*))(\dot{\mathbf{\Phi}}(\theta^*)\dot{\theta}^*)\right], \\
\begin{bmatrix} \gamma_1(\theta^*) \\ \vdots \\ \gamma_{n-m}(\theta^*) \end{bmatrix} &= \mathbf{B}^{\perp}(\mathbf{\Phi}(\theta^*))G(\mathbf{\Phi}(\theta^*)).
\end{aligned} \tag{2.35}$$

Con las relaciones dadas en la ec.(2.35), se puede encontrar que $\theta = \theta^*$ es la solución al conjunto de ecuaciones diferenciales

$$\alpha_i(\theta)\ddot{\theta} + \beta_i(\theta)\dot{\theta}^2 + \gamma_i(\theta) = 0,\ i = 1, \cdots, n-m. \tag{2.36}$$

La cantidad de ecuaciones del conjunto dado por la ec.(2.36) estará en función del número de grados de libertad no actuados. Por ejemplo, para un sistema con un grado de libertad no actuado, solo existirá una ecuación diferencial de este tipo.

De la ec.(2.36) y la ec.(2.35), [Shiriaev, *et al.*, 2005] presenta dos teoremas:

Teorema 2.2.1 *Suponiendo que la función $\alpha(\theta)$ tiene ceros aislados. Si la solución $(\theta(t), \dot{\theta}(t))$ del generador de movimientos (ec.2.36) existe y es diferenciable, con las condiciones iniciales $\theta(0) = \theta_0, \dot{\theta}(0) = \dot{\theta}_0$, entonces durante toda la solución, la función*

$$I(\theta, \dot{\theta}, \theta_0, \dot{\theta}_0) = \dot{\theta}^2 - \psi(\theta_0, \theta)\left[\dot{\theta}_0^2 - \int_{\theta_0}^{\theta} \psi(s, \theta_0)\frac{2\gamma(s)}{\alpha(s)}ds\right] \tag{2.37}$$

con:

$$\psi(\theta_0, \theta) = e^{-2\int_{\theta_0}^{\theta} \frac{\beta(\tau)}{\alpha(\tau)}d\tau}$$

se mantiene en 0.

El teorema 2.2.1 presenta a la **integral de movimiento**, la cual se puede ver como una función de error, pues siempre que el sistema se encuentre en la órbita su valor será 0. Entonces en el mismo articulo se propone un segundo teorema:

Teorema 2.2.2 *Considerando las constantes k y l, la derivada temporal de $I(\theta, \dot{\theta}, k, l)$ calculada sobre la solución $(\theta(t), \dot{\theta}(t))$, del generador de movimientos forzado*

$$\alpha(\theta)\ddot{\theta} + \beta(\theta)\dot{\theta}^2 + \gamma(\theta) = u, \tag{2.38}$$

se puede calcular como

$$\frac{d}{dt}I = \dot{\theta}\left[\frac{2}{\alpha(\theta)}u - \frac{2\beta(\theta)}{\alpha(\theta)}I\right].$$

Con la ec.(2.38), se puede encontrar la parametrización que permite obtener las restricciones virtuales deseadas; pero la forma en la que se asegure que eso suceda aún no es muy clara. Para ello se utilizará la segunda parte del método empleado.

Estabilización orbital

Aunque los conceptos y la definición formal de estabilización orbital se encuentran en la sección 2.3.3, aquí se utilizarán las coordenadas transversales para generar un controlador que permita estabilizar el movimiento periódico en la órbita.

Se puede notar que las restricciones virtuales (ec.2.34), son en realidad límites impuestos una vez que se define la órbita deseada. Por ello es posible generar las funciones de error entre los estados reales y los deseados

$$\xi_1 = q_1(t) - \phi_1(\theta), \cdots, \xi_n = q_n(t) - \phi_n(\theta). \tag{2.39}$$

Las funciones resultantes en la ec.(2.39) son llamadas **coordenadas transversales excesivas**. En el caso en que los sistemas cuenten con la variable de parametrización θ igual a un estado del sistema q_i, una de las funciones puede ser representada por la combinación lineal de las otras, por lo que las coordenadas podrían expresarse de la siguiente manera, [Shiriaev*, et al.*, 2005]:

$$\xi = \left[\xi_1, \cdots, \xi_{n-1}\right]^T \text{ y } \theta \tag{2.40}$$

Entonces, de la ec.(2.39) se obtiene la siguiente relación:

$$q_n = \phi_n(\theta) + h(\xi_1, \cdots, \xi_{n-1}, \theta) \tag{2.41}$$

donde $h(\cdot)$ es una función escalar cuyo argumento son los errores de seguimiento orbital ξ_j (con $j = 1, \ldots, n-1$).

De las ecuaciones pasadas existe una relación directa entre las variables de estado, las restricciones virtuales y las coordenadas transversales; a raíz de esto se tiene una transformación para denotar sus relaciones:

$$\begin{aligned} \dot{\mathbf{q}} &= \mathbf{L}(\theta, \xi)\begin{bmatrix}\dot{\xi}\\ \dot{\theta}\end{bmatrix}, \\ \ddot{\mathbf{q}} &= \mathbf{L}(\theta, \xi)\begin{bmatrix}\ddot{\xi}\\ \ddot{\theta}\end{bmatrix} + \mathbf{N}(\theta, \dot{\theta}, \xi, \dot{\xi}). \end{aligned} \tag{2.42}$$

Las matrices $\mathbf{L}(\theta,\xi)$ y $\mathbf{N}(\theta,\dot{\theta},\xi,\dot{\xi})$ están dadas por

$$\mathbf{L}(\theta,\xi)=\begin{bmatrix}\mathbf{1}_{(n-1)} & \mathbf{0}_{(n-1)x1}\\ \frac{\partial h}{\partial \xi} & \frac{\partial h}{\partial \theta}\end{bmatrix}+\begin{bmatrix}\mathbf{0}_{nx(n-1)} & \dot{\mathbf{\Phi}}\end{bmatrix}, \tag{2.43}$$

$$\mathbf{N}(\theta,\dot{\theta},\xi,\dot{\xi})=\begin{bmatrix}\mathbf{1}_{(n-1)} & \mathbf{0}_{nx(n-1)}\end{bmatrix}\left[\mathbf{L}^{-1}(\theta,\xi)\mathbf{M}^{-1}(\mathbf{q})\mathbf{B}(\mathbf{q})\right]. \tag{2.44}$$

Reescribiendo el sistema de la ec.(2.32) se puede encontrar que

$$\ddot{\mathbf{q}}=M^{-1}(\mathbf{q})[-C(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}-G(\mathbf{q})+\mathbf{B}(\mathbf{q})\mathbf{u}]. \tag{2.45}$$

Con la ecuación anterior, analicemos un poco qué implican física y matemáticamente los matrices previamente definidas. En la ec.(2.42) la matriz $\mathbf{L}(\theta,\xi)$ representa la transformación de la dinámica de las variables de estado $\mathbf{q}$ con las coordenadas transversales ξ y las restricciones virtuales (parametrizadas con θ). Por su parte, la matriz $\mathbf{N}(\theta,\dot{\theta},\xi,\dot{\xi})$ contiene tres términos importantes: la inversa de la matriz de inercia $\mathbf{M}^{-1}(\mathbf{q})$, la matriz de acoplamiento de entradas $\mathbf{B}(\mathbf{q})$ y la inversa de la matriz de transformación $\mathbf{L}^{-1}(\theta,\xi)$; el producto $\mathbf{M}^{-1}(\mathbf{q})\mathbf{B}(\mathbf{q})$ se puede considerar un **coeficiente de control** para los estados, ya que es el único término que multiplica a la entrada de control $\mathbf{u}$ (como se puede observar en la ec.2.45), a su vez, la matriz $\mathbf{L}^{-1}(\theta,\xi)$, está premultiplicando a este coeficiente de control, esto indica que el término N, transforma el coeficiente de control del sistema de la ec.(2.32), a las coordenadas transversales y restricciones debidas a la órbita deseada.

Si igualamos la ec.(2.45) con la ec.(2.42), se tiene:

$$\mathbf{L}(\theta,\xi)\begin{bmatrix}\ddot{\xi}\\ \ddot{\theta}\end{bmatrix}+\mathbf{N}(\theta,\dot{\theta},\xi,\dot{\xi})=M^{-1}(\mathbf{q})[-C(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}-G(\mathbf{q})+\mathbf{B}(\mathbf{q})\mathbf{u}].$$

Definiendo $\mathbf{N}(\theta,\dot{\theta},\xi,\dot{\xi})=K\in\mathbb{R}^{(n-1)\times(n-1)}$ como una matriz de ganancias, entonces se puede tener una representación de las coordenadas transversales como

$$\ddot{\xi}=\mathbf{K}(\xi,\theta)\mathbf{u}+\mathbf{R}(\xi,\dot{\xi},\theta,\dot{\theta}).$$

Donde $\mathbf{R}(\cdot)$ es una matriz que no depende de la entrada $\mathbf{u}$. De esta manera podemos encontrar una entrada de control realimentado para garantizar la estabilidad del sistema

$$\mathbf{u}=\mathbf{K}^{-1}(\xi,\theta)\left[w-\mathbf{R}(\xi,\dot{\xi},\theta,\dot{\theta})\right]. \tag{2.46}$$

La entrada de control dada por la ec.(2.46) conlleva a un lazo cerrado de la siguiente forma:

$$\ddot{\xi}=w \tag{2.47}$$

$$\alpha_i(\theta)\ddot{\theta}+\beta_i(\theta)\dot{\theta}^2+\gamma_i(\theta)=g_\xi(\theta,\dot{\theta},\ddot{\theta},\xi,\dot{\xi})\xi+g_{\dot{\xi}}(\theta,\dot{\theta},\ddot{\theta},\xi,\dot{\xi})\dot{\xi}+g_w(\theta,\dot{\theta},\xi,\dot{\xi})w \tag{2.48}$$

Donde $g_\xi(\cdot)$, $g_{\dot{\xi}}(\cdot)$ y $g_w(\cdot)$ son funciones suaves.

El sistema encontrado en la ec.(2.48), cumple con la forma del generador de movimientos forzado, pero está acotado dentro de las restricciones virtuales y las coordenadas transversales elegidas. Al

tratarse de soluciones periódicas, podemos definir la solución $\theta = \theta^*(t)$. Retomando el teorema 2.2.2, y considerando la ec.(2.38), se tiene:

$$\dot{I} = \frac{2\dot{\theta}^*(t)}{\alpha(\theta^*(t))}\left[g_\xi(t)\xi + g_{\dot{\xi}}(t)\dot{\xi} + g_w(t)w - \beta(\theta^*(t))I\right] \tag{2.49}$$

Tomando en cuenta a las siguientes funciones como periódicas[12]

$$\begin{aligned} g_\xi(t) &= g_\xi(\theta^*(t), \dot{\theta^*}(t), \ddot{\theta^*}(t), \mathbf{0}_{(n-1)x1}, \mathbf{0}_{(n-1)x1}), \\ g_{\dot{\xi}}(t) &= g_{\dot{\xi}}(\theta^*(t), \dot{\theta^*}(t), \ddot{\theta^*}(t), \mathbf{0}_{(n-1)x1}, \mathbf{0}_{(n-1)x1}), \\ g_w(t) &= g_w(\theta^*(t), \dot{\theta^*}(t), \mathbf{0}_{(n-1)x1}, \mathbf{0}_{(n-1)x1}). \end{aligned}$$

A partir de las ecuaciones (2.47) y (2.49), se eligen las coordenadas transversales "estándares" $\xi \in \mathbb{R}^{2n-1}$ para este procedimiento:

$$\xi := \begin{bmatrix} y \\ \dot{y} \\ I \end{bmatrix} \tag{2.50}$$

de las cuales si se obtiene su dinámica:

$$\dot{\xi} = \begin{bmatrix} \dot{y} \\ \ddot{y} \\ \dot{I} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{(n-1)\times(n-1)} & \mathcal{I}_{(n-1)\times(n-1)} & \mathbf{0}_{(n-1)\times 1} \\ \mathbf{0}_{(n-1)\times(n-1)} & \mathbf{0}_{(n-1)\times(n-1)} & \mathbf{0}_{(n-1)\times 1} \\ \frac{2\dot{\theta}^*(t)}{\alpha(\theta^*(t))}g_y(t) & \frac{2\dot{\theta}^*(t)}{\alpha(\theta^*(t))}g_{\dot{y}}(t) & -\frac{2\dot{\theta}^*(t)}{\alpha(\theta^*(t))}\beta(\theta^*(t)) \end{bmatrix}\xi + \begin{bmatrix} \mathbf{0}_{(n-1)\times(n-1)} \\ \mathcal{I}_{(n-1)\times(n-1)} \\ \frac{2\dot{\theta}^*(t)}{\alpha(\theta^*(t))}g_w(t) \end{bmatrix} w \tag{2.51}$$

Con $\mathcal{I} \in \mathbb{R}^{(n-1)\times(n-1)}$ como la matriz identidad.

De esta forma notamos que el sistema de la ec.(2.51) tiene la forma de un LTVs.

$$\dot{\xi} = \mathbf{A}(t)\xi + \mathbf{B}(t)w \tag{2.52}$$

Con $\mathbf{A}(t) \in \mathbb{R}^{(2n-1)\times(2n-1)}$ y $\mathbf{B}(t) \in \mathbb{R}^{(2n-1)\times(n-1)}$ periódicas, i.e. $\mathbf{A}(t) = \mathbf{A}(t+T)$ y $\mathbf{B}(t) = \mathbf{B}(t+T)$ con un periodo $T > 0$.

Dentro de la sección 2.2.4.2 se encontró que la entrada de control w está dada por

$$w(t, \xi) = -\mathbf{R}^{-1}(t)\mathbf{B}(t)\mathbf{P}(t)\xi. \tag{2.53}$$

Donde $\mathbf{P}(t)$ es la solución periódica y positiva de la ecuación diferencial de Ricatti (ec.2.29). Si se tiene $Q > 0$ y $R > 0$ se puede garantizar la estabilidad exponencial de las trayectorias del sistema (2.51) a una órbita predefinida, [Surov*, et al.*, 2015].

De forma general, podemos sintetizar el método propuesto por [Shiriaev*, et al.*, 2005] en cuatro pasos:

[12]Una función periódica $f \in \mathbb{R}^n$ es aquella que dado un periodo positivo y constante T, cumple con $f(t+T) = f(t)$, [Chen, 1999].

1. **Proponer la restricción virtual** para generar la órbita deseada en función de una variable θ.
2. Encontrar las **coordenadas transversales** que cumplan con la **integral de movimiento** (ec.2.37).
3. Diseñar la **entrada de control** (ec.2.53) para el sistema LTVs de la ec.(2.52).
4. A la entrada de control anterior añadir la **entrada de control nominal**[13], obtenida con las transformaciones que otorgan las matrices $\mathbf{L}(\theta, \xi)$ y $\mathbf{N}(\theta, \dot{\theta}, \xi, \dot{\xi})$.

Con este método se asegura la generación y estabilización de una órbita periódica forzada.

2.3. Estabilidad

La **estabilidad** es probablemente la característica más importante dentro de un sistema dinámico, pues si el mismo es inestable, estudiar sus respuestas y la convergencia se vuelve algo sin sentido.

Por su importancia, se han generado muchas definiciones y enfoques para la estabilidad. En esta tesis estudiaremos las dos perspectivas principales (estabilidad BIBO y estabilidad en el sentido de Lypaunov), y una visión distinta (estabilidad orbital), que será el eje fundamental de nuestra investigación.

2.3.1. Estabilidad BIBO

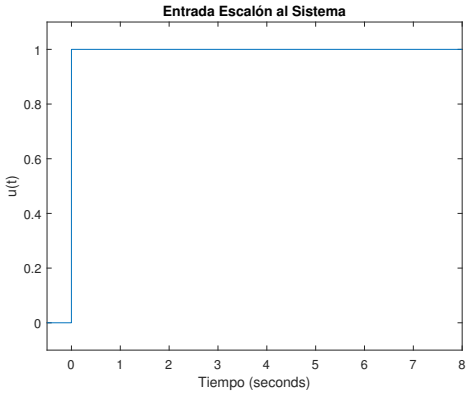
La estabilidad **Bounded Input - Bounded Output (BIBO)** nos presenta un encuadre un tanto lógico. Se basa en dos simples proposiciones que son complementarias, [Nise, 2015]:

- Un sistema es BIBO estable si para **toda** entrada acotada su respuesta (salida) es acotada (Figura 2.10).
- Un sistema es BIBO inestable si para **cualquier** entrada acotada su respuesta (salida) no es acotada (Figura 2.11).

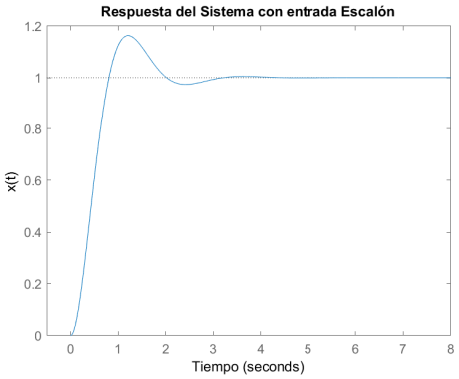
Es natural darnos cuenta que evaluar todas las entradas acotadas para cualquier sistema es una tarea imposible, entonces nace la duda, ¿cómo se analiza la estabilidad BIBO de un sistema? En realidad es una tarea que puede tener varios criterios; el criterio más común es encontrar los polos de un sistema mediante su función de transferencia (sección 2.1.4) o su modelo en el espacio de estados (sección 2.1.5) y observar en cual región del plano complejo se encuentra. Recordemos que los polos del sistema rigen su comportamiento temporal, y pueden provocar salidas acotadas o no acotadas según su posicionamiento (Figuras 2.2a y 2.2b).

A pesar de que el enfoque de la estabilidad BIBO presenta un buen primer acercamiento, no es una perspectiva completa pues solo se centra en el estudio de la salida del sistema en función de la entrada elegida, pero no del comportamiento interno del sistema y sus estados, esto provoca que sea necesaria una definición más general.

[13] La entrada de control nominal es la que permite parametrizar el sistema en función de la curva deseada, [Sætre, 2022].

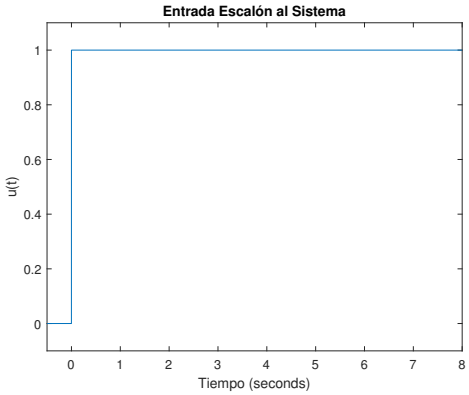


(a) Entrada Escalón (Acotada).

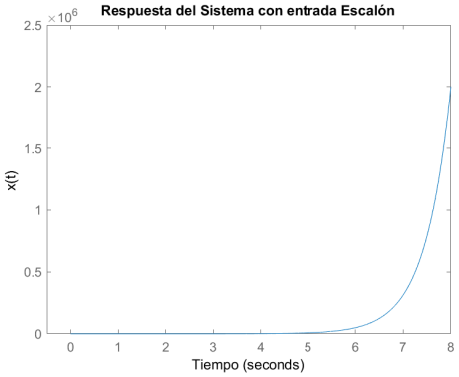


(b) Respuesta del Sistema (Acotada).

Figura 2.10 Sistema BIBO estable.



(a) Entrada Escalón (Acotada).



(b) Respuesta del Sistema (No Acotada).

Figura 2.11 Sistema BIBO inestable.

2.3.2. Estabilidad en el sentido de Lyapunov

En 1892 Alexandr Mikhailovich Lyapunov, introdujo por primera vez su teoría sobre la estabilidad de sistemas dinámicos no lineales, años después este enfoque se convirtió en el más general y útil para los sistemas de control, [Slotine y Li, 1991].

La estabilidad en el sentido de Lyapunov estudia el comportamiento de las variables de estado de un sistema en torno a un punto de equilibrio. [Slotine y Li, 1991] define a los puntos de equilibrio de la siguiente manera:

Definición 2.3.1 (Punto de equilibrio) *Un estado* $\mathbf{x}^*$ *es un estado de equilibrio (o punto de equilibrio) del sistema, si una vez que* $\mathbf{x}(\mathbf{t})$ *es igual a* $\mathbf{x}^*$*, se mantiene ahí para todo tiempo futuro (en ausencia de entradas y perturbaciones).*

Cabe destacar que la definición 2.3.1 aplica a los puntos naturales en los que el sistema llega al equilibrio; no obstante, si se hace uso de un controlador es posible elegir cualquier estado en el cual se llegue al equilibrio; pero, este será llamado **punto de operación**.

A grandes razgos, un sistema es estable si al iniciarlo en un lugar cercano al punto de operación (o equilibrio) deseado, se mantiene alrededor de ese punto siempre. Analizando desde el punto de vista matemático se puede definir la estabilidad en el sentido de Lyapunov desde la perspectiva de [Sætre, 2022]:

Definición 2.3.2 (Estabilidad en el sentido de Lyapunov) *Un punto de equilibrio* $\mathbf{x}^*(\mathbf{t})$ *se considera:*

1. ***Estable*** *si para todo* $\epsilon > 0 \ \exists \ \delta = \delta(\epsilon) > 0 \ \forall \ \mathbf{x}(\cdot; \mathbf{x_0}), \ \mathbf{x}(0, \mathbf{x_0}) = \mathbf{x_0} \rightarrow \|\mathbf{x_0} - \mathbf{x}^*(0)\| < \delta \implies \|\mathbf{x}(t, \mathbf{x_0}) - \mathbf{x}^*(t)\| < \epsilon$ *para todo* $t \geq 0$.
2. ***Asintóticamente Estable*** *si es estable en el sentido de Lyapunov y* $\|\mathbf{x}(t, x_0) - \mathbf{x}^*(t)\| \rightarrow 0$ *cuando* $t \rightarrow \infty$.
3. ***Exponencialmente Estable*** *si es estable en el sentido de Lyapunov y existen las constantes* $\delta, \ C, \ \lambda$ *tal que* $\|\mathbf{x_0} - \mathbf{x}^*(0)\| < \delta$ *implica* $\|\mathbf{x}(t, x_0) - \mathbf{x}^*(t)\| \leq Ce^{-\lambda t}$ *para todo* $t \geq 0$.

A simple vista la definición 2.3.2 parece ser muy complicada; pero, no lo es en realidad. Imaginemos dos círculos, uno con radio δ y otro más grande con radio ϵ que ambos circundan a la solución $\mathbf{x}^*(\mathbf{t})$. Un sistema es **estable en el sentido de Lyapunov** (ESL), si al tener una condición inicial x_0 dentro del círculo con radio δ, en todo momento los estados $\mathbf{x}(\mathbf{t}, \mathbf{x_0})$ se mantienen dentro del círculo con radio ϵ (Figura 2.12a). Además se vuelve **asintóticamente estable en el sentido de Lyapunov** (AESL) si conforme pasa el tiempo los estados tienden a la solución $\mathbf{x}^*(t)$ siguiendo una asíntota (Figura 2.12b) o una función exponencial (Figura 2.12c). Si el sistema no cumple con alguno de los casos anteriores entonces se deduce que el sistema es **inestable en el sentido de Lyapunov** (-ISL- Figura 2.12d).

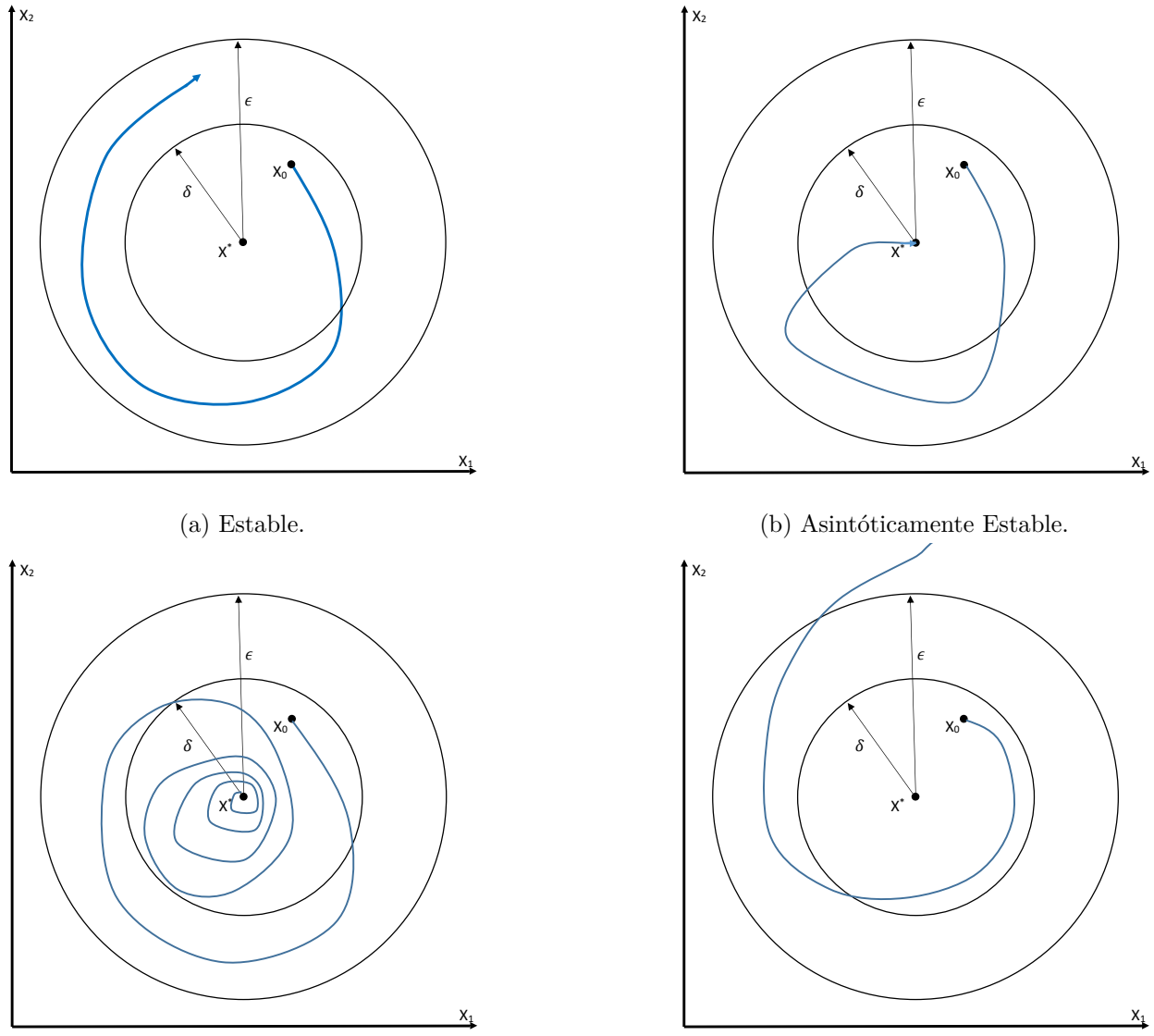
Si la estabilidad asintótica (o exponencial) es válida en todo el espacio de estados ($\mathbf{R}^n$) con cualquier condición inicial, el punto de equilibrio es asintóticamente (o exponencialmente) estable de manera global (GAS), [Slotine y Li, 1991].

Aunque la figura 2.12 nos presenta una forma cualitativa de comprobar estabilidad, debemos valernos de una herramienta cuantitativa para determinar la estabilidad en el sentido de Lyapunov de los sistemas. Como estamos tratando de dar una explicación generalizada (esquemas lineales o no lineales) a continuación se describe el **método directo de Lyapunov**.

2.3.2.1. Método directo de Lyapunov

Existe una relación conceptual entre la estabilidad y la evolución energética de un sistema. Físicamente la energía de un sistema (lineal o no lineal) es disipada constantemente, logrando que eventualmente se llegue a un estado de equilibrio. Así es posible deducir que para analizar la estabilidad es necesario examinar la evolución de una función escalar. Partiendo de lo anterior, el propósito principal del método directo de Lyapunov es encontrar una función parecida a la de energía, con el fin de analizar la estabilidad sin obtener una solución explicita a las ecuaciones diferenciales del sistema, [Slotine y Li, 1991]. Partiendo de un sistema no lineal dado por:

$$\dot{x}(t) = f(t, \mathbf{x}). \tag{2.54}$$



(a) Estable.

(b) Asintóticamente Estable.

(c) Exponencialmente Estable.

(d) Inestable.

Figura 2.12 Estabilidad en el sentido de Lyapunov

Las funciones necesarias para analizar la estabilidad se llaman **funciones de Lyapunov** ($V(\mathbf{x})$), y para que una función sea **candidata de Lyapunov** debe cumplir lo siguiente:

- La función debe depender de todo el estado $\mathbf{x}(t)$ (podría también depender del tiempo, aunque no es necesario).
- $V(\mathbf{x})$ debe ser **positiva definida**. [Shtessel, *et al.*, 2014] presenta la definición de una función positiva definida:

  Definición 2.3.3 (Función positiva definida) *Considerando un dominio $\mathcal{D} \subset \mathbb{R}^n$ en el espacio de estados que contiene al punto de equilibrio. Una función $V(\mathbf{x})$ es **positiva definida** en el dominio $\mathcal{D}$ si:*

- $V(\mathbf{x}) \geq 0$ *para todo* $\mathbf{x} \in \mathcal{D}$.
- $V(\mathbf{x}) = 0$ *implica* $\mathbf{x} = 0$.

- $V(\mathbf{x})$ tiene un mínimo en el punto de equilibrio de interés.

Finalmente, el teorema de Lyapunov para asegurar que $V(\mathbf{x})$ es una función de Lyapunov y por lo tanto que el sistema es estable lo da [Shtessel, *et al.*, 2014]:

Teorema 2.3.1 (Estabilidad Local) *Considerando el sistema no lineal de la ec.*(2.54)*. Si existe una función positiva definida y diferenciable* $V : \mathbb{R}$ *x* $\mathcal{D} \in \mathbb{R}$ *tal que:*

$$\dot{V} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} f(t, \mathbf{x}) \leq 0$$

entonces el punto de equilibrio es ***estable****. Si además,* $\dot{V} < 0$ *para* $\mathbf{x} \neq 0$ *entonces el punto de equilibrio es* ***asintóticamente estable****.*

Para garantizar que la estabilidad sea global, solo debemos aseverar que $\mathcal{D} = \mathbb{R}^n$. Esto se logra si $V(\mathbf{x})$ es **radialmente no acotada** ($\lim_{\mathbf{x}\to\infty} V(\mathbf{x}) = \infty$), es decir, crece en todo $\mathbf{x}(t)$.

Teorema 2.3.2 (Estabilidad Global) *Considerando el sistema no lineal de la ec.*(2.54)*. Si existe una función positiva definida, radialmente no acotada y diferenciable* $V : \mathbb{R}$ *x* $\mathcal{D} \in \mathbb{R}$ *tal que:*

$$\dot{V} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} f(t, \mathbf{x}) \leq 0$$

entonces el punto de equilibrio es ***globalmente estable****. Si además,* $\dot{V} < 0$ *para* $\mathbf{x} \neq 0$ *entonces el punto de equilibrio es* ***asintóticamente estable de manera global****.*

2.3.2.2. Funciones de Lyapunov comunes

Hasta este momento hemos descrito los requerimientos para que una función sea candidata de Lyapunov, no obstante, el método para encontrar estas funciones no se ha estudiado. En la literatura hay diversas maneras para tratar de encontrar este tipo de funciones; sin embargo, de forma usual se utilizan funciones cuadráticas, pues cumplen con las propiedades de ser positivas definidas y radialmente no acotadas. A continuación se mencionan las funciones más comunes según la naturaleza del sistema.

1. **LTIs**: Debido a su característica de invarianza temporal, es posible definir una función cuadrática candidata de Lyapunov de la siguiente manera:

$$V(\mathbf{x}) = \mathbf{x}^T \mathbf{P} \mathbf{x} \tag{2.55}$$

Si la matriz $\mathbf{P}$ es positiva definida, la ec.(2.55) es una función candidata de Lyapunov.

Al derivar $V(\mathbf{x})$:

$$\dot{V}(\mathbf{x}) = \dot{\mathbf{x}}^T \mathbf{P} \mathbf{x} + \mathbf{x}^T \mathbf{P} \dot{\mathbf{x}}$$

Y considerando un LTIs $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$:

$$\dot{V}(\mathbf{x}) = (\mathbf{A}\mathbf{x})^T\mathbf{P}\mathbf{x} + \mathbf{x}^T\mathbf{P}\dot{\mathbf{x}} \implies \dot{V}(\mathbf{x}) = \mathbf{x}^T\mathbf{A}^T\mathbf{P}\mathbf{x} + \mathbf{x}^T\mathbf{P}(\mathbf{A}\mathbf{x})$$
$$\dot{V}(\mathbf{x}) = \mathbf{x}^T(\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A})\mathbf{x}$$

De debe garantizar que $\dot{V}(\mathbf{x}) < 0$, así que se define una nueva matriz simétrica $\mathbf{Q}$, tal que:

$$\mathbf{Q} = -(\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A})$$
$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} + \mathbf{Q} = \mathbf{0} \tag{2.56}$$

A la ec.(2.56) se le conoce como **ecuación de Lyapunov**, y postula los siguientes resultados, [Antsaklis y Michel, 2007]:

Teorema 2.3.3 (Estabilidad para LTIs) *Dado el sistema* $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ *y considerando la ec.*(2.56)*, el punto de equilibrio es:*

a) ***ESL*** *si* $\exists\ \mathbf{P} > 0$ *tal que* $\mathbf{Q} \geq 0$.

b) ***AESL*** *si* $\exists\ \mathbf{P} > 0$ *tal que* $\mathbf{Q} > 0$.

c) ***ISL*** *si* $\exists\ \mathbf{P} < 0, \mathbf{P} \leq 0$ *tal que* $\mathbf{Q} > 0$.

2. **LTVs**: Partiendo del sistema $\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x}$, se podría pensar que al igual que los LTIs, si los eigenvalores de la matriz $\mathbf{A}(t)$ tiene parte real negativa, el sistema es estable; sin embargo, esta aseveración no es correcta. Para el análisis de estabilidad de un sistema variante en el tiempo se debe cumplir que $\mathbf{A}(t) + \mathbf{A}^T(t)$ tiene todos sus eigenvalores con parte real negativa, [Slotine y Li, 1991], i.e.:

$$\exists\ \lambda > 0, \forall i, \forall t \geq 0, \lambda_i(\mathbf{A}(t) + \mathbf{A}^T(t)) \leq -\lambda$$

Si se utiliza una función cuadrática candidata de Lyapunov $V(\mathbf{x}) = \mathbf{x}^T\mathbf{x}$, se tiene:

$$\dot{V} = \mathbf{x}^T\dot{\mathbf{x}} + \dot{\mathbf{x}}^T\mathbf{x} = \mathbf{x}^T(\mathbf{A}(t) + \mathbf{A}^T(t))\mathbf{x} \leq -\lambda\mathbf{x}^T\mathbf{x} = -\lambda V$$
$$\implies 0 \leq \mathbf{x}^T\mathbf{x} \leq V(0)e^{-\lambda t}$$

Esto quiere decir que el sistema tiende al punto de equilibrio exponencialmente.

Existe otra metodología para determinar la estabilidad de los LTVs, se basa en los siguientes principios: si $\mathbf{A}(t)$ es una matriz acotada, sus eigenvalores son de parte real negativa y la integral

$$\int_0^\infty \mathbf{A}^T(t)\mathbf{A}(t)\ dt < \infty$$

existe y es finita, el sistema es GAS.

2.3.3. Estabilización Orbital

El principal eje de esta tesis es generar y estabilizar una **órbita forzada**, a partir de un control en lazo cerrado; por lo cual, es necesario definir los conceptos básicos de órbita y estabilización orbital.

Existen algunos sistemas en los que es mucho más importante que los estados sigan un movimiento deseado, antes que evaluar el tiempo en el que eso ocurre. De aquí nace la premisa de la **estabilización orbital**, que de forma general se puede entender como la acción que permite converger asintóticamente a una órbita trazada por la solución del espacio de estados; en principio, sin importar el tiempo de alcance a la órbita.

Pero, ¿qué es una órbita? Partiendo del sistema descrito por la ec.(2.54) con una solución $\mathbf{x}^*(t)$ definida para todo $t \geq 0$. A la curva que traza la evolución temporal del sistema se le llama **trayectoria**, mientras que la curva de estado que se obtiene de la proyección sobre el espacio de estados se le conoce como **órbita**, [Sætre, 2022], y está descrita por:

$$\mathcal{O} := \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} = \mathbf{x}^*(t),\ t \in \mathbb{R}_{\geq 0}\} \tag{2.57}$$

Para entender la diferencia entre una **trayectoria** y una **órbita**, hay que entender su concepto. La trayectoria se asocia con la solución $\mathbf{x}^*$ del sistema y con el tiempo inicial t_0. La órbita únicamente depende de la solución, pues por naturaleza es invariante en el tiempo e insensible a desfases temporales; esta invarianza temporal nos evidencía una característica de las órbitas: no pueden tener intersecciones consigo mismas.

En la literatura se encuentra que existen varios tipos de órbitas; sin embargo, para este trabajo nos centraremos exclusivamente en las **órbitas periódicas**. En su tesis doctoral [Sætre, 2022] define a este tipo de órbitas de la siguiente manera:

Definición 2.3.4 (Órbitas periódicas -ciclos-) *Una órbita no trivial*[14] *que sus soluciones son periódicas, i.e. existe alguna $T > 0$, tal que $\mathbf{x}^*(t+T) = \mathbf{x}^*(t)$ para toda $t \geq 0$.*

Debido a que en la noción de estabilización orbital debemos de deshacernos de la dependencia temporal, la estabilidad en el sentido de Lyapunov parece no encajar de manera correcta, mucho menos cuando se habla de estabilidad asintótica, a causa de que la solución del espacio de estados se acota a la órbita deseada, esto provoca que en ningún momento la distancia entre la solución y el punto de equilibrio pueda ser 0. De esta manera es necesaria otra definición de estabilidad, en la que podamos utilizar una función de distancia entre el estado $\mathbf{x}$ y algún punto de la órbita $\mathbf{y} \in \mathcal{O}$:

$$\text{dist}(\mathcal{O}, \mathbf{x}) := \inf_{\mathbf{y} \in \mathcal{O}} \|\mathbf{x} - \mathbf{y}\|.$$

El término $\text{ínf}_{\mathbf{y} \in \mathcal{O}}$ quiere decir que la distancia tiene una cota inferior en la órbita $\mathcal{O}$, y se le llama **elemento ínfimo**.

El fundamento que mejor se ajusta a los requerimientos dados es la definición de **estabilidad órbital (o de Poincarè)**, [Sætre, 2022]:

[14] Una órbita trivial es tal cual un punto en el espacio $\mathbb{R}^n$, que en esta tesis es llamado **punto de equilibrio**.

Definición 2.3.5 (Estabilidad Orbital -de Poincarè-) *La solución* $\mathbf{x}^*(\mathbf{t})$ *se considera:*

1. ***Orbitalmente estable*** *si para todo* $\epsilon > 0 \; \exists \; \delta = \delta(\epsilon) > 0 \; \forall \; \mathbf{x}(\cdot; \mathbf{x_0}),\; \mathbf{x}(0, \mathbf{x_0}) = \mathbf{x_0} \rightarrow \text{dist}(\mathcal{O}, \mathbf{x}) < \delta \implies \text{dist}(\mathcal{O}, \mathbf{x}) < \epsilon$ *para todo* $t \geq 0$.
2. ***Asintótica y orbitalmente estable*** *si es orbitalmente estable y* $\text{dist}(\mathcal{O}, \mathbf{x}) \rightarrow 0$ *cuando* $t \rightarrow \infty$.
3. ***Exponencial y orbitalmente Estable*** *si es orbitalmente estable y existen las constantes* $\delta,\; C,\; \lambda$ *tal que* $\text{dist}(\mathcal{O}, \mathbf{x}) < \delta$ *implica* $\text{dist}(\mathcal{O}, \mathbf{x}) \leq Ce^{-\lambda t}$ *para todo* $t \geq 0$.

Con la definición 2.3.5 se denota que el sentido de estabilidad orbital no es más que llevar el sentido de Lyapunov a una vecindad de la órbita $\mathcal{O}$, en lugar de a las soluciones $\mathbf{x}^*(t)$ que dependen del tiempo. Así se puede concluir de manera general que para las soluciones $\mathbf{x}^*(t)$ de la ec.(2.54), la estabilidad en el sentido de Lyapunov implica directamente estabilidad orbital, mas no necesariamente al revés.

2.3.3.1. Órbita parametrizada

Al inicio de la sección 2.3.3, se introdujó por primera vez el concepto de **órbita** y se propusó como la proyección sobre el espacio de estados de una sistema no lineal. Inicialmente se describió con el sistema dado por la ec.(2.54); sin embargo, como se requiere quitar la dependencia temporal del sistema, se considera ahora un sistema no lineal invariante en el tiempo con una entrada de control:

$$\dot{\mathbf{x}} = f(\mathbf{x}) + B(\mathbf{x})\mathbf{u} \tag{2.58}$$

Una vez más, $\mathbf{x}(t) \in \mathbb{R}^n$ es el vector de estados, $\mathbf{u} \in \mathbb{R}^m$ es el vector de entradas y la función vectorial $f(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^n$ así como la función matricial $B(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}$ son al menos dos veces diferenciables $\mathcal{C}^2$.

Es necesario entender el concepto de la parametrización de una órbita. Se puede definir una curva utilizando una única variable a la que denotaremos como $s = s(t)$, esta variable pertenece a un espacio $S \subseteq \mathbb{R}$. La idea principal es poder definir a los estados y a las entradas en función de la variable s y generar un conjunto de posibles soluciones llamado *maneuver*:

$$\mathcal{M} := \{(\mathbf{x}, \mathbf{u}) \in \mathbb{R}^n \times \mathbb{R}^m : \mathbf{x} = \mathbf{x_s}(s),\; \mathbf{u} = \mathbf{u_s}(s),\; s \in S\}\,. \tag{2.59}$$

El conjunto de soluciones de la ec.(2.59) es posible al tener distintas entradas, lo cual provoca una cantidad diversa de órbitas deseadas; pero, si se tiene la dupla forzada $(\mathbf{x_s}(s), \rho(s))$ se puede obtener una única entrada $\mathbf{u_s}(s)$. La función $\rho(s)$ es el **operador de proyección** y permite definir la trayectoria (órbita) deseada[15].

Considerando las aseveraciones del párrafo anterior, se encuentra la proyección del *maneuver* $\mathcal{M}$ sobre el espacio de estados, lo que ahora se conoce como órbita $\mathcal{O}$:

$$\mathcal{O} := \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} = \mathbf{x_s}(s),\;\; s \in S\} \tag{2.60}$$

[15]La definición formal se encuentra en la página 67 de la tesis de [Sætre, 2022] -*Definition 4.5*-.

Para asegurar que la proyección obtenida es la que lleva a el *maneuver* sobre el espacio de estados de la ec.(2.58), se debe cumplir la siguiente propiedad, [Sætre, 2022]:

$$\dot{s} = \rho(s) \tag{2.61}$$

Como la ec.(2.59) y la ec,(2.60) dependen del espacio de estados dado por la ec.(2.58), es evidente que para que la órbita $\mathcal{O}$ exista, es forsozamente necesaria la entrada de control $\mathbf{u}$, es decir es una **órbita forzada**. También ambas ecuaciones nos permiten encontrar las siguientes propiedades para ella:

1. $\mathcal{O}$ es $\mathcal{C}^2$ y una subvariedad de dimensión 1 o nula de $\mathbb{R}^n$.
2. Se puede desvanecer únicamente en puntos aislados s_e para $\rho(s_e) \equiv 0$.
3. $\mathcal{O}$ es una subvariedad embebida de $\mathbb{R}^n$, lo que implica que existe una vecindad (con radio $\epsilon > 0$) en el que cada punto tiene una única proyección ortonormal con la órbita.
4. Es un espacio con control invariante respecto al sistema representado por la ec.(2.58).

La última propiedad denota que el *maneuver* y la órbita son consistentes con la dinámica de la ec.(2.58).

A raiz de la propiedad 3 se puede definir una vecindad $\mathcal{N}_{\mathcal{O}}(\delta)$ que rodea a la órbita forzada:

$$\mathcal{N}_{\mathcal{O}}(\delta) := \{\mathbf{x} \in \mathbb{R}^n : \text{ dist}(\mathcal{O}, \mathbf{x}) < \delta\}$$

Con todo lo anterior es claro que para poder trabajar con órbitas forzadas, es necesario encontrar una entrada $\mathbf{u}$ que no solo permita estabilizarlas dentro de una vecindad, si no también generar el movimiento para llegar a ellas; de aquí nace el planteamiento del problema para la estabilización orbital, [Sætre, 2022]:

Problema 2.3.1 (Problema de Estabilización Orbital) *Para la ec.*(2.58) *obtener una ley de control* $\mathbf{u} = k(\mathbf{x})$*, donde* $k : \mathbb{R}^n \to \mathbb{R}^m$ *es localmente Lipschitz y continua en una vecindad suficientemente grande de* $\mathcal{O}$*, tal que* $\mathcal{O}$ *sea asintóticamente estable para el sistema en lazo cerrado:*

$$\dot{\mathbf{x}} = f(\mathbf{x}) + B(\mathbf{x})k(\mathbf{x}) \tag{2.62}$$

y que para cualquier $\epsilon > 0$*, exista un* $\delta = \delta(\epsilon) > 0$ *tal que las soluciones* $\mathbf{x}(\cdot)$ *de la ec.*(2.62) *satisfagan:*

- $\mathbf{x}(t) \in \mathcal{N}_{\epsilon}(\mathcal{O})$ *para todo* $t \geq t_0$ *(Estabilidad).*
- $\text{dist}(\mathcal{O}, \mathbf{x}(t)) \to 0$ *si* $t \to \infty$ *(Atracción).*

En la sección 2.2.5.1 se encuentra uno de los métodos para diseñar la entrada de control $\mathbf{u}$ que permite resolver el problema 2.3.1. Este método será el implementado en este trabajo de tesis.

Capítulo 3

Implementación del Control por Linealización de Coordenadas Transversales

En este capítulo se desarrolla la implementación del control por linealización de coordenadas transversales para el prototipo conocido como *Butterfly Robot*. En la sección 3.1 se presenta el planteamiento del problema a resolver, se familiariza con la planta, y se obtiene el modelo dinámico de la misma.

En la sección 3.2 se presenta el esquema de control propuesto para garantizar la estabilización orbital así como la deducción de la entrada de control necesaria para asegurar la estabilidad orbital.

Finalmente, en la sección 3.3 se lleva a cabo el análisis de las aplicaciones desarrolladas en el entorno de *C++*, para su implementación en el prototipo físico.

3.1. Planteamiento del problema

Como se mostró en la última sección del Capitulo 2, el enfoque principal de esta tesis es resolver el problema de estabilización orbital (Problema 2.3.1); sin embargo, por las características del sistema *Butterfly Robot* se centrará únicamente en el caso de órbitas periódicas, donde se asume que el sistema admite una solución $\mathbf{x}^*(t+T) = \mathbf{x}^*(t)$, para un periodo finito $T > 0$. Partiendo del sistema dinámico

$$\dot{\mathbf{x}} = f(\mathbf{x}) + B(\mathbf{x})\mathbf{u}. \tag{3.1}$$

.
El problema principal sufre una pequeña modificación, resultando en lo siguiente, [Sætre, 2022]:

Problema 3.1.1 (Problema de Estabilización Orbital Periódica) *Encontrar una función matricial* $\mathbf{u} = k(\mathbf{x})$*, donde* $k : \mathbb{R}^n \to \mathbb{R}^m$ *sea* $\mathcal{C}^1$ *y satisfaga* $\|k(y)\| = 0$ *para toda* $y \in \mathcal{O}$*, tal que la entrada de control force la convergencia de las trayectorias del sistema* (3.1) *a la órbita*

$$\mathcal{O} = \{\mathbf{x} \in \mathbb{R}^n : \ \mathbf{x} = \mathbf{x}^*(t),\ t \in [t_{conv}, T)\},$$

en un tiempo de convergencia t_{conv} *y la convierta en un* ***ciclo límite asintóticamente estable.***

De esta manera, se utilizará el algoritmo de estabilización orbital mediante la linealización de coordenadas transversales propuesto por [Shiriaev, *et al.*, 2010] para resolver el problema 3.1.1 en un caso de aplicación denominado *Butterfly Robot*. Por lo anterior es necesario adentrarnos y entender las características físicas y matemáticas del prototipo.

3.1.1. *Butterfly Robot*

El primer acercamiento al sistema *Butterfly Robot* es encontrado en el artículo de [Lynch, *et al.*, 1998], donde se propone que una pelota puede seguir un movimiento a manera de malabarismo sobre la mano de una persona. Imaginemos que la pelota empieza su movimiento en el centro de la palma de la mano, si generamos una fuerza que desplace a la pelota hacia los dedos, y posteriormente por el dorso hasta la muñeca, se obtiene la trayectoria propia que define a la mariposa (Figura 3.1).

Figura 3.1 Movimiento que define la trayectoria de la mariposa.

Realizar la maniobra de la mariposa no es precisamente una tarea sencilla, y la razón es que en ningún momento la pelota es sujetada por la mano, tan solo gira alrededor de ella. A este tipo de fenómenos, donde el objeto a controlar (la pelota) no está sujeto por el objeto actuado (la mano), se les llama **sistemas no prensiles**, [Lynch, *et al.*, 1998]. Si consideramos además que el movimiento no es generado por una mano, sino por un robot, nace la pregunta: ¿cómo le enseñamos a un robot a manipular objetos que no sostiene?

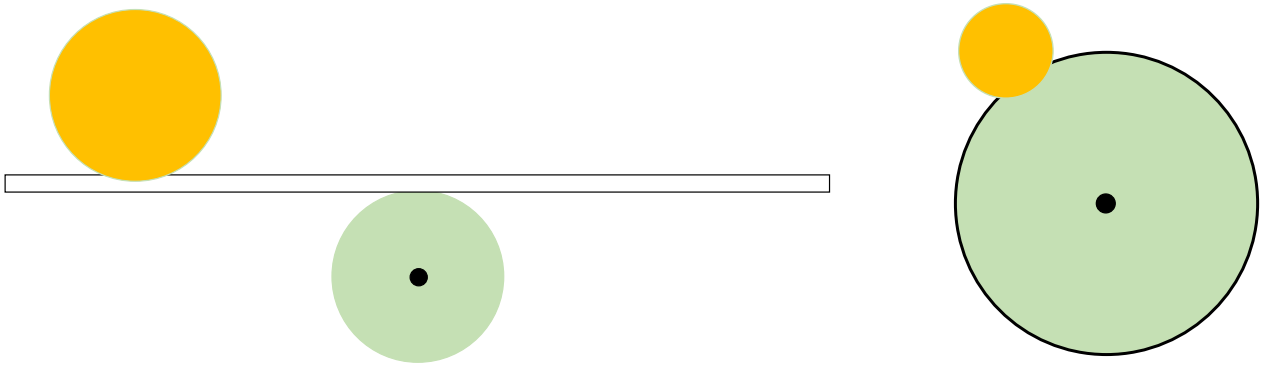
Aunque intuitivamente el no tener sujeción sobre el objeto a controlar podría ser considerado algo completamente malo, el robot puede utilizar las fuerzas gravitatorias, centrífugas y de Coriolis para controlar (virtualmente) más grados de libertad de los que podría al tomar el elemento, [Lynch y Mason, 1999]. Los sistemas no prensiles pueden, en realidad, presentar algunos beneficios:

- **Sujeción:** La dinámica no prensil permite a robots manipuladores lidiar con objetos muy largos, pesados o con figuras complejas para ser sujetados.

- **Flexibilidad:** Permite al manipulador controlar partes diversas al mismo tiempo, con alguna sección actuada.
- **Mayor dimensión en el espacio de trabajo:** Cuando se tiene sujeción, es necesario que el manipulador tenga al menos la misma cantidad de grados de libertad que el objeto a controlar. Si el objeto se puede mover en el manipulador (no existe sujeción) es posible controlar (indirectamente) más grados de libertad de los del robot.

Para poder contar con estos beneficios es necesario tener un conocimiento profundo de la planta, con el fin de poder obtener una buena planeación del movimiento, y por ende, un buen controlador, [Lynch y Mason, 1999].

Existen ejemplos comunes de sistemas no prensiles (Figura 3.2), como lo son el *Ball and Beam* ([Hauser, *et al.*, 1992]), o el *Ball-on-disk* ([Ryu, *et al.*, 2013]), estas aplicaciones se asemejan al *Butterfly Robot* pues de igual forma maniobran una pelota sobre platos sin contar con sujeción sobre ella; entonces, ¿por qué utilizar el ejemplo del ***Butterfly Robot***? Resulta que por su forma y tamaño (Figura 3.3) es verdaderamente complicado producir el movimiento deseado; tanto que, según [Surov, *et al.*, 2015], los primeros enfoques para resolver el problema de estabilización orbital en el *Butterfly Robot* desarrollados por [Ryu, *et al.*, 2013] y [Cefalo, *et al.*, 2006] evitan el uso de un modelo avanzado para la planeación del movimiento, y en sus modelos dinámicos desprecian las fuerzas de contacto de la pelota con la mariposa y la rotación de la misma, aún cuando ambos efectos no son triviales.



(a) *Ball and Beam* (b) *Ball-on-Disk*

Figura 3.2 Sistemas no prensiles distintos al *Butterfly Robot*.

3.1.2. Prototipo *Butterfly Robot*

Para poder controlar al sistema *Butterfly Robot* primero se tiene que saber cómo se conforma físicamente la planta. De primera instancia se puede dividir en 2 partes esenciales al prototipo completo, el **sistema mecánico** y el **sistema de control**, este último a su vez se divide en 2 subsistemas: el primero se encarga de interpretar las señales y calcular la entrada de control necesaria; y, el segundo de recibir esta señal y comunicarsela al motor utilizado.

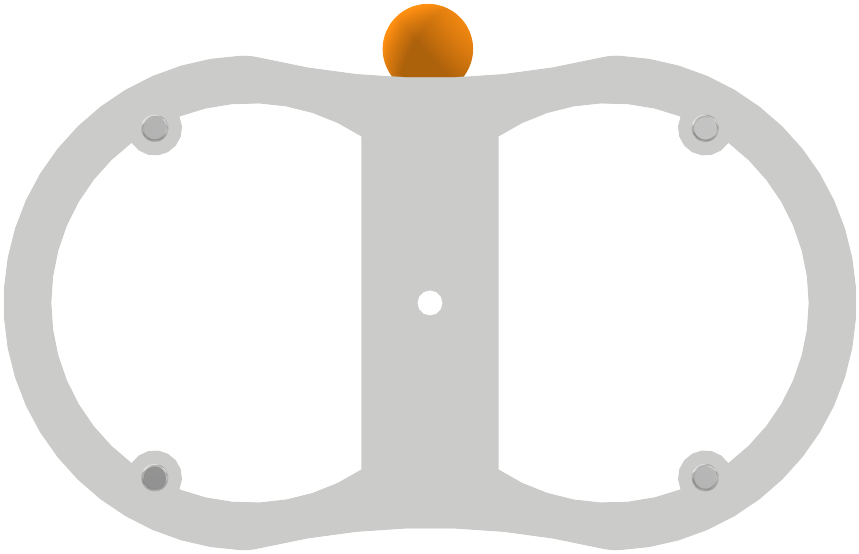


Figura 3.3 Sistema *Butterfly Robot*.

De forma esquemática se presentan las siguientes partes (Figura 3.4):

- **Sistema Mecánico**
  1. Dos platos transparentes con la figura de la mariposa.
  2. Pelota naranja de 33.2 [mm] de diámetro.
- **Sistema de Control**
  3. Motor *Maxon 590953*.
  4. *Encoder* de alta resolución $8192\frac{pts}{rev}$.
  5. Cámara *Basler acA 1300-200uc*.
  6. *Beaglebone Green* - Controlador del motor (No mostrado en la figura 3.4).
  7. Intel Nuc (CPU - No mostrado en la figura 3.4).

Como se muestra anteriormente, el sistema mecánico se compone únicamente por los dos platos transparentes que forman a la mariposa y la pelota naranja. Por su parte, el sistema de control se compone de un segmento de visión y sensado, constituido por la cámara y el *encoder* acoplado al motor; una parte de procesamiento del controlador (CPU y *Beaglebone*); y, finalmente la sección de actuación (Motor).

3.1.3. Dinámica del *Butterfly Robot*

Dentro del marco para el modelado de un sistema dinámico presentado en la sección 2.1.3, es necesario primero **identificar sus variables de estado (grados de libertad)**. El sistema dinámico correspondiente al prototipo *Butterfly Robot* se centra en los platos transparentes y concéntricos con la forma de la mariposa (accionados por el motor) y la pelota que se maniobra sobre ellos.

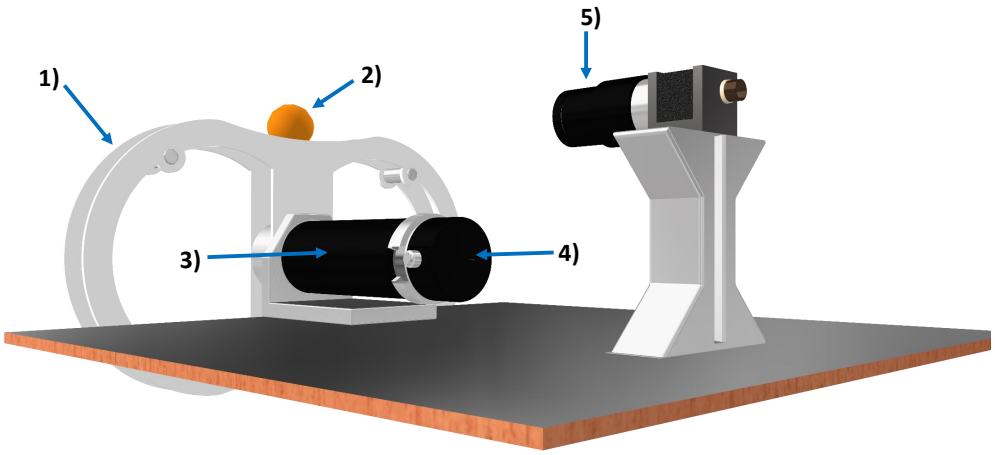


Figura 3.4 Prototipo *Butterfly Robot*.

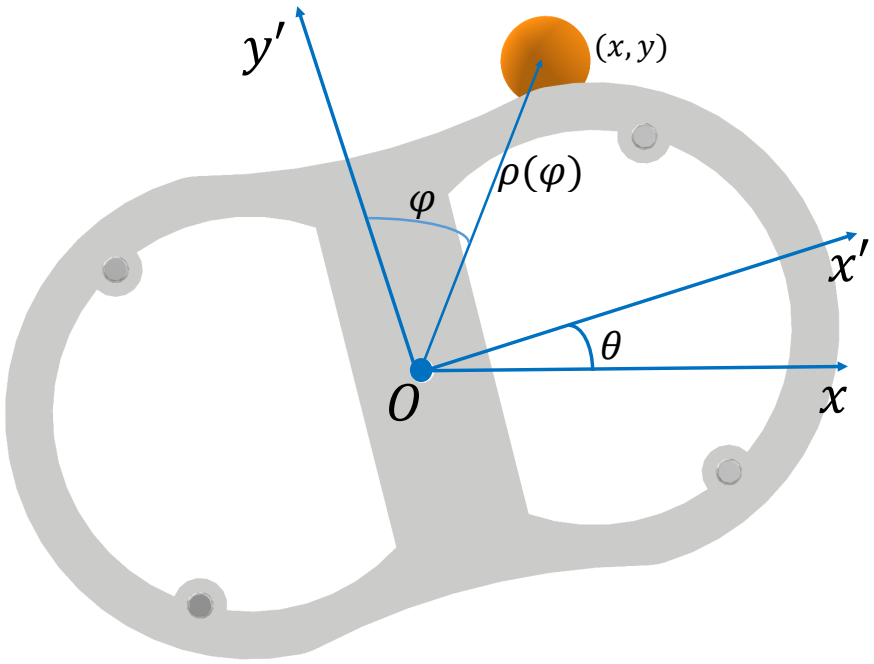


Figura 3.5 Coordenadas Generalizadas.

En la figura 3.5 se muestran las 4 coordenadas generalizadas del *Butterfly Robot*, descritas a continuación:

- La mariposa (platos) tiene un grado de libertad (θ):
  - θ es la posición angular de los platos, permite conocer la variación angular del marco de referencia xy con respecto al marco de referencia $x'y'$.

- La pelota tiene tres grados de libertad (φ, x, y):
  - φ es la posición angular de la pelota respecto al sistema de coordenadas $x'y'$.
  - (x, y) es la posición de la pelota respecto al marco inercial de referencia.

El siguiente paso es **encontrar el vector de posición de la pelota** en función de estas variables de estado; sin embargo, hasta este punto, las variables de estado no presentan información alguna sobre la forma de la órbita (curva) deseada, entonces ¿cómo se encuentra este vector de posición? Debido a que se asume que la pelota siempre está sobre los platos que definen a la mariposa, ahora es necesario pensar en las coordenadas del sistema referidas a esta curva, por lo cual se debe incluir una parametrización. Esta información se puede encontrar a partir de la distancia entre el centro de la mariposa Oxy y el centro de la pelota, denotado por el escalar ρ.

Partamos de la figura 3.6:

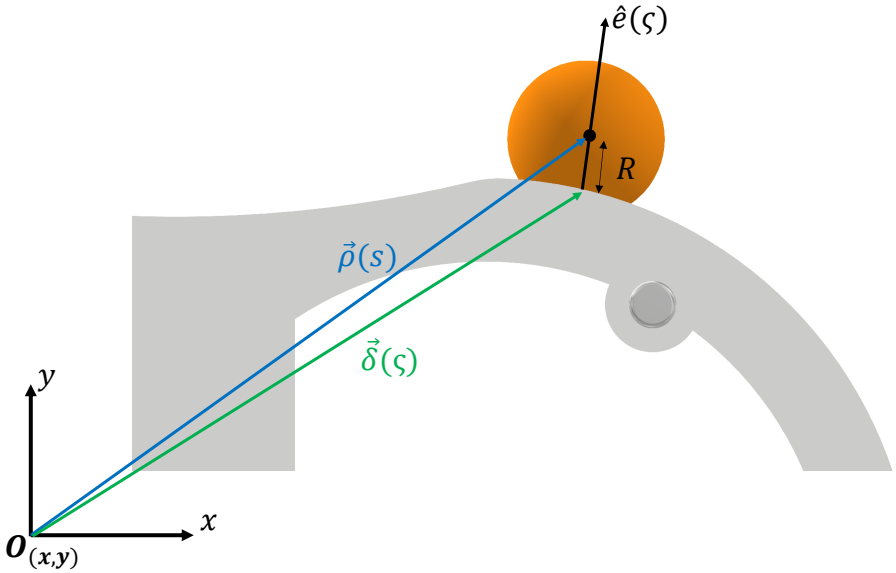


Figura 3.6 Coordenadas en la vecindad de la órbita.

Donde s y $\varsigma \in S \subseteq \mathbb{R}$ es la pararametrización del sistema y representan la posición de la pelota respecto a la curva dada por ρ y por δ, respectivamente; se cumple que $s(\varsigma) = \int_0^{\varsigma} \|\frac{d\rho}{d\varsigma}\| d\varsigma$. Además se muestran dos vectores:

1. $\vec{\delta}(\varsigma)$: representa al vector de posición en el perímetro de la curva. Su norma denota la distancia del centro del marco de referencia Oxy, al perímetro de la mariposa; esto quiere decir que $\|\vec{\delta}\| = \delta$ representa la forma de la curva y está dada por:

$$\delta(\varsigma) = 0.1095 - 0.0405 \cos 2\varsigma \tag{3.2}$$

2. $\vec{\rho}(s)$: Es el vector de posición del centro de la pelota; su norma $\|\vec{\rho}\| = \rho$ representa la distancia entre el origen Oxy y el centro de la pelota; matemáticamente se denota como

$$\vec{\rho}(\varsigma) = \vec{\delta}(\varsigma) + \hat{e}(\varsigma)R, \tag{3.3}$$

con

$$\hat{e}(\varsigma) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \delta'(\varsigma).$$

Considerando $\delta'(\varsigma) = \frac{d\delta(\varsigma)}{d\varsigma}$. Notamos que este último vector siempre apunta ortogonalmente a la órbita.

A partir de la parametrización, ya es posible definir el vector de posición de la pelota. Es necesario considerar la posición angular de los platos (θ) y el vector $\vec{\rho}(s)$

$$\mathbf{r} = \begin{bmatrix} x \\ y \end{bmatrix} = \mathbf{\Pi}(\theta)[\vec{\rho}(s)]. \tag{3.4}$$

Donde $\mathbf{\Pi}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$ es la matriz de rotación en dos dimensiones.

Si se deriva el vector dado por la ec.(3.4), se encuentra su velocidad y rapidez

$$\dot{\mathbf{r}} = \mathbf{\Pi}(\theta)\vec{\rho}'\dot{s} + \mathbf{\Pi}'(\theta)\vec{\rho}\dot{\theta}.$$

Si se toma $\vec{\rho}' = \vec{\tau}$, entonces

$$\dot{\mathbf{r}} = \mathbf{\Pi}(\theta)\vec{\tau}\dot{s} + \mathbf{\Pi}'(\theta)\vec{\rho}\dot{\theta}.$$

Entonces, la rapidez al cuadrado es

$$\begin{aligned} \|\dot{r}\|^2 &= \left(\mathbf{\Pi}(\theta)\vec{\tau}\dot{s} + \mathbf{\Pi}'(\theta)\vec{\rho}\dot{\theta}\right)^T \left(\mathbf{\Pi}(\theta)\vec{\tau}\dot{s} + \mathbf{\Pi}'(\theta)\vec{\rho}\dot{\theta}\right), \\ \|\dot{r}\|^2 &= \|\vec{\rho}\|^2\dot{\theta}^2 + 2\vec{\rho}\times\vec{\tau}\dot{\theta}\dot{s} + \dot{s}^2. \end{aligned}$$

Así la **energía cinética** está dada por

$$k = \frac{1}{2}m\|\dot{r}\|^2 = \frac{1}{2}m\left(\rho^2\dot{\theta}^2 + 2\vec{\rho}\times\vec{\tau}\dot{\theta}\dot{s} + \dot{s}^2\right).$$

Pero, se deben considerar las inercias del plato y de la pelota

$$\begin{aligned} E_{b_p} &= \frac{j_p}{2}(\dot{\theta})^2, \\ E_{b_b} &= \frac{j_b}{2}(\dot{\varphi} + \dot{\theta})^2. \end{aligned}$$

Entonces

$$k_T = \frac{1}{2}m\|\dot{r}\|^2 + \frac{j_p}{2}(\dot{\theta})^2 + \frac{j_b}{2}(\dot{\varphi} + \dot{\theta})^2.$$

Considerando que no existe deslizamiento $\dot{\varphi} = -\frac{1}{R}\dot{s}$, se tiene, [Surov, *et al.*, 2015]:

$$k_T = \frac{1}{2}m\left(\rho^2\dot{\theta}^2 + 2\vec{\rho}\times\vec{\tau}\dot{\theta}\dot{s} + \dot{s}^2\right) + \frac{j_p}{2}(\dot{\theta})^2 + \frac{j_b}{2}\left(\frac{1}{R^2}\dot{s}^2 - \frac{2}{R}\dot{\theta}\dot{s} + \dot{\theta}^2\right) \tag{3.5}$$

Para la **energía potencial** se considera la coordenada y de la pelota en el espacio.

$$E_p = mg\begin{bmatrix}0 & 1\end{bmatrix}\mathbf{\Pi}\vec{\rho} \tag{3.6}$$

Resultando en el **lagrangiano**

$$\begin{aligned}&\mathcal{L} = k_T - E_P,\\ &\mathcal{L} = \frac{1}{2}m\left(\rho^2\dot{\theta}^2 + 2\vec{\rho}\times\vec{\tau}\dot{\theta}\dot{s} + \dot{s}^2\right) + \frac{j_p}{2}(\dot{\theta})^2 + \frac{j_b}{2}\left(\frac{1}{R^2}\dot{s}^2 - \frac{2}{R}\dot{\theta}\dot{s} + \dot{\theta}^2\right) - mg\begin{bmatrix}0 & 1\end{bmatrix}\mathbf{\Pi}\vec{\rho}.\end{aligned} \tag{3.7}$$

Es de interés conocer la forma del manipulador del sistema, por lo cual retomando los conceptos estudiados en la sección 2.1.6 se encuentran las siguientes matrices y vectores:

- **Matriz de Inercia**

  Como el lagrangiano está dado por:

$$\mathcal{L} := \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - E_{P_T}(\mathbf{q}) \tag{3.8}$$

  Basta con encontrar la similitud entre la ec.(3.7) y la ec.(3.8):

$$\begin{aligned}k_T &= \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}}\\ k_T &= \frac{1}{2}m\begin{bmatrix}\dot{\theta} & \dot{s}\end{bmatrix}\begin{bmatrix}\rho^2 + \frac{J_p}{m} + \frac{J_b}{m} & \vec{\rho}\times\vec{\tau} - \frac{J_b}{mR}\\ \vec{\rho}\times\vec{\tau} - \frac{J_b}{mR} & 1 + \frac{J_b}{mR^2}\end{bmatrix}\begin{bmatrix}\dot{\theta}\\ \dot{s}\end{bmatrix}\end{aligned}$$

  Con lo que finalmente se obtiene la matriz de inercia:

$$\mathbf{M}(\mathbf{q}) = m\begin{bmatrix}\rho^2 + \frac{J_p}{m} + \frac{J_b}{m} & \vec{\rho}\times\vec{\tau} - \frac{J_b}{mR}\\ \vec{\rho}\times\vec{\tau} - \frac{J_b}{mR} & 1 + \frac{J_b}{mR^2}\end{bmatrix} \tag{3.9}$$

- **Vector de fuerzas potenciales**

  El vector de fuerzas potenciales está dado por:

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix}\frac{\partial E_{P_T}(\mathbf{q})}{\partial q_1}\\ \frac{\partial E_{P_T}(\mathbf{q})}{\partial q_2}\\ \vdots\\ \frac{\partial E_{P_T}(\mathbf{q})}{\partial q_n}\end{bmatrix} \tag{3.10}$$

Sustituyendo la ec.(3.6) en la ec.(3.10):

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} \frac{\partial E_{P_T}(\mathbf{q})}{\partial q_1} \\ \frac{\partial E_{P_T}(\mathbf{q})}{\partial q_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial E_{P_T}(\mathbf{q})}{\partial \theta} \\ \frac{\partial E_{P_T}(\mathbf{q})}{\partial s} \end{bmatrix}$$

$$\mathbf{G}(\mathbf{q}) = mg \begin{bmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} \mathbf{\Pi}' \vec{\rho} \\ \begin{pmatrix} 0 & 1 \end{pmatrix} \mathbf{\Pi} \vec{\tau} \end{bmatrix} \tag{3.11}$$

Considerando $\mathbf{\Pi}' = \mathbf{\Pi}'(\theta) = \frac{d\mathbf{\Pi}(\theta)}{d\theta}$.

- **Matriz de Coriolis**

Finalmente es posible utilizar la definición de la matriz de Coriolis:

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \sum_{i=1}^{n} \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_i} \dot{q}_i = \frac{1}{2} \begin{bmatrix} \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_1} \dot{\mathbf{q}} \\ \vdots \\ \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_n} \dot{\mathbf{q}} \end{bmatrix} \tag{3.12}$$

Sustituyendo la ec.(3.9) en la ec.(3.12):

$$\mathbf{C}(\mathbf{q}) = m \begin{bmatrix} \vec{\tau}^T \vec{\rho} \dot{s} & \vec{\tau}^T \vec{\rho} \dot{\theta} + \vec{\rho} \times \vec{\kappa} \dot{s} \\ \vec{\tau}^T \vec{\rho} \dot{\theta} & 0 \end{bmatrix} \tag{3.13}$$

Así se tiene el modelo en la forma del manipulador para las variables de estado $\mathbf{q} = \begin{bmatrix} \theta & s \end{bmatrix}^T$. Sin embargo, no es posible medir directamente la variable de estado s, por lo cual se debe utilizar una variable de estado medible además de θ. Si se considera que la pelota tiene un solo punto de contacto, es posible utilizar la variable φ, dejando únicamente las variables de estado de las posiciones angulares $\mathbf{q} = \begin{bmatrix} \theta & \varphi \end{bmatrix}^T$; no obstante el modelo en la forma del manipulador ahora se debe acoplar a las variables de estado elegidas. Según [Surov, *et al.*, 2015]:

$$\frac{\partial(\theta, s)}{\partial(\theta, \varphi)} = \begin{bmatrix} 1 & 0 \\ 0 & \|\rho'\| \end{bmatrix}$$

Lo cual implica que el modelo final está dado por:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \begin{bmatrix} u \\ 0 \end{bmatrix} \tag{3.14}$$

Donde

$$\mathbf{M}(\mathbf{q}) = m \begin{bmatrix} \rho^2 + \frac{J_p}{m} + \frac{J_b}{m} & \left(\vec{\rho} \times \vec{\tau} - \frac{J_b}{mR} \right) s' \\ \left(\vec{\rho} \times \vec{\tau} - \frac{J_b}{mR} \right) s' & \left(1 + \frac{J_b}{mR^2} \right) s'^2 \end{bmatrix}, \tag{3.15}$$

$$\mathbf{C}(\mathbf{q}) = m \begin{bmatrix} s'\vec{\tau}\cdot\vec{\rho}\dot{\varphi} & s'\vec{\tau}\cdot\vec{\rho}\dot{\theta} + \left(s'^2\vec{\rho}\times\vec{\kappa} + \left(\vec{\rho}\times\vec{\tau} - \frac{J_b}{mR}\right)s''\right)\dot{\varphi} \\ -s'\vec{\tau}\cdot\vec{\rho}\dot{\theta} & \left(1+\frac{J_b}{mR^2}\right)s's''\dot{\varphi} \end{bmatrix}, \tag{3.16}$$

$$\mathbf{G}(\mathbf{q}) = mg \begin{bmatrix} (0,1)\cdot\mathbf{\Pi}'(\theta)\vec{\rho} \\ (0,1)\cdot\mathbf{\Pi}(\theta)\vec{\tau}s' \end{bmatrix}. \tag{3.17}$$

La notación usada es:

- m: Masa de la pelota $[kg]$.
- J_p: Momento de inercia respecto al eje z de los platos $[kgm^2]$.
- J_b: Momento de inercia respecto al eje z de la pelota $[kgm^2]$.
- R: Radio de la pelota $[m]$.
- g: Aceleración debido a la gravedad $\left[\frac{m}{s^2}\right]$.
- u: Entrada de control (par del motor-$[Nm]$).
- $\mathbf{\Pi}(\theta)$: Matriz de rotación en dos dimensiones.
- $\mathbf{\Pi}'(\theta) = \frac{d\mathbf{\Pi}(\theta)}{d\theta}$: Derivada de la matriz de rotación en dos dimensiones con respecto a θ.
- $\vec{\rho} = \vec{\rho}(\varphi)$: Vector de posición del centro de la pelota en el plano $x'y'$.
- $\rho = \|\vec{\rho}\|$: Distancia en metros entre el origen O y el centro de la pelota.
- $s = s(\varphi)$: Posición de la bola a lo largo de la curva ρ. Con $s' = \frac{ds(\varphi)}{d\varphi}$ y $s'' = \frac{d^2s(\varphi)}{d\varphi^2}$.
- $\vec{\tau} = \vec{\rho}' = \frac{d\vec{\rho}}{ds}$: Vector tangente a $\vec{\rho}$ en el punto φ.
- $\vec{\kappa} = \vec{\rho}'' = \frac{d^2\vec{\rho}}{ds^2}$: Vector de curvatura a $\vec{\rho}$ en el punto φ.

Los parámetros son mostrados a continuación:

Parámetro	**Magnitud**
Masa de la pelota	$m = 0.003[kg] = 3[g]$
Momento de inercia de los platos respecto al eje z	$j_p = 0.89 \times 10^{-3}\ [kgm^2]$
Momento de inercia de la pelota respecto al eje z	$j_b = 5.8 \times 10^{-7}\ [kgm^2]$
Radio de la pelota	$R = 16.6[mm]$
Aceleración debido a la gravedad	$g = 9.81\ \left[\frac{m}{s^2}\right]$

Tabla 3.1 Parámetros del modelo del *Butterfly Robot*.

De primera instancia pueden parecer erróneos los términos $\mathbf{M}_{1,2}$ y $\mathbf{M}_{2,1}$ de la ec.(3.15), pues se realiza una resta entre un producto cruz $\vec{\rho}\times\vec{\tau}$ (que da como resultado un vector) y un escalar $\frac{J_b}{mR}$; sin embargo, se debe poner especial atención en el parámetro j_b, el cual está definido como el momento de inercia en el eje z del sistema cartesiano inercial; esta definición es de vital importancia, pues al realizar el producto cruz se obtiene lo siguiente:

$$\vec{\rho} \times \vec{\tau} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ \rho_1 & \rho_2 & 0 \\ \tau_1 & \tau_2 & 0 \end{vmatrix} = (\rho_1\tau_2 - \rho_2\tau_1)\vec{k} = \begin{bmatrix} 0 \\ 0 \\ \rho_1\tau_2 - \rho_2\tau_1 \end{bmatrix}.$$

El resultado precisamente se encuentra respecto al eje z, por lo cual es posible hacer la operación entre los valores dichos. Lo anterior también sucede en la definición de la matriz de Coriolis, descrita por la ec.(3.16), y la razón de que esto ocurra es la mera simplificación de la notación utilizada en la descripción del modelo, y se obvia ya que el sistema está anclado en ese eje (z) y solo puede moverse en los ejes x y y.

Una vez obtenido el modelo matemático que describe al *Butterfly Robot* (ec.3.14), se realizará la deducción de la entrada de control que permita resolver el problema de estabilidad orbital.

3.2. Estabilización Orbital para el *Butterfly Robot*

3.2.1. Esquema de Control

Con el fin de garantizar la estabilidad orbital del sistema *Butterfly Robot*, y por lo tanto resolver el problema de estabilización orbital periódica (3.1.1), [Sætre, 2022] propone una entrada de control descrita de la siguiente manera:

$$u(\mathbf{x}) = u_{nom}(\mathbf{x}) + u_{fb}(\mathbf{x}) + u_{rob}(\mathbf{x}) \tag{3.18}$$

Donde u_{nom} es la entrada nominal de control que permite parametrizar el sistema, es decir, $u_{nom}(x_s(s)) \equiv u_s(s)$; u_{fb} es la entrada que asegura convergencia con la órbita $\mathcal{O}$ y u_{rob} es la entrada de control que robustece al sistema para compensar perturbaciones y errores en el modelado matemático (El diagrama de bloques para el sistema de control se muestra en la figura 3.7).

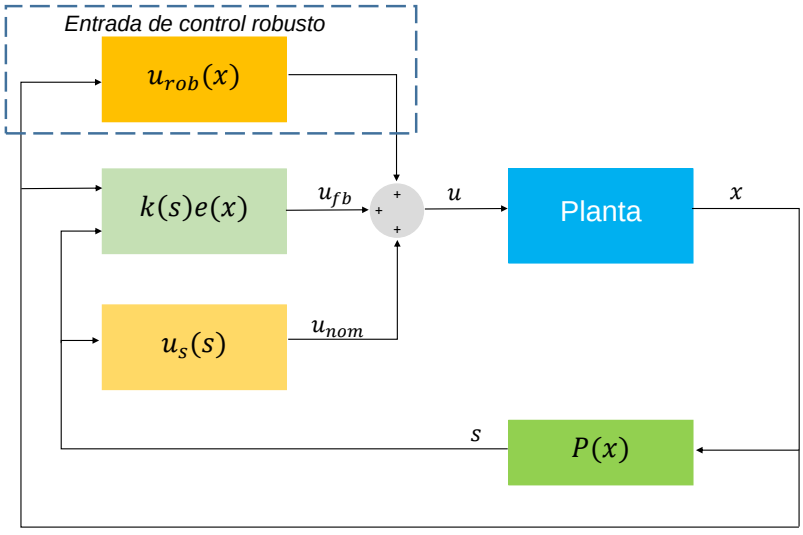


Figura 3.7 Diagrama de bloques del sistema de control.

Dentro del diagrama de bloques presentado, el término $P(x)$ es el operador de proyección (estudiado en las secciones 2.2.5.1 y 2.3.3), $k(s)$ es la matriz de ganancias y $e(x)$ es una función de error (en este caso, correspondiente a las coordenadas transversales).

Para efectos de esta tesis se acotará la entrada de control general (ec.3.18) a solamente los primeros dos términos: la entrada de control nominal u_{nom} y la entrada de control de *feedback* u_{fb}; por lo que, basándose en la figura 3.7, la estructura utilizada será:

$$u(\mathbf{x}) = u_s(s) + k(s)e(\mathbf{x}) = u_s(P(\mathbf{x})) + k(P(\mathbf{x}))e(\mathbf{x}) \tag{3.19}$$

3.2.2. Control por linealización de coordenadas transversales

Para encontrar la entrada de control que asegure la estabilidad orbital en el prototipo *Butterfly Robot* se utilizará el método desarrollado en [Shiriaev, *et al.*, 2005] y [Surov, *et al.*, 2015].

3.2.2.1. Restricción Virtual

El modelo del prototipo *Butterfly Robot* dado por la ec.(3.14), tiene las variables de estado $\mathbf{q} = \begin{bmatrix}\theta & \varphi\end{bmatrix}^T$. Para la generación del movimiento deseado se restringe la evolución de la variable θ mediante una relación con la otra variable de estado φ descrita por la función Θ, [Surov, *et al.*, 2015].

$$\theta = \Theta(\varphi^*)$$

$$\Theta(\varphi^*) = \varphi^* + arctan\left(\frac{a\sin(2\varphi^* - \pi)\left[\Pi(\varphi^*)\vec{\tau}\right]_1 - \left[\Pi(\varphi^*)\vec{\tau}\right]_2}{a\sin(2\varphi^* - \pi)\left[\Pi(\varphi^*)\vec{\tau}\right]_2 + \left[\Pi(\varphi^*)\vec{\tau}\right]_1}\right) \tag{3.20}$$

Donde $[\cdot]_i$ significa la $i-$*ésima* componente del vector, y φ^* es la solución a la ecuación del generador de movimientos para un sistema con un grado de libertad subactuado, dada por la ec.(3.21).

$$\alpha(\varphi)\ddot{\varphi} + \beta(\varphi)\dot{\varphi}^2 + \gamma(\varphi) = 0 \tag{3.21}$$

Recordando las definiciones de $\alpha(\cdot)$, $\beta(\cdot)$ y $\gamma(\cdot)$; y, aplicándolas en el prototipo, se tiene:

$$\alpha(\varphi) = ms'\left(\left(\vec{\rho}\times\vec{\tau} - \frac{J_b}{mR}\right)\Theta' + \left(1 + \frac{J_b}{mR^2}\right)s'\right), \tag{3.22}$$

$$\beta(\varphi) = ms'\left(\left(\vec{\rho}\times\vec{\tau} - \frac{J_b}{mR}\right)\Theta'' - \vec{\rho}\cdot\vec{\tau}\Theta'^2 + \left(1 + \frac{J_b}{mR^2}\right)s''\right), \tag{3.23}$$

$$\gamma(\varphi) = ms'g\begin{bmatrix}0 & 1\end{bmatrix}\mathbf{\Pi}(\varphi)\vec{\tau}. \tag{3.24}$$

Con $\Theta' = \frac{d\Theta}{d\varphi}$ y $\Theta'' = \frac{d^2\Theta}{d\varphi^2}$. La restricción propuesta en la ec.(3.20), permite que las variables de estado sigan la órbita de la mariposa (ec.3.2). Donde el único parámetro desconocido es a, el cual se puede entender como un factor de *suavidad*, pues conforme el factor a va disminuyendo, la suavidad de Θ aumenta; en la figura 3.8 se muestra el cambio de suavidad conforme disminuye el factor a, luego del valor $a = -0.03$ la gráfica no presenta grandes cambios en suavidad de forma que el valor más apto para este factor es ese mismo de acuerdo con [Surov, *et al.*, 2015].

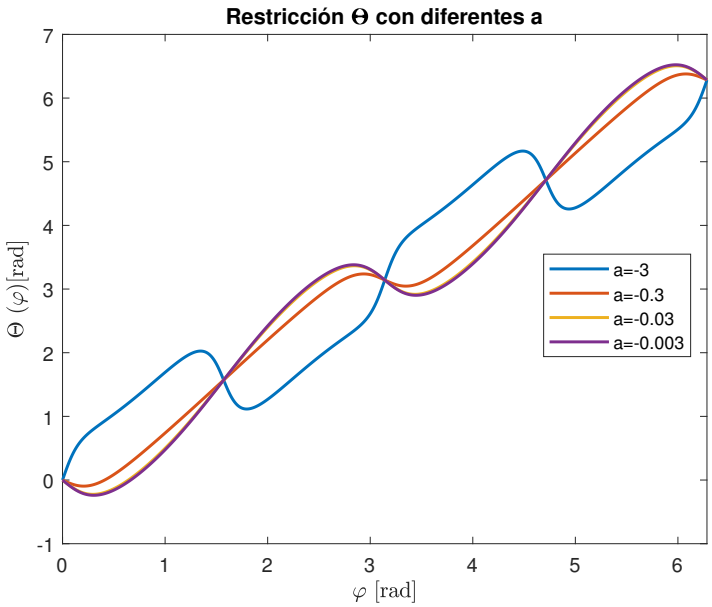


Figura 3.8 Restricción $\Theta(\varphi)$ con diferentes valores de a.

Otra deducción importante viene de recapitular que el sistema estudiado tiene como objetivo seguir una trayectoria **periódica**, por lo que este comportamiento periódico se repetirá en las funciones matemáticas utilizadas para el diseño del controlador. En la figura 3.9 se muestra como precisamente la restricción Θ tiene peridiocidad cada π radianes.

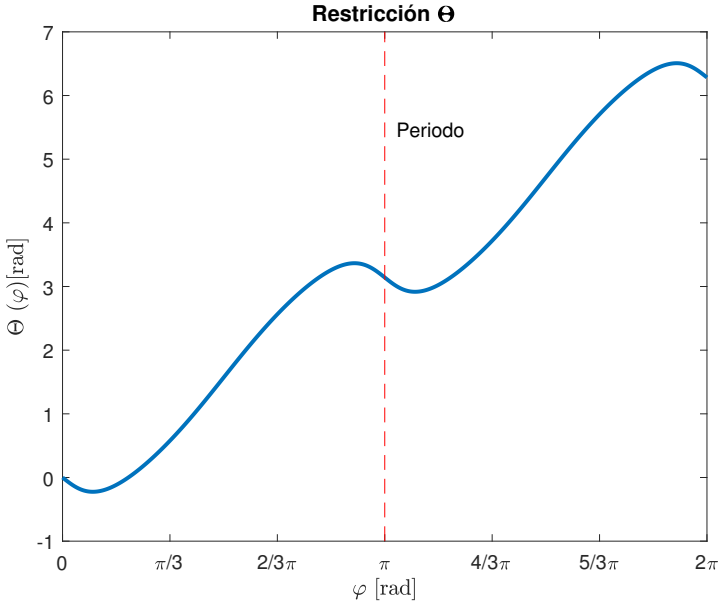


Figura 3.9 Periodo de la restricción $\Theta(\varphi)$.

Una vez propuesta la restricción virtual $\Theta(\varphi^*)$, es momento de definir las coordenadas transversales con las que se diseña el controlador.

3.2.2.2. Coordenadas Transversales

Según la teoría revisitada en la sección 2.2.5.1 es posible elegir un conjunto de coordenadas transversales para evitar el uso del operador de proyección propuesto en el esquema de control de la figura 3.7. En [Shiriaev, *et al.*, 2010] se propone un set de coordenadas transversales estándar:

$$\begin{aligned} y &= \theta - \Theta(\varphi^*), \\ dy &= \dot{\theta} - \Theta'(\varphi^*)\dot{\varphi}^*, \\ I &= \dot{\varphi}^2 - \psi(\varphi_0, \varphi^*)\left[\dot{\varphi_0}^2 - \int_{\varphi_0}^{\varphi^*} \psi(v, \varphi_0)\frac{2\gamma(v)}{\alpha(v)}dv\right] \\ &= \dot{\varphi}^2 - \dot{\varphi^*}^2. \end{aligned}$$

Con $\psi(\varphi_0, \varphi^*) = e^{-2\int_{\varphi_0}^{\varphi^*} \frac{\beta(v)}{\alpha(v)}dv}$. Las coordenadas transversales son funciones de error para la variable de estado, su derivada y la integral de movimiento lo cual asegura el seguimiento de la órbita deseada; esto es natural, ya que, como se estudió en la sección 2.2 es necesaria una función de error para implementar un controlador por retroalimentación de estados.

Hay que poner especial atención en la definición de la coordenada I; pues, al tratarse de una órbita periódica y por las características físicas del sistema, se tiene un fenómeno de simetría en la integral de movimiento, i.e. $I(\varphi^*, \dot{\varphi^*}) = I(\varphi^*, -\dot{\varphi^*})$. Este hecho se muestra en la figura 3.10.

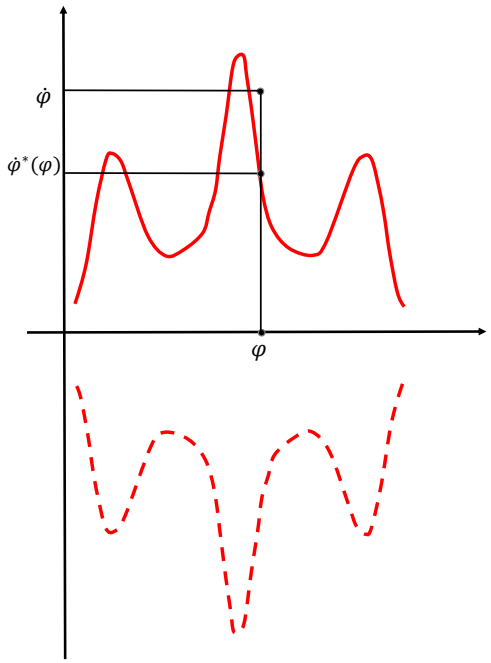


Figura 3.10 Efecto de simetría en la integral de movimiento para el *Butterfly Robot*.

Para evitar la duplicidad de soluciones debida a la connotación cuadrática que tiene la última coordenada transversal (correspondiente a la integral de movimiento), se debe prescindir de ella. Aprovechando la restricción virtual que relaciona θ con φ, convertimos la última coordenada transversal

en una función de error entre la variable medida $\dot{\varphi}$ y la deseada $\dot{\varphi}^*$, esta coordenada será llamada z, es decir,

$$z = \dot{\varphi} - \dot{\varphi}^*.$$

Considerando $\dot{\varphi} = \frac{d\varphi}{dt}$ y $\dot{\varphi}^* = \frac{d\varphi^*}{dt}$.

De esta manera, finalmente se obtiene el vector de coordenadas transversales ξ para el sistema *Butterfly Robot*:

$$\xi = \begin{bmatrix} y \\ dy \\ z \end{bmatrix} = \begin{bmatrix} \theta - \Theta(\varphi^*) \\ \dot{\theta} - \Theta'(\varphi^*)\dot{\varphi^*} \\ \varphi - \varphi^* \end{bmatrix} \tag{3.25}$$

3.2.2.3. Entrada de Control para el LTVs

La elección de las coordenadas transversales dadas por la ec.(3.25), y la restricción virtual de la ec.(3.20), permiten encontrar un sistema lineal variante en el tiempo (LTVs), de la forma

$$\dot{\xi} = \mathbf{A}(t)\xi + \mathbf{B}(t)w. \tag{3.26}$$

Las matrices $A(t)$ y $B(t)$ son dependientes del tiempo, y para el sistema en particular son descritas de la siguiente manera, [Surov, *et al.*, 2015]

$$\mathbf{A}(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ \frac{g_y(\varphi^*(t))}{\alpha(\varphi^*(t))} & \frac{g_{\dot{y}}(\varphi^*(t))}{\alpha(\varphi^*(t))} & \frac{\gamma(\varphi^*(t)) - \beta(\varphi^*(t))\dot{\varphi^*}^2}{\alpha(\varphi^*(t))\dot{\varphi^*}} \end{bmatrix}, \tag{3.27}$$

$$\mathbf{B}(t) = \begin{bmatrix} 0 \\ 1 \\ \frac{g_v(\varphi^*(t))}{\alpha(\varphi^*(t))} \end{bmatrix}. \tag{3.28}$$

Donde los términos g_y, $g_{\dot{y}}$ y g_v son

$$\begin{aligned} g_y &= -mgs' \left(\cos\left(\frac{y}{2} + \Theta\right), -\sin\left(\frac{y}{2} + \Theta\right) \right) \vec{\tau}\, \mathrm{sinc}\left(\frac{y}{2}\right), \\ g_{\dot{y}} &= m\vec{\tau}^T \vec{\rho} s'(\dot{y} + 2\Theta'\dot{\varphi}), \\ g_v &= -ms' \left(\vec{p} \times \vec{\tau} - \frac{J_b}{mR} \right). \end{aligned}$$

Si se presta atención en las matrices de la ec.(3.27) y la ec.(3.28), notamos que los factores están todos evaluados en la trayectoria deseada para φ, es decir, la solución de la ecuación de movimientos (ec.3.21), φ^*. Esto es importante, ya que, el LTVs corresponde a la sección de estabilización orbital estudiada en 2.2.5.1, por lo cual se entiende que el sistema ya fue llevado a la órbita de la mariposa y ahora solo se quiere asegurar la estabilidad en ella.

Considerando que el sistema de la ec.(3.26) actúa sobre las coordenadas transversales ξ, se puede deducir que la entrada de control w asegura la convergencia del sistema *Butterfly Robot* con la órbita deseada. Esta ecuación puede presentar distintas metodologías para su resolución; sin embargo, como se estudió en la sección 2.2.4.2, se utilizará la noción del control óptimo LQR que, para sistemas como este, tiene como solución la entrada de control óptima w,

$$w(t,\xi) = -\mathbf{R}^{-1}(t)\mathbf{B}(t)\mathbf{P}(t)\xi. \tag{3.29}$$

Donde $\mathbf{P}(t)$ es la solución positiva definida de la ecuación diferencial de Ricatti

$$\dot{\mathbf{P}}(t) + \mathbf{A}^T(t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{A}(t) + \mathbf{Q}(t) = \mathbf{P}(t)\mathbf{B}(t)\mathbf{R}^{-1}(t)\mathbf{B}^T(t)\mathbf{P}(t). \tag{3.30}$$

De la cual, las matrices $Q(t) \in \mathbb{R}^{3\times3}$ y $R(t) \in \mathbb{R}^{3\times3}$ son matrices propuestas a partir de una función de costo deseada, para asegurar estabilidad deben cumplir ser positivas definidas o positivas semidefinas, respectivamente, i.e. $Q \geq 0$ y $R > 0$, [Surov, *et al.*, 2015].

Es muy importante la consideración periódica de la órbita, pues provoca que la ec.(3.30) tenga soluciones periódicas para cada coordenada transversal. Resolviendo el LTVs se encuentran las siguientes ganancias para un ciclo de π [rad].

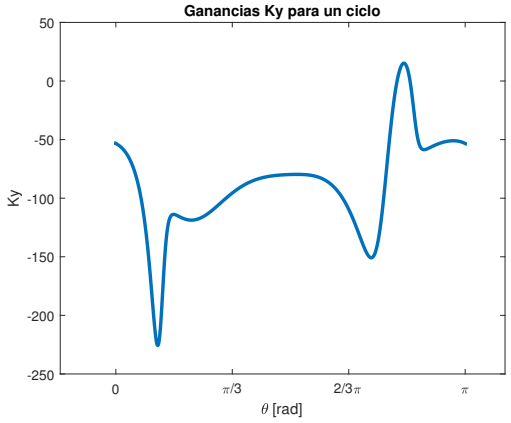


Figura 3.11 Ganancias Coordenada K_y.

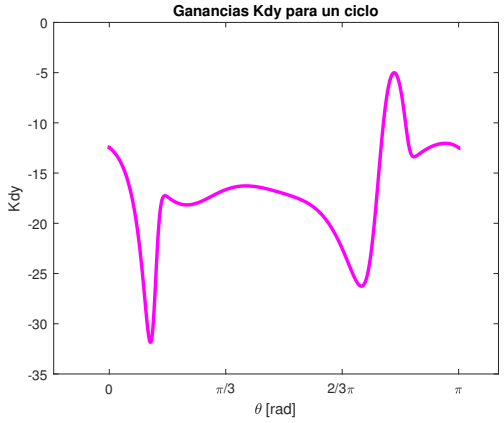


Figura 3.12 Ganancias Coordenada K_{dy}.

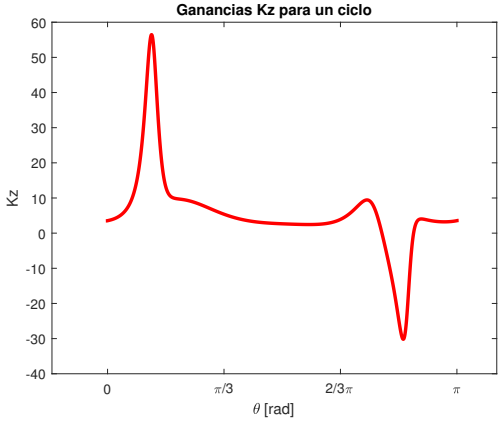


Figura 3.13 Ganancias Coordenada K_z.

De esta manera se puede concluir que la ec.(3.29) es la entrada de control de *feedback* que asegura la convergencia de las trayectorias del sistema a la órbita deseada.

3.2.2.4. Entrada de Control Nominal

Una vez obtenida la entrada de control para el sistema LTVs de la ec.(3.26), es necesario encontrar la entrada de control nominal que permita parametrizar al sistema con respecto a la restricción virtual establecida y las coordenadas transversales deseadas.

Partimos del modelo del sistema dado por la ec.(3.14), el cual requiere una transformación que le permita seguir el movimiento dado por la restricción de la ec.(3.20); para ello se requiere de las matrices de transformación $\mathbf{L}(\varphi^*)$ y $\mathbf{N}(\varphi^*, \dot{\varphi}^*)$. Para el caso del *Butterfly Robot*, las matrices están en función de la restricción $\Theta(\varphi^*)$ de la siguiente manera:

$$\begin{aligned} \mathbf{L}(\varphi^*) &= \begin{bmatrix} 1 & \Theta' \\ 0 & 1 \end{bmatrix} \\ \mathbf{N}(\varphi^*, \dot{\varphi^*}) &= \begin{bmatrix} \Theta'' \dot{\varphi^*}^2 \\ 0 \end{bmatrix} \end{aligned} \tag{3.31}$$

Finalmente considerando las matrices anteriores y la entrada de control de *feedback* (ec.3.29), la entrada de control que permite la generación y estabilización de la órbita de la mariposa es, [Surov*, et al.*, 2015]:

$$u = \frac{w + \left[\mathbf{L}^{-1}\left(\mathbf{N} + \mathbf{M^{-1}CL}\begin{pmatrix}\dot{y} \\ \dot{\varphi}\end{pmatrix} + \mathbf{M^{-1}G}\right)\right]_1}{\left[\mathbf{L^{-1}M^{-1}}\right]_{1,1}}. \tag{3.32}$$

Donde $[\cdot]_i$ significa la $i-$ésima componente del vector, y $[\cdot]_{i,j}$ es la componente de la matriz con columna $i-$ésima y fila $j-$ésima.

Dentro de la ec.(3.32) vale la pena rescatar ciertos aspectos. Como se mencionó anteriormente, las matrices $\mathbf{L}(\varphi^*)$ y $\mathbf{N}(\varphi^*, \dot{\varphi}^*)$ ayudan a parametrizar el sistema con respecto a la restricción deseada (entrada nominal u_{nom}), por su parte, la entrada de control w asegura la convergencia con la órbita dada (entrada de *feedback* u_{fb}), es por ello que esta última entrada debe estar normalizada por el factor de transformación $\left[\mathbf{L^{-1}M^{-1}}\right]_{1,1}$, para poder actuar sobre el sistema ya parametrizado. Por otra parte, se utilizan únicamente las primeras componentes del vector ($[\mathbf{a}]_1$) y la matriz resultante ($[\mathbf{b}]_{1,1}$)[1], ya que esta acción de control sólo actúa de manera directa sobre la primer variable de estado θ.

3.3. Implementación computacional

Para la implementación en el prototipo construido por *Robotikum* son necesarias distintas aplicaciones ejecutandose y comunicandose a la par mediante el protocolo TCP. Estas aplicaciones fueron

[1]Considerando $\mathbf{a} = \left[\mathbf{L}^{-1}\left(\mathbf{N} + \mathbf{M^{-1}CL}\begin{pmatrix}\dot{y} \\ \dot{\varphi}\end{pmatrix} + \mathbf{M^{-1}G}\right)\right]$ y $\mathbf{b} = \left[\mathbf{L^{-1}M^{-1}}\right]$.

desarrolladas por [Surov, *et al.*, 2015] en los lenguajes *C++* y *Python* dentro del entorno de *Linux* (en su distribución *Ubuntu*) y se encuentran dentro de las dos computadoras utilizadas en el *Butterfly Robot*: el Intel Nuc y el *Beaglebone Green*.

En el Intel Nuc se ubican las primeras dos aplicaciones, correspondientes al procesamiento de visión y al cálculo de la entrada de control:

- **Aplicación de visión (*CameraApp - Ball Tracker*)**.

  Para la realización de la aplicación de visión del *Butterfly Robot*, son necesarias dos partes: la identificación de la pelota y platos, así como la estimación de su posición y velocidad. Para ello se utilizan las bibliotecas correspondientes a *Open CV* y a *Basler*, las cuales se integran en *C++* para generar patrones de reconocimiento de la cámara.

  El entrenamiento inicia distinguiendo zonas oscuras de zonas luminosas o con color, con el fin de poder identificar tanto a los platos como a la pelota; es por esto que en primera instancia se utilizan patrones correspondientes al tablero de un ajedrez (Figura 3.14).

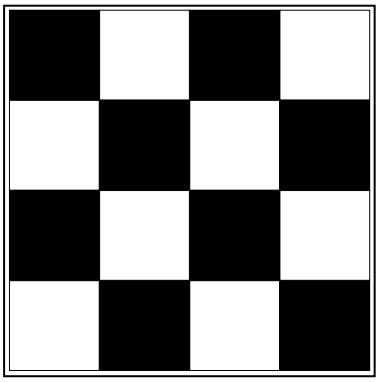
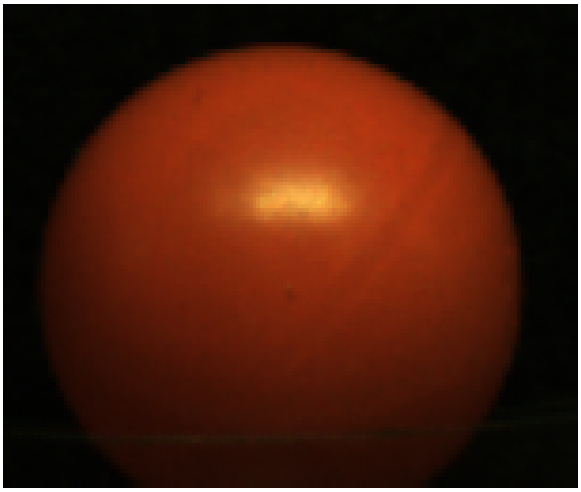


Figura 3.14 Patrón de Ajedrez.

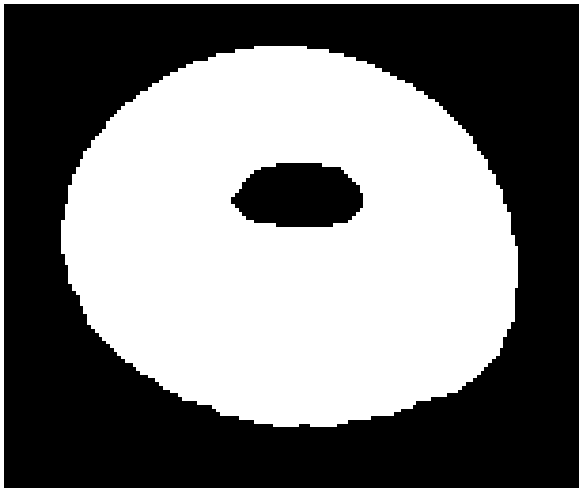
  Lo anterior explica la razón por la que en el prototipo, la cámara apunta a un fondo negro que es antecedido por el sistema de la mariposa. Con el inicio del entrenamiento es posible identificar los bordes del plato en contraste con el fondo.

  Es necesario diferenciar entre los platos y la pelota para obtener la coordenada φ. Por consiguiente, se realiza un reconocimiento del color específico de la pelota, tomando varias muestras fotográficas de la misma y midiendo la variaciones que causan en los pixeles interpretados por la computadora, estos valores se utilizaron para generar un filtro que separa el color de la pelota de los zonas oscuras y las zonas muy luminosas (las cuales ayudan a distinguir a los platos), y así, identificar a la pelota y generar un espectro de su área de manera suave, como se muestra en la figura 3.15.

  El filtro mostrado ayuda a identificar cualquier tipo de objeto que se interponga entre la visión y la pelota, y al color que tenga lo identifica como una sombra más, esto ayuda a que el sistema no pueda ser “engañado” con objetos distintos a la pelota, pero lo obliga a



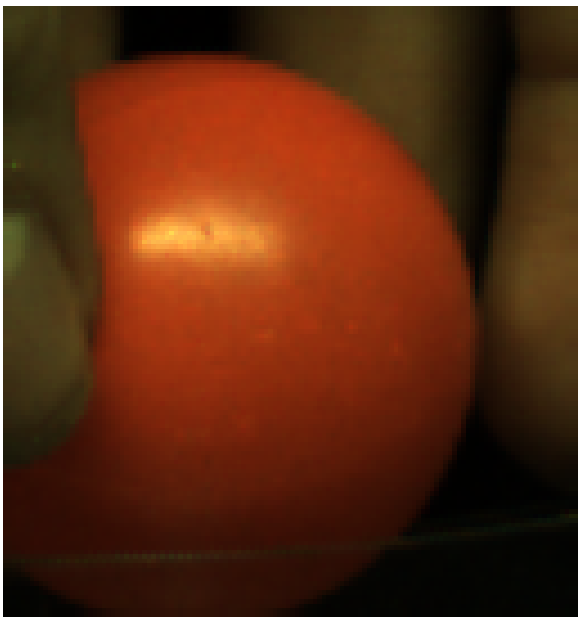
(a) Pelota Real.



(b) Imagen Filtrada.

Figura 3.15 Identificación de la pelota a través del sistema de visión.

tener que ocupar especificamente su color (Figura 3.16). Este ajuste de color puede cambiarse utilizando los programas de *Python* llamados `color_filter.py`, `intrinsic_calibration.py` y `extrinsic_calibration.py`, encontrados en la carpeta `dev\CameraApp\Scripts` dentro del Nuc.



(a) Pelota Real sujetada.

(b) Imagen Filtrada de la pelota sujetada.

Figura 3.16 Identificación otros objetos además de la pelota a través del sistema de visión.

Finalmente, los programas que permiten la identificación de objetos y posición son compilados mediante el comando `cmake -H. -Bbuild.`[2], lo cual genera una nueva carpeta de construcción

[2]El archivo *Cmake* se encuentra en el apéndice A.1.

(*build*), en la cual mediante el comando `make` se genera un archivo ejecutable llamado `cam`. Este archivo permite cuatro configuraciones distintas:

1. `Autocalib.json`: Permite la calibración automática de la cámara en función de los archivos de configuración de *Python*. Para ejecutar esta configuración desde la terminal se utiliza el archivo `./autocalib.sh` o el comando `sudo chrt -r 50 ./build/cam -c configs/autocalib.json`.
2. `Logger.json`: Toma fotos al sistema y las guarda en la carpeta `dev\CameraApp\dump`. Para ejecutar esta configuración desde la terminal se utiliza el archivo `./logger.sh` o el comando `sudo chrt -r 50 ./build/cam -c configs/logger.json`.
3. `Recorder.json`: A partir de las fotos generadas, calcula e imprime la posición de la pelota y de los platos. Para ejecutar esta configuración desde la terminal se utiliza el archivo `./recorder.sh` o el comando `sudo chrt -r 50 ./build/cam -c configs/recorder.json`.
4. `Sensor.json`: Configura a la cámara como un sensor de visión, el cual espera una conexión con el sistema de control dentro del mismo Nuc. Esta configuración es la que se utiliza para ejecutar el entorno visual del sistema *Butterfly Robot*. Para ejecutar esta configuración desde la terminal se utiliza el archivo `./run.sh` o el comando `sudo chrt -r 50 ./build/cam -c configs/sensor.json`.

Como se mencionó anteriormente, este sistema es únicamente para desarrollar la visión y medición de la variable φ, por lo cual se debe comunicar con los siguientes entes (cálculo de control y control de servomotor), que es posible gracias a la comunicación TCP.

De manera breve, esta aplicación toma fotografías constantes del sistema y las procesa en tiempo real para obtener los parámetros deseados. Cabe destacar que la cámara se comunica con el Intel Nuc a través de comunicación USB 3.0, consiguiendo un muestreo de 1.2 [ms].

- **Aplicación para el cálculo de la entrada de control (*nuc_controller - control_system*).**

  La segunda aplicación que se ejecuta dentro del Intel Nuc se trata del algoritmo de control que hace el procesamiento de las variables de estado $q = \begin{bmatrix}\theta & \varphi\end{bmatrix}^T$, e instaura la entrada de control descrita por la ec.(3.32) al prototipo. El programa generado en *C++* presenta la interconexión de distintos códigos en un archivo *main* llamado `buttefly.cpp` que permite, finalmente, la implementación real del controlador.

  Los códigos principales son: `butterfly.cpp` y `overturn_control.cpp`[3]. El primer código mencionado es el paso intermedio entre la aplicación de visión implementada con el uso de la cámara (*CameraApp - Ball Tracker*) y la aplicación para generar la entrada de control con el servomotor *Maxon* (*Servo_controller*), ya que recibe los datos provenientes de la cámara y los transforma en una entrada de control que comunica al *beaglebone* para finalmente accionar el servomotor[4].

[3] Ambos códigos son mostrados en su totalidad dentro del apéndice A.2 y A.3, respectivamente.

[4] La tercer aplicación será estudiada más a detalle en la página 65; sin embargo, es importante mencionar que mediante ella se obtiene la variable de estado faltante θ.

La implementación se divide en tres bloques de código principales:

1. **Inicialización del Hardware**: Se acciona con la función `void Butterfly::init()`, y permite asegurar que existe comunicación (vía TCP) entre las 3 aplicaciones para la implementación computacional.

```
1  void Butterfly::init(jsonxx::Object const& jscfg) //Inicializa el hardware
2  {
3       info_msg("initializing hardware..");
4
5       auto const& jsctrl = json_get<jsonxx::Object>(jscfg, "controller");
6       m_sync_delay = json_get<int64_t>(jsctrl, "cam_delay_usec", 0, 1e+5); //Procesa
    un antidelay de las senales de la camara
7       m_delay_theta.init(m_sync_delay);
8       m_delay_dtheta.init(m_sync_delay);
9
10      m_servo = ServoIfc::capture_instance(); //Indica comunicacion con la camara
11      m_servo->init(jscfg);
12
13      m_camera = Camera::capture_instance(); //Indica comunicacion con el servomotor
14      m_camera->init(jscfg);
15
16      info_msg("done");
17 }
18
```

Código 3.1 Función `void Butterfly::init()`

Notamos que en el código 3.1 se cuenta con una sección que se encarga de compensar los efectos de *delay*, naturales por la conexión USB; además, se presenta la validación de la comunicación entre aplicaciones.

2. **Medición de parámetros**: Se acciona con la función `void Butterfly::measure()`, y mide o reconstruye las variables de estado: θ del encoder en el servomotor, y φ de la aplicación de la cámara, además de darles un tratamiento que contrarresta el *delay* de comunicación.

```
1  void Butterfly::measure() //Medicion y acondicionamiento de las variables de estado
2  {
3       int64_t t_servo;
4       int status = m_servo->get_state(t_servo, m_theta, m_dtheta, true); //Toma theta
    de la aplicacion del servo
5       if (status < 0)
6       throw runtime_error("servo disconnected");
7
8       double theta_delayed = m_delay_theta.process(t_servo, m_theta);
9       double dtheta_delayed = m_delay_dtheta.process(t_servo, m_dtheta);
10
11      int64_t t_cam;
12      status = m_camera->get(t_cam, m_x, m_y); //Toma x, y de la aplicacion de la
    camara
13
14      switch (status)
15      {
16           case 1:
17           {
18                m_vx = m_diff_x.process(t_cam, m_x);
19                m_vy = m_diff_y.process(t_cam, m_y);
20
21                double alpha = atan2(m_x, m_y);
22                double dalpha = (m_y * m_vx - m_x * m_vy) / (m_x * m_x + m_y * m_y);
23
24                m_phi = theta_delayed + alpha; //Calcula varphi
25                m_dphi = dtheta_delayed + dalpha;
26                m_ball_found = true;
27                break;
28           }
29           case 0:
30           {
31                break;
```

```
                        }
                        default:
                        {
                             if (m_ball_found) //Identifica que la pelota no es reconocida por el
    sistema de vision
                             info_msg("ball was lost");
                             m_ball_found = false;
                             break;
                        }
                   }
              }

```

Código 3.2 Función `void Butterfly::measure()`

El código 3.2 realiza la medición de las variables reales del sistema. Cabe resaltar que de la aplicación de la cámara se toma la posición (x, y), para luego calcular la variable φ en función de la variable θ considerando el *delay* mencionado anteriormente. También, es importante considerar que **si la pelota no es encontrada, el sistema se detiene y se desactiva automáticamente**.

3. **Función Main**: Se acciona con la función `void Butterfly::start()`, en esta sección se hace el cálculo de la entrada de control mandando a llamar a la función `set_torque()` propia del código `overturn_control.cpp`.

```
1    void Butterfly::start(callback_t const& cb) //Programa Main para el Butterfly Robot
2    {
3         if (!m_camera || !m_servo) //Confirma la interconexion de aplicaciones
4         throw runtime_error("Butterfly not initialized yet");
5
6         m_camera->start();
7         m_servo->start();
8
9         int status;
10        int64_t t, t0;
11        t0 = get_time_usec();
12        //Codigo agregado por mi para obtener datos practicos
13        ofstream filetime("/home/butterfly/dev/nuc_controller/out/timeOut.txt");//New
14        ofstream filetorque("/home/butterfly/dev/nuc_controller/out/torqueOut.txt");//
    New
15        ofstream filetheta("/home/butterfly/dev/nuc_controller/out/thetaOut.txt");//New
16        ofstream filephi("/home/butterfly/dev/nuc_controller/out/phiOut.txt");//New
17        ofstream filedtheta("/home/butterfly/dev/nuc_controller/out/dthetaOut.txt");//
    New
18        ofstream filedphi("/home/butterfly/dev/nuc_controller/out/dphiOut.txt");//New
19        ofstream filex("/home/butterfly/dev/nuc_controller/out/xOut.txt");//New
20        ofstream filey("/home/butterfly/dev/nuc_controller/out/yOut.txt");//New
21
22        while (!m_stop) //Si se reconoce la pelota se mantiene en el while
23        {
24             t = get_time_usec();
25             measure();
26
27             BflySignals signals;
28             get_signals(t - t0, signals);
29             status = cb(signals);
30             if (!status)
31             m_stop = true;
32
33             m_servo->set_torque(signals.torque); //Manda al servomotor la senal de
    control
34             filetime << signals.t << ",";//New
35             filetorque << signals.torque << ",";//New
36             filetheta << signals.theta << ",";//New
37             filephi << signals.phi << ",";//New
38             filedtheta << signals.dtheta << ",";//New
39             filedphi << signals.dphi << ",";//New
40             filex << signals.x << ",";//New
41             filey << signals.y << ",";//New
42
43        }
44
```

```
45                      m_servo->stop(); //Detiene el proceso del servo
46                      m_camera->stop(); //Detiene el proceso de la camara
47                      //Se cierran los archivos de escritura de datos
48                      filetime.close();//New
49                      filetorque.close();//New
50                      filetheta.close();//New
51                      filephi.close();//New
52                      filedtheta.close();//New
53                      filedphi.close();//New
54                      filex.close();//New
55                      filey.close();//New
56
57
58                      info_msg("stopped");
59              }
60
```

Código 3.3 Función `void Butterfly::start()`

Como se muestra en el código 3.3, se agregaron lineas que permitieron adquirir datos prácticos del sistema *Butterfly Robot*; y, así como se muestra al final de cada una de ellas, todas las lineas modificadas y agregadas en este trabajo de tesis se identificarán con el comentario `//New`.

La función *main* del código `butterfly.cpp`, utiliza una dependencia de `overturn_control.cpp`, en este último se instaura la entrada de control dada por la ec.(3.32). La estructura del programa `overturn_control.cpp` contiene lo siguiente:

1. **Splines**: En la sección 3.2.2.3 se analizó que, debido a que la curva de la mariposa es una órbita periódica, las ganancias k_y, k_{dy} y k_z que son solución de la ec.(3.30) son también periódicas en un ciclo de π [rad] y están descritas por los valores presentados en las figuras 3.11, 3.12 y 3.13; entonces, en lugar de resolver en tiempo real la ecuación diferencial de Ricatti, es conveniente presentarlas mediante un ente llamado *spline*[5], en el cual se ingresan los valores de salida de cada ganancia con respecto a cada uno de los ángulos θ medidos.

   Los *splines* igualmente son utilizados en aquellas variables que tienen comportamiento periódico, tales como la restricción virtual $\Theta(\varphi)$ y el vector de posición $\vec{\rho}$.

2. **Matrices y Vectores**: Las matrices y los vectores fueron modificados con el fin de aligerar el procesamiento computacional, pues en la versión desarrollada por [Surov, *et al.*, 2015], las matrices contienen funciones senoidales que pueden ser simplificadas matemáticamente (ejemplo para el vector de fuerzas potenciales **G** en el código 3.4)

```
1               inline Vec2 sub_G(double const& theta, double const& phi)
2               {
3                    auto rho_val = spline_rho(phi);
4                    auto rho_d1_val = spline_rho(phi, 1);
5                    //return Vec2(0.02943*rho_val*sin(phi)*cos(theta) - 0.02943*rho_val*sin(theta)*
    cos(phi), 0.02943*((rho_d1_val*sin(phi) + rho_val*cos(phi))*sin(theta)/sqrt(pow(rho_d1_val*sin(
    phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)) + (rho_d1_val*cos(
    phi) - rho_val*sin(phi))*cos(theta)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(
    rho_d1_val*cos(phi) - rho_val*sin(phi), 2)))*sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2)
     + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2))); \\Matriz dada por PhD. Maksim Surov
6                    return Vec2(0.02943*rho_val*sin(phi)*cos(theta) - 0.02943*rho_val*sin(theta)*cos
    (phi), 0.02943*((rho_d1_val*sin(phi) + rho_val*cos(phi))*sin(theta) + (rho_d1_val*cos(phi) -
    rho_val*sin(phi))*cos(theta))); //New (Simplificacion)
7               }
8
```

[5]Un *spline* es una función muy utilizada computacionalmente, que permite asignar un valor de salida a cada variable de entrada. Los *splines* se conforman de dos arreglos de números, uno que funge como variable de entrada y el otro como salida. Según el muestreo dado para la variable de entrada, el *spline* puede ser una función *suave*.

Código 3.4 Simplificación del vector de fuerzas potenciales **G**.

3. **Función `double get_torque()`**: Dentro de ella se hace el cálculo de la entrada de control, utilizando las variables de estado medidas (obtenidas en las aplicaciones de visión y servomotor) y las deseadas (obtenidas de los *splines*), con lo que finalmente se obtiene el siguiente código:

```
            double get_torque(double theta, double phi, double dtheta, double dphi)
            {
                auto n = int(floor(phi / _PI));
                phi -= _PI * n;
                theta -= _PI * n;
                //Asigna las variables deseadas con los splines (todos dependientes de varphi -
    phi-)
                auto dphi_s = spline_dphi(phi);
                auto theta_s = spline_vc(phi);
                auto vc1 = spline_vc(phi, 1);
                auto vc2 = spline_vc(phi, 2);
                auto kz = spline_kz(phi);
                auto ky = spline_ky(phi);
                auto kdy = spline_kdy(phi);
                auto dtheta_s = vc1 * dphi_s;
                //Calcula las coordenadas transversales
                auto z = dphi - dphi_s;
                auto y = theta - theta_s;
                auto dy = dtheta - vc1 * dphi;
                //Entrada de control w
                auto w = z * kz + y * ky + dy * kdy;
                //Calculo de las matrices y vectores
                Vec2 dq_s(dtheta_s, dphi_s);
                Mat2x2 invL(1, -vc1, 0, 1);
                auto M = sub_M(theta, phi);
                auto C = sub_C(theta_s, phi, dtheta_s, dphi_s);
                auto G = sub_G(theta, phi);
                auto invM = inv(M);
                auto K = invL * invM;
                //Adquisicion de datos
                fileky << ky << ",";//New
                filekdy << kdy << ",";//New
                filekz << kz << ",";//New
                fileyco << y << ",";//New
                filedyco << dy << ",";//New
                filezco << z << ",";//New
                filedphis << dphi_s << ",";//New
                filethetas << theta_s << ",";//New
                filedthetas << dtheta_s << ",";//New
                filevc1 << vc1 << ",";//New
                filevc2 << vc2 << ",";//New
                filew << w << ",";//New
                //Entrada de control tau(u)
                auto tau = (w + (K * (G + C*dq_s)).at(0,0) + vc2 * pow(dphi_s, 2)) / K.at(0,0);
                return tau;

            }

```

Código 3.5 Función **`double get_torque()`**

Así como en el caso de la función **`void Butterfly::start()`**, dentro del código 3.5 se utilizaron funciones de adquisición de datos experimentales.

Una vez analizados estos dos códigos y ejecutando los comandos `cmake -H. -Bbuild.` y `make`, se genera un archivo ejecutable que se conecta a la aplicación de visión. Para correr este último programa desde la terminal de Linux, es necesario posicionarnos en la carpeta `dev\nuc_controller\build`, y usar el comando `sudo ./build/ButterflyController -c configs/controller.json`, o dentro de

la carpeta `dev\nuc_controller` ejecutar el archivo `./run.sh`.

Estas dos primeras aplicaciones (visión y cálculo de la entrada de control), se comunican desde el Intel Nuc hasta el *Beaglebone Green*, donde se encuentra la aplicación correspondiente al controlador del servomotor.

- **Aplicación para controlar el servomotor (*servo_controller*).**

  Para accionar el servomotor *Maxon 590953*, es necesario contar con la información proveniente de la aplicación de cálculo de la entrada de control. Dentro del *Beaglebone* se instala el sistema operativo *Linux* en su distribución *Debian*; se configura la dirección IP `192.168.7.2`, y se alojan los *drivers* para controlar al servomotor.

  Dentro del Intel Nuc se lleva a cabo el proceso de construcción de programas mediante los comandos `cmake -H. -Bbuild.` y `make`, para finalmente obtener los archivos ejecutables que contienen todos los comandos de comunicación.

  En esta aplicación se mide la variable de estado θ mediante el uso de un *encoder* absoluto de alta resolución, y hace la transducción entre la señal de control calculada y el voltaje necesario para generar el par requerido. Es importante remarcar que, con el fin de evitar daños en el motor y en el sistema mecánico general, **el par máximo que se puede obtener es de** $\pm 0.1[Nm]$**, si el cálculo excede este valor, el sistema de control se desactiva automáticamente**.

La comunicación vía TCP permite la conexión entre las tres aplicaciones utilizando sus IPs establecidas. Para la relación entre el Nuc y el *Beaglebone*, se le asignan las IPs `192.168.7.4` y `192.168.7.2`, respectivamente; mientras que para la comunicación entre las dos aplicaciones dentro del Intel Nuc se utiliza la IP `localhost` con dirección `127.0.0.1`, pues la comunicación es interna en la computadora (Figura 3.17).

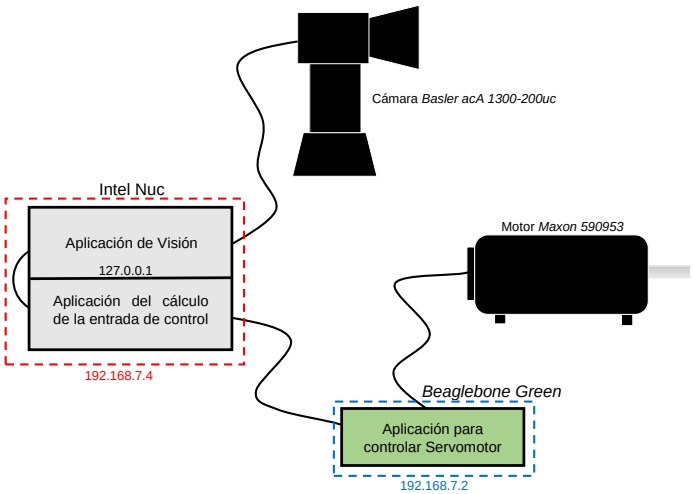


Figura 3.17 Implementación Computacional.

Con la implementación computacional desarrollada, se aplicó la entrada de control de la ec.(3.32) al sistema *Butterfly Robot* descrito por la ec.(3.14).

Capítulo 4

Resultados Experimentales

Considerando el modelo matemático, la entrada de control y la implementación del algoritmo de control por linealización de coordenadas transversales, propuesto por [Shiriaev*, et al.*, 2010] y [Surov*, et al.*, 2015]. Se realizaron pruebas experimentales en el prototipo *Butterfly Robot*, desarrollado por la empresa *Robotikum*, con el fin de evaluar su desempeño.

En el presente capítulo se muestran y discuten los resultados experimentales obtenidos. En la sección 4.1 se presentan los resultados cuando el sistema dinámico no sufre perturbaciones externas (i.e. desplazamiento de la pelota, aceleración o frenado de los platos, etc.); mientras que, en la sección 4.2, se consideran perturbaciones aleatorias que afectan al sistema.

4.1. Escenario 1: Sistema sin presencia de perturbaciones

Dentro de la evaluación de resultados se tomaron distintos escenarios en los que el sistema puede actuar. Para empezar se implementó el controlador por linealización de coordenadas transversales en la planta *Butterfly Robot* sin presentar perturbaciones externas. Los resultados obtenidos se mostrarán a continuación.

4.1.1. Restricción Virtual sin Perturbación

Para poder seguir la órbita que da forma a la mariposa (ec.3.2), se definió en el capitulo 3 la restricción virtual que permite este hito, la cual está descrita por la ec.(3.20). Al realizar la evaluación experimental se consiguió el resultado mostrado en la figura 4.1.

Realizando un análisis de la figura 4.1, notamos una aproximación de la restricción Θ que cumple con el comportamiento periódico; sin embargo, el resultado experimental difiere en la *suavidad* de la curva con respecto al valor nominal, esto tiene una explicación que recae en la implementación computacional; en la sección 3.3, se presentó el uso de una herramienta informática denominada *spline*, esta herramienta sólo está definida en un ciclo de π [rad], por lo cual cuando el ángulo medido se excede de este valor, el *spline* se reinicia como si el ángulo fuera 0 [rad], de aquí que cada vez que se cruza por el periodo se tiene una linea recta en lugar de una curva suave. Es de gran relevancia indicar que en esa zona donde los cambios en la trayectoria ocurren bruscamente, el lazo de control es incapaz de seguir la trayectoria “correctamente”.

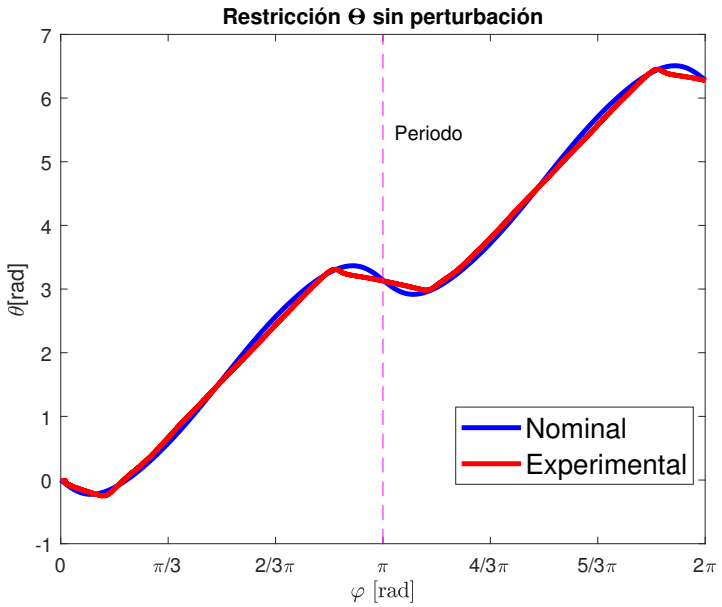


Figura 4.1 Restricción virtual Θ sin perturbación externa.

4.1.2. Seguimiento de órbita sin Perturbación

Según el resultado correspondiente a la restricción virtual, el sistema debe converger con la órbita de manera asintótica. Por las características de la implementación, la pelota ya se encuentra en la órbita, por lo que su comportamiento solo tiene que asegurar que no se salga de ella, es decir, físicamente que no se caiga de los platos que forman a la mariposa. Iniciando con condiciones iniciales nulas, i.e. $\varphi_0 = 0[rad]$ y $\dot{\varphi}_0 = 0[\frac{rad}{s}]$, se encontró el siguiente comportamiento.

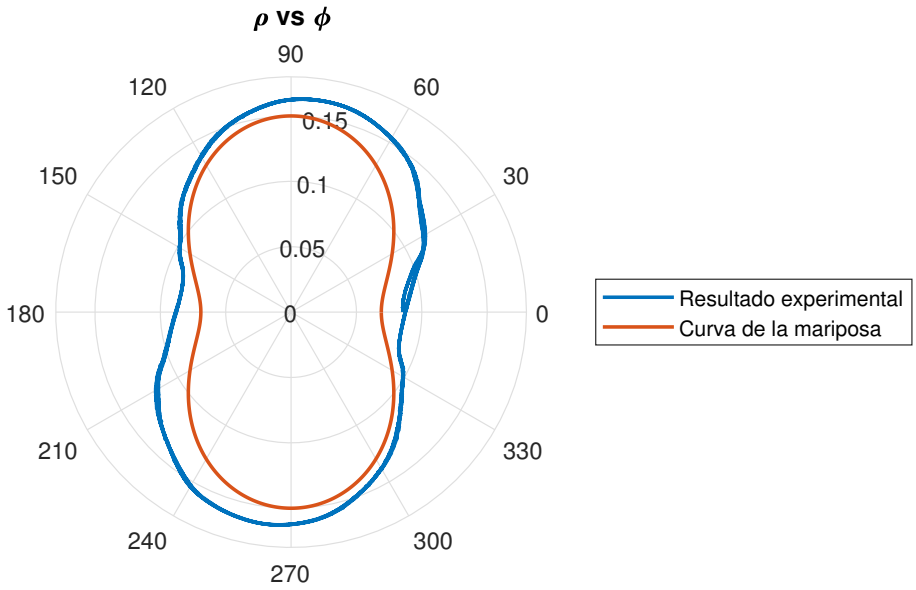


Figura 4.2 Órbita trazada por ρ sin perturbación externa.

En la figura 4.2 se observa un fenómeno de especial interés, pues de la deducción hecha en la sección 3.1.3 se sabe que el vector $\vec{\rho}$ apunta al centro de la pelota, por lo cual un desfase entre la curva

de la mariposa y la curva trazada por la pelota podría considerarse normal. Sin embargo, de ser esa la única razón, el desfase sería equidistante en todo momento, y basándonos en el resultado obtenido, esto no ocurre así. La explicación se encuentra en entender las limitaciones físicas que el sistema en sí mismo tiene, es decir, por las restricciones físicas y computacionales que el *Butterfly Robot* presenta, existen movimientos que no son posibles, aunque la restricción virtual trate de llevarlos a cabo.

Otra consideración por lo cual la curva experimental presenta este comportamiento no equidistante se puede deber a las dinámicas no modeladas, como lo son elasticidades en los cuerpos, fricción entre los mismos, y las dinámicas del motor; así como la velocidad en la que se mueve la pelota, que si bien dentro de este trabajo no se analiza la repercusión que esta tiene en los resultados experimentales, será importante evaluarse como un parámetro en el diseño del controlador para trabajos futuros.

A pesar de todo lo mencionado, el sistema sí cumple con el objetivo de control, que es mantener en una órbita periódica el movimiento de la pelota, logrando así un ciclo límite estable (Problema 3.1.1).

4.1.3. Coordenadas Transversales sin Perturbación

Es momento de evaluar qué ocurre con las funciones de error conocidas como coordenadas transversales. En la sección 3.2, encontramos un set de coordenadas transversales (ec.3.25) que presentaban el error entre las variables medidas (θ, $\dot{\theta}$ y φ) y las variables deseadas (Θ, $\Theta'\dot{\varphi}^*$ y φ^*). Se estudió también que si estas coordenadas se llevan al origen, implica que el sistema se encuentra en la órbita. El resultado experimental es el siguiente (considerando un tiempo de 12[s], en el cual se realizaron 2 ciclos y medio a la curva de la mariposa[1]).

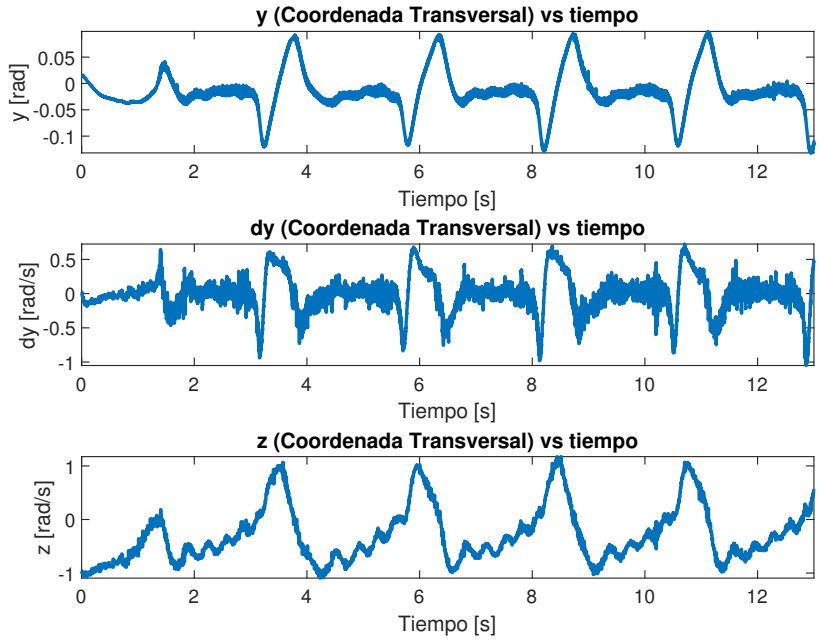


Figura 4.3 Coordenadas transversales ξ sin perturbación externa.

[1]El tiempo fue elegido con el fin de mostrar el comportamiento periódico de las coordenadas transversales.

La figura 4.3 comprueba que las coordenadas transversales no se encuentran estables en **0**, sino que, presentan un comportamiento periódico y acotado en una vecindad muy cercana (menores a $\pm 0.15[rad]$ para la coordenada y, y menores a $\pm 1[\frac{rad}{s}]$ para las coordenadas dy y z). Esta gráfica comprueba la congruencia de nuestro sistema, pues como se demostró en la figura 4.2, el error entre la órbita deseada y la órbita experimental no tiene valores constantes a lo largo de todos los puntos de la curva, pero sí mantiene el comportamiento cíclico propio de la órbita.

4.1.4. Posición (x,y) de la pelota sin Perturbación

Con el fin de demostrar el comportamiento periódico y sin perturbaciones externas se mostrará la evolución temporal de las coordenadas (x, y) correspondientes a la pelota. El resultado experimental arroja lo siguiente (Figura 4.4).

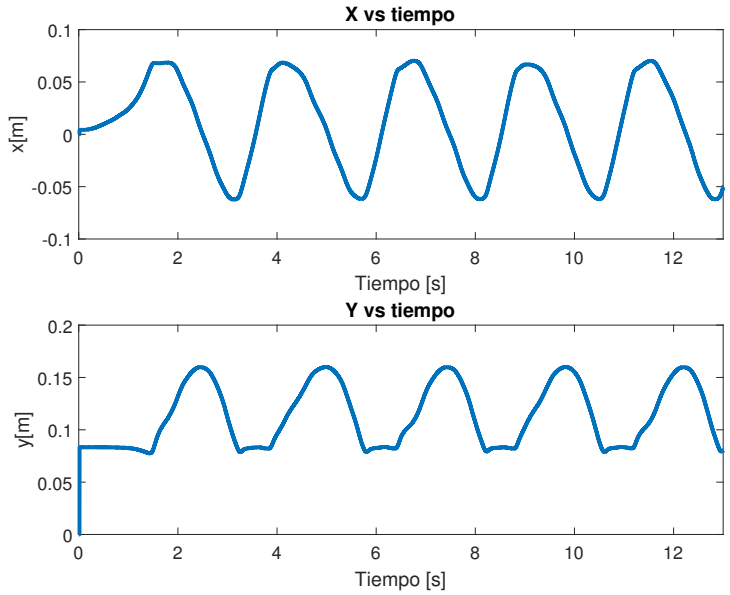


Figura 4.4 Evolución temporal (x,y) de la pelota sin perturbación externa.

La gráfica anterior muestra como la posición (x, y) mantiene una peridiocidad en todo momento cuando no se tienen perturbaciones. Este fenómeno es esperado, pues al tratarse de una curva periódica, las posiciones deben mantener tal comportamiento. La coordenada y describe temporalmente la evolución experimental de la mariposa, donde las crestas de la gráfica representan físicamente cuando la pelota pasa por los lóbulos más amplios de la mariposa, mientras que las partes más constantes en la gráfica representan cuando la pelota se encuentra en la parte más angosta de la mariposa; de esta manera se confirma que la pelota sigue la trayectoria ciclica dada.

4.1.5. Entradas de Control sin Perturbación

Es importante analizar qué sucede con las entradas de control utilizadas. De primera mano, se observa el comportamiento de la entrada de control w (ec.3.29) para el sistema LTVs de las coordenadas transversales dado por la ec.(3.26). El resultado experimental tiene la siguiente forma.

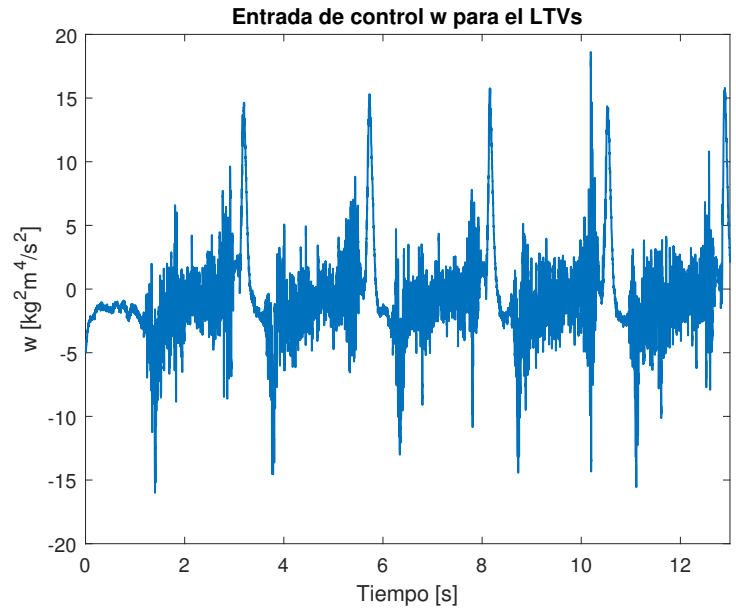


Figura 4.5 Entrada de control w para el sistema LTVs sin perturbación externa.

Dentro de la figura 4.5 se encuentra de nueva cuenta un comportamiento periódico, lo cual se obvia al recordar que las ganancias correspondientes a k_y, k_{dy} y k_z tienen la misma característica. Esta entrada de control por sí sola, tiene magnitud muy alta y no presenta las unidades de par mecánico, lo cual denota la importancia de normalizarla cuando se agrega a la entrada total de control u.

En segunda instancia, analizaremos el comportamiento experimental de la entrada de control total (ec.3.32). La conducta experimental se muestra en la figura 4.6.

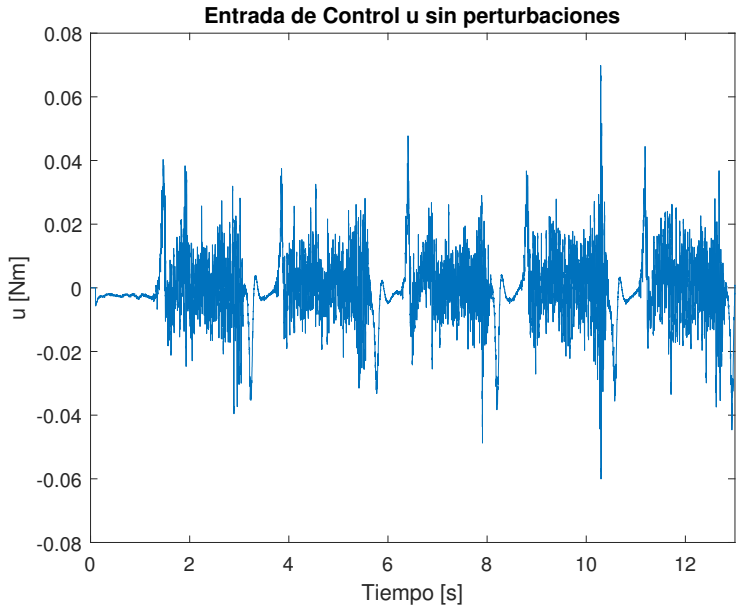
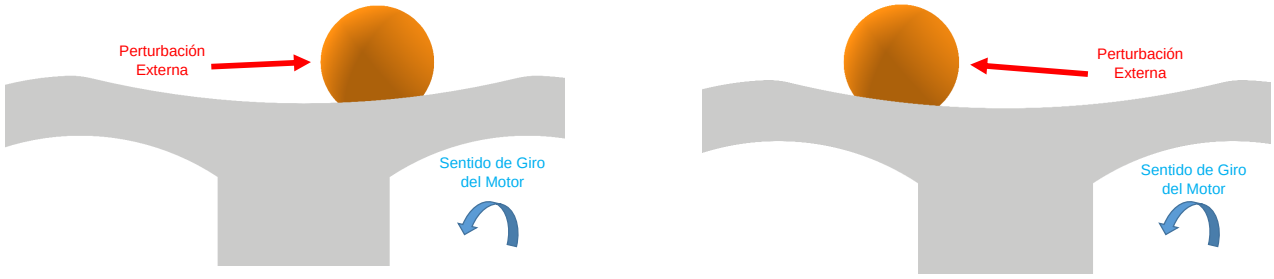


Figura 4.6 Entrada de control u sin perturbación externa.

La entrada de control u también es periódica, y al estar normalizada en función de las restricciones virtuales, entrega valores realizables para el servomotor *Maxon*. En la sección 3.3 se encontró que existe un limitador en la salida del servomotor acotado a $\pm 0.1[Nm]$, en la figura 4.6 se demuestra que todos los valores están por debajo de esa cota, provocando la correcta acción del sistema de control.

4.2. Escenario 2: Sistema con presencia de perturbaciones

Al sistema se le perturbó de dos maneras: en primera instancia empujando la pelota hacía el sentido contrario de giro del motor para acelerar su movimiento (Figura 4.7a), y en segunda aplicando una fuerza que regresara a la pelota hacia el mismo sentido del giro del motor (Figura 4.7b); ambas perturbaciones se realizaron en la zona más estrecha del *Butterfly Robot*.



(a) Perturbación 1. (b) Perturbación 2.

Figura 4.7 Perturbaciones aplicadas al sistema.

4.2.1. Restricción Virtual con Perturbación

La restricción virtual es el ente que permite definir el movimiento de la órbita en el prototipo real. En presencia de perturbaciones se espera que el sistema mantenga la relación entre θ y φ. Los resultados se presentan en la figura 4.8.

Con la respuesta obtenida se comprueba que la relación dada por la función $\Theta(\varphi)$ mantiene la conducta experimental deseada, aunque con unas pequeñas variaciones en los cruces por $n\pi$ [rad] (con $n = 1, 2, 3, \cdots$), debido a las perturbaciones aplicadas.

4.2.2. Seguimiento de Órbita con Perturbación

Siguiendo la lógica que presenta la restricción virtual experimental mostrada en la figura 4.8, se espera que la órbita experimental se mantenga como un ciclo límite estable. Se consideraron las condiciones iniciales $\varphi_0 = 0[rad]$ y $\dot{\varphi}_0 = 0[\frac{rad}{s}]$.

Los resultados expuestos en la figura 4.9 reforzan lo discutido en la sección 4.1.2, pues ocurre el efecto de desfase entre la curva de la mariposa y la curva trazada por la pelota, debido a las mismas limitaciones físicas y computacionales de la implementación, además de dinámicas no modeladas.

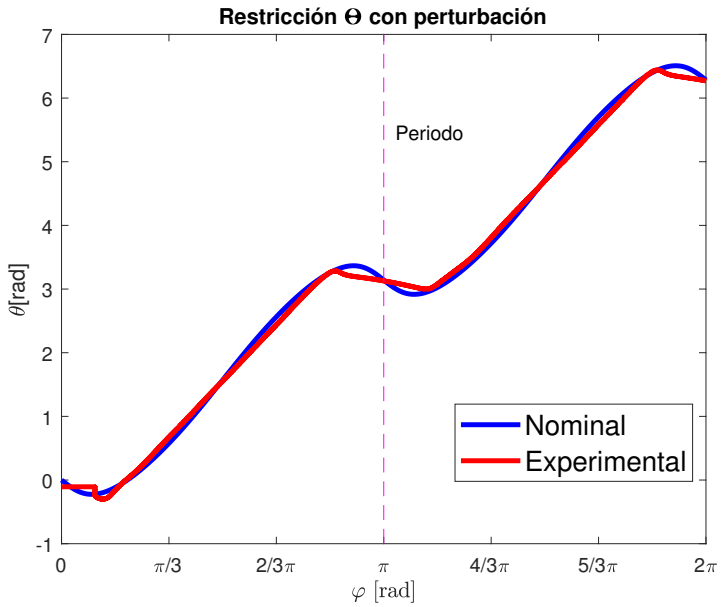


Figura 4.8 Restricción virtual Θ con perturbación externa.

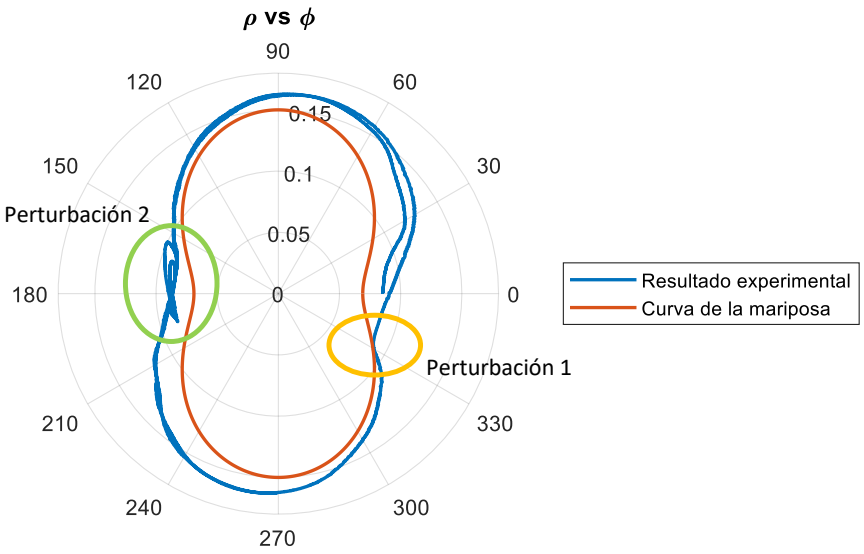


Figura 4.9 Órbita trazada por ρ con perturbación externa.

No obstante, lo que es más fructífero de estudiar en este caso, son los efectos que las perturbaciones generaron en el resultado experimental. Partamos de la primera perturbación hecha, la cual se presenta en el lado derecho de la figura 4.9, donde se observa que al acelerar la pelota en el sentido contrario de giro del motor provoca una acción de control tan rápida que el desfase entre las curvas se hace casi nulo, y la curva experimental deja de ser suave en ese momento (zona naranja de la figura 4.9). La segunda perturbación es un tanto más evidente en la gráfica, el lado izquierdo presenta un fenómeno que físicamente significa el regreso de la pelota; es decir, al momento de empujar la pelota en el sentido

de giro del motor, esta se desplaza como si regresara a una posición previa de la que tiene antes de la perturbación, por lo cual la curva genera estas pequeñas semi-elipses (zona verde de la figura 4.9).

Pese a los efectos que las perturbaciones ingresadas al sistema provocan, el controlador cumple con lo deseado, mantener a la órbita como un ciclo límite exponencialmente estable. De aquí la importancia de utilizar un esquema de estabilidad orbital.

4.2.3. Coordenadas Transversales con Perturbación

Considerando los resultados experimentales anteriores, el comportamiento de las coordenadas transversales presentará variaciones significativas con respecto a lo mostrado en la figura 4.3 por tratarse del caso no perturbado. Los efectos de la perturbación se muestran en la figura 4.10.

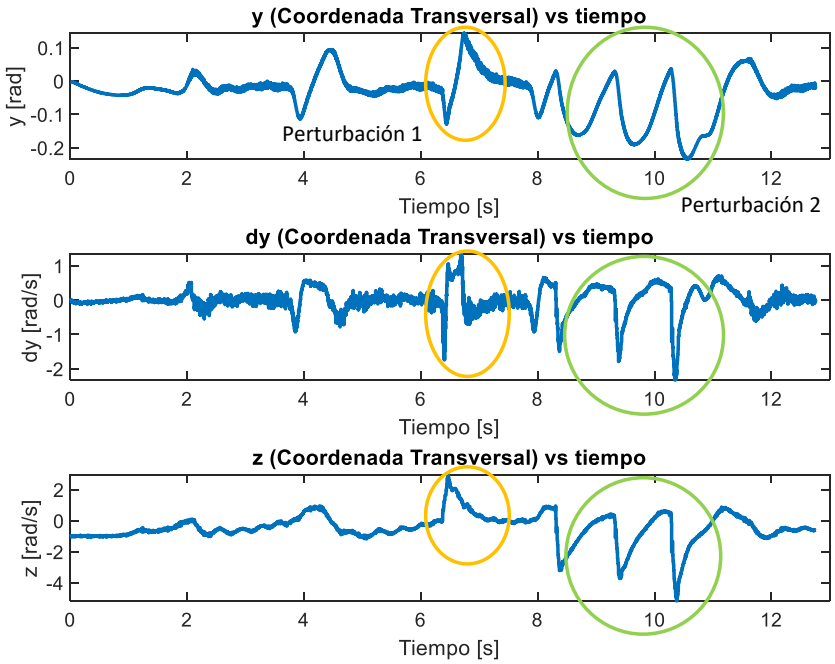


Figura 4.10 Coordenadas transversales ξ con perturbación externa.

El primer cambio significativo es que el comportamiento periódico que se presentaba en el escenario sin perturbaciones ya no es evidente, lo cual se debe a que los errores entre las variables tienen cambios inesperados, que se evidencían en las funciones de error (coordenadas transversales); a pesar de ello, los errores si bien son considerables no provocan que el sistema se inestabilice y se cancele la acción de control. En el caso de la perturbación que acelera la pelota, las coordenadas transversales presentan un error relativamente pequeño (menor a $\pm 0.25[rad]$ para la coordenada y, menor a $\pm 2.5[\frac{rad}{s}]$ para la coordenada dy y menor a $\pm 4.5[\frac{rad}{s}]$ para la coordenada z), pues el controlador lo interpreta como un acercamiento a la órbita deseada; pero, en el caso de la segunda perturbación, las coordenadas transversales crecen en función de qué tanto se desplaza la pelota, hasta estabilizarlo y continuar con su comportamiento periódico. Lo anterior será más evidente en las gráficas correspondientes a la posición $(x.y)$ de la pelota.

4.2.4. Posición (x,y) de la pelota con Perturbación

El comportamiento periódico presente en el escenario sin perturbaciones, tiene un cambio significativo una vez que se cuenta con ellas. Hasta este punto, las gráficas mostradas si bien presentan efectos detectables en el comportamiento experimental, no son totalmente claras. En la posición (x, y) se espera que las perturbaciones descritas se muestren más evidentes.

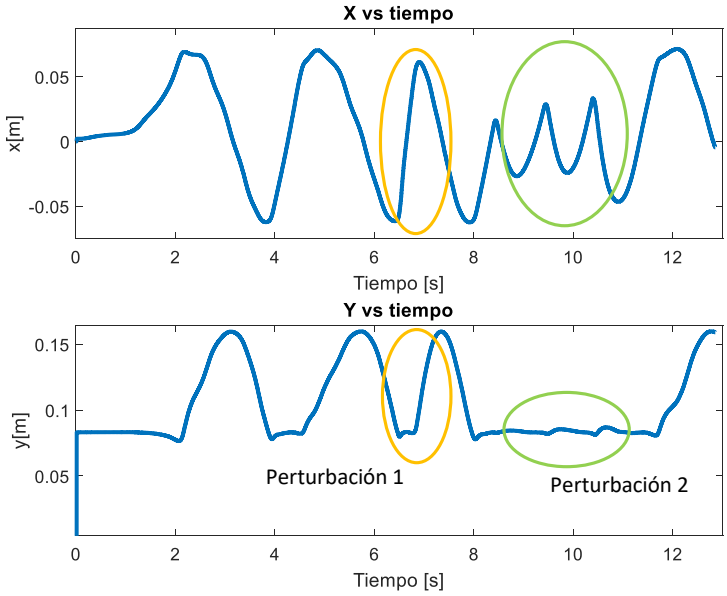


Figura 4.11 Evolución temporal (x,y) de la pelota con perturbación externa.

La figura 4.11, es la que mejor presenta las consecuencias que las perturbaciones tienen en el sistema y expone el efecto de ellas en el prototipo físico. Se analizará la coordenada y; el resultado de la primer perturbación se hace evidente en el tercer lóbulo (zonas naranjas de la figura 4.11), el cual es más corto que los anteriores, debido a la aceleración que se le imprimió a la pelota; por su parte, la segunda perturbación es evidente después del tercer lóbulo (zonas verdes de la figura 4.11), pues como la pelota se regresaba continuamente a posiciones pasadas en la parte más angosta de la mariposa (que es casi constante), en la gráfica la posición tiene variaciones mínimas.

4.2.5. Entradas de Control con Perturbación

Así como sucedió en cada uno de los resultados experimentales anteriores, el comportamiento periódico no se presenta al momento de tener perturbaciones. Los controladores deben ajustar su acción en función de lo medido, entonces se deben observar las entradas de control experimental. Empezaremos con la entrada w.

Dentro de la figura 4.12 se encuentran los efectos que el control presenta para contrarrestar a las perturbaciones. Cuando la perturbación acelera a la pelota, la acción de control debe ser más rápida, y por lo tanto violenta, tal y como se observa a los 6 [s] (zona naranja de la figura 4.12); en el caso en que la pelota retrocede y se mantiene en la parte central, la acción de control es más lenta y su magnitud crece conforme el sistema pasa tiempo con la perturbación, esto se muestra a partir de los 8

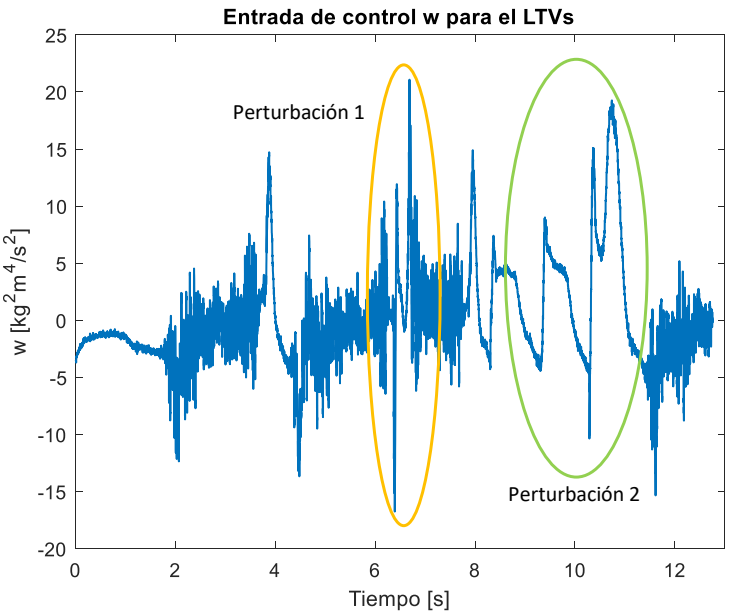


Figura 4.12 Entrada de control w para el sistema LTVs con perturbación externa.

[s] (zona verde de la figura 4.12).

Notaremos que en la entrada de control total u sucederán efectos similares a los que se tienen en la entrada de control para el LTVs.

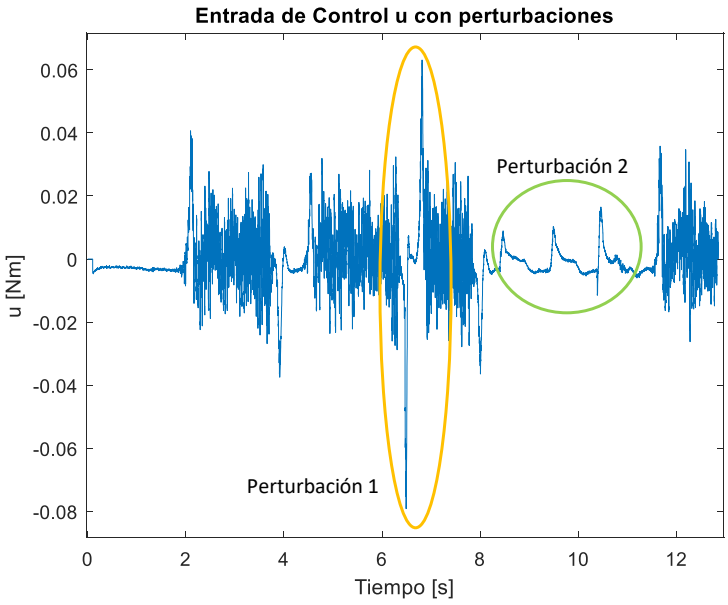


Figura 4.13 Entrada de control u con perturbación externa.

Los resultados experimentales que la entrada de control de la ec.(3.32) presenta en el prototipo *Butterfly Robot* (figura 4.13), demuestran que pueden contrarrestrar perturbaciones pequeñas en

magnitud de manera satisfactoria, pues mantiene su acción acotada en $\pm 0.1[Nm]$, permitiendo la implementación adecuadamente. Si las perturbaciones provocan una acción de control mayor a los niveles permitidos el sistema se deshabilitará automáticamente.

4.3. Discusión de los resultados

Los resultados obtenidos bajo los diferentes escenarios demuestran que el modelado del sistema otorga una representación matemática bastante apropiada del prototipo real, pues en la verificación experimental se obtiene un comportamiento adecuado para generar y estabilizar una órbita deseada.

El primer escenario demuestra que la entrada de control utilizada cumple con la tarea de generar la oscilación necesaria para conducir las dinámicas del sistema a la órbita correspondiente al movimiento de la mariposa, y una vez que se encuentra ahí, asegurar la estabilidad dentro de ella. Esto se muestra en el comportamiento periódico en cada uno de los resultados establecidos en la sección 4.1.

El segundo escenario permite comprobar que aún sin considerar la entrada de control robusto, el controlador presenta cierto grado de robustez experimental ante perturbaciones; pues a pesar de presentarse fuerzas externas inesperadas, la acción de control solventa estos efectos y mantiene a las dinámicas del *Butterfly Robot* dentro de la curva de la mariposa. Experimentalmente se traduce en que el sistema no se inestabiliza y el controlador continúa su acción de manera correcta.

Al momento de incluir perturbaciones en el sistema, la acción de control tiende a aumentar o disminuir su magnitud según el sentido en que se aplique la perturbación. Se demostró que cuando la perturbación va en sentido contrario al giro del motor, se necesita mayor magnitud y rapidez, con el fin de contrarrestar la aceleración provocada a la pelota; en contraste, cuando la perturbación va en el sentido de giro del motor, la acción de control tiende a disminuir su rapidez y su magnitud. Debido a lo anterior, la acción de control puede sobrepasar el valor máximo permitido en la programación ($\pm 0.1[Nm]$), provocando una desactivación automática del sistema de control. De esta manera, se puede conjeturar que existe una cota para las perturbaciones, que de ser excedida provoca que el sistema se desactive, y el controlador propuesto ya no presente el comportamiento correcto.

Capítulo 5

Conclusiones

El concepto de estabilización orbital presenta un enfoque relativamente nuevo para el diseño de controladores por retroalimentación de estados, es por ello que la implementación práctica de esta metodología representa una avance importante en el quéhacer tecnológico referente a los sistemas de control para sistemas dinámicos no prensiles. La validación experimental implica no solo el conocimiento teórico que compete al área de control, si no también un proceso de implementación de la planta, dentro de lo que se engloba el ámbito computacional (software), electrónico y de comunicación.

Para el *Butterfly Robot* se aplicó el algoritmo de control por linealización de coordenadas transversales, donde fue necesario llevar un proceso de modelado matemático, programación y evaluación experimental. En primera instancia, el modelo matemático presentado fue desarrollado siguiendo una metodología basada en las ecuaciones de Euler-Lagrange, y se uso como eje principal para la adecuación del algoritmo de control. Una vez que la entrada de control fue deducida mediante el enfoque de control óptimo (LQR) y el procedimiento descrito por [Surov*, et al.*, 2015], se procedió a la implementación computacional; si bien, el software ya estaba realizado, se llevó a cabo un análisis completo de cada una de las aplicaciones que lo integran, con el fin de entender el funcionamiento y la interrelación de cada parte, y de esta manera, generar un manual de uso para el robot. En la sección 3.3 se encontró que cada una de las aplicaciones utilizadas se centra en resolver un problema de implementación: el sistema de visión obtiene la variable de estado φ; el sistema de cálculo de control se encarga de leer las variables de estado, interpretarlas, acondicionarlas y cálcular la entrada de control necesaria; y, el sistema de control del servomotor permite medir la variable θ y "traducir" la señal de control en voltaje para el motor.

Con la implementación del prototipo físico se realizó la validación experimental. Se constató que el modelo matemático representa de manera adecuada al sistema, pues permitió la correcta experimentación bajo dos escenarios. El primer escenario fue libre de perturbaciones externas, y sirvió para comprobar la realización práctica de la estabilización orbital sobre la curva de la mariposa; el sistema presentó un comportamiento correcto para generar y estabilizar a la órbita, convirtiéndola en un ciclo periódico limite estable. El segundo escenario fungió para probar si la entrada de control utilizada (que carece de una entrada de control robusto), es resistente a perturbaciones acotadas realizadas en la zona más estrecha de la curva de la mariposa; se encontró que para perturbaciones pequeñas en magnitud (ver sección 4.2), el controlador mantiene a las trayectorias del sistema dentro de la vecindad que describe a la curva. Sin embargo, en ambos casos se presentan desfases entre la curva deseada y la curva experimental, los cuales se deben a limitaciones físicas y computacionales

que la implementación representa; además, de todas las dinámicas no modeladas como son fricciones secas, parámetros internos del motor o perturbaciones de alta magnitud.

Lo anterior denota que el algoritmo de control por linealización de coordenadas transversales resuelve de manera satisfactoria al problema de estabilización orbital periódica (Problema 3.1.1), y evidencía la ventaja que tiene el enfoque de estabilización orbital respecto al *tracking control*, pues se comprobó que la dependencia temporal puede prescindirse en el diseño del controlador, y aún así asegurar que el sistema se mantendrá en todo momento acotado al movimiento deseado.

5.1. Trabajo Futuro

El trabajo futuro se centra en la implementación de distintos algoritmos de control que complementen o reemplacen alguna de las entradas correspondientes al esquema de control general (ec.3.18). A continuación se listan algunos:

- **Control Robusto**: Existen dos razones primordiales para instaurar una entrada de control robusto. La primera retoma lo mencionado en la sección 4.1, la cual expone la existencia de un desfase no equidistante entre la curva experimental y la curva nominal, que puede deberse a dinámicas no modeladas o a perturbaciones, así es posible corregir estos problemas utilizando una entrada de control robusto, donde el principal exponente es el enfoque de control dado por SMC ([Shtessel*, et al.*, 2014]) . La segunda razón recae en el tipo de perturbaciones, las dos perturbaciones realizadas en esta tesis fueron hechas en la zona más estrecha de la mariposa, pues aquellas perturbaciones realizadas en los lóbulos más grandes provocaban que el sistema se inestabilizara, se debe evaluar el uso de la entrada de control robusto como una solución para este problema.
- **Función *Spline***: El uso de funciones *spline* para la implementación computacional, si bien permite llevarla a cabo, también limita el comportamiento del sistema, pues existen regiones donde se pierde la *suavidad* o estas funciones se vuelven discontinuas. Es por ello que se deben evaluar mejores herramientas computacionales u otros algoritmos que resuelvan al sistema LTVs dado por la ec.(3.26), con el fin de mejorar y optimizar el comportamiento al momento del procesamiento digital.
- **Cambios de trayectoria**: Por las limitaciones propias del controlador por linealización de coordenadas transversales, los cambios de trayectoria implican un cálculo completo para redefinir la entrada de control.
- **Implementación computacional**: El software desarrollado es relativamente antiguo, pues hace uso de *Ubuntu 16.04*. La actualización del sistema está disponible en el GitLab del profesor Maksim Surov; sin embargo, su implementación aún no es muy clara, y debe ser estudiada.

Apéndice A

Códigos para la implementación del controlador

Para la implementación del controlador se mostrarán 3 códigos completos.

A.1. Archivo de creación para la aplicación de visión

```
# Specify the minimum CMAKE version required
cmake_minimum_required(VERSION 2.8)


#
# CameraApp
#
project(CameraApp CXX)

find_package(OpenCV REQUIRED)
include_directories(${OpenCV_INCLUDE_DIRS})

set(CAMERA_APP_SRC
src/camera_basler1300_200uc.cpp
src/color_filter.cpp
src/common.cpp
src/cv_helpers.cpp
src/detector.cpp
src/image_saver.cpp
src/imgdump.cpp
src/logger.cpp
src/main_camera.cpp
src/recorder.cpp
src/sensor.cpp
src/traces.cpp

src/camera.h
src/color_filter.h
src/common.h
src/cv_helpers.h
src/cv_parallel.h
src/detector.h
src/image_saver.h
src/imgdump.h
src/logger.h
src/parse_args.h
src/recorder.h
src/sensor.h
src/tcp.h
src/traces.h
src/serializer.h
)

if (UNIX)
```

```
45      set(CAMERA_APP_SRC ${CAMERA_APP_SRC} src/tcp_unix.cpp)
46      elseif (WIN32)
47      set(CAMERA_APP_SRC ${CAMERA_APP_SRC} src/tcp_win.cpp)
48      else ()
49      message(FATAL_ERROR "unsupported OS")
50      endif()
51
52      add_executable(CameraApp ${CAMERA_APP_SRC})
53
54      if (UNIX)
55      find_package(Threads REQUIRED)
56      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -march=native -mtune=native -Ofast")
57
58      target_link_libraries(CameraApp "${CMAKE_THREAD_LIBS_INIT}")
59      execute_process(COMMAND /opt/pylon5/bin/pylon-config --cflags OUTPUT_VARIABLE PYLON_CFLAGS
        OUTPUT_STRIP_TRAILING_WHITESPACE)
60      execute_process(COMMAND /opt/pylon5/bin/pylon-config --libs OUTPUT_VARIABLE PYLON_LDFLAGS
        OUTPUT_STRIP_TRAILING_WHITESPACE)
61      execute_process(
62      COMMAND /opt/pylon5/bin/pylon-config --libdir
63      OUTPUT_VARIABLE PYLON_LIBDIR
64      OUTPUT_STRIP_TRAILING_WHITESPACE
65      )
66      target_link_libraries(CameraApp ${PYLON_LIBDIR}/libpylonbase.so)
67      target_link_libraries(CameraApp ${PYLON_LIBDIR}/libpylonutility.so)
68      target_link_libraries(CameraApp ${PYLON_LIBDIR}/libGenApi_gcc_v3_0_Basler_pylon_v5_0.so)
69      target_link_libraries(CameraApp ${PYLON_LIBDIR}/libGCBase_gcc_v3_0_Basler_pylon_v5_0.so)
70      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${PYLON_CFLAGS}")
71
72      elseif (WIN32)
73      set(OpenCV_DIR $ENV{OPENCV_BUILD})
74      message(STATUS "OpenCV found at: " ${OpenCV_DIR})
75      add_definitions(-D_CRT_SECURE_NO_WARNINGS=1 -DNOMINMAX)
76      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /fp:fast")
77      target_link_libraries(CameraApp ws2_32)
78
79      else()
80      message(FATAL_ERROR "unsupported OS")
81
82      endif()
83
84
85      add_subdirectory(3rd_party/jsonxx)
86      set_target_properties(CameraApp PROPERTIES OUTPUT_NAME cam)
87      target_link_libraries(CameraApp ${OpenCV_LIBS})
88      target_link_libraries(CameraApp jsonxx)
```

Código A.1 Archivo Cmake para la integración de la aplicación de visión.

A.2. Código `butterfly.cpp`

```
1       #include "butterfly.h"
2       #include "filters.h"
3       #include <fstream> //New
4
5       using namespace std;
6
7
8       Butterfly::Butterfly() //Inicializa las variables a utilizar
9       {
10          m_theta = 0;
11          m_dtheta = 0;
12          m_x = 0;
13          m_y = 0;
14          m_vx = 0;
15          m_vy = 0;
16          m_phi = 0;
17          m_dphi = 0;
18          m_stop = false;
19          m_sync_delay = 0;
20          m_ball_found = false;
21      }
22
```

```
23      Butterfly::~Butterfly()
24      {
25      }
26
27      void Butterfly::init(jsonxx::Object const& jscfg) //Inicializa el hardware
28      {
29           info_msg("initializing hardware..");
30
31           auto const& jsctrl = json_get<jsonxx::Object>(jscfg, "controller");
32           m_sync_delay = json_get<int64_t>(jsctrl, "cam_delay_usec", 0, 1e+5); //Procesa un antidelay de las
     senales de la camara
33           m_delay_theta.init(m_sync_delay);
34           m_delay_dtheta.init(m_sync_delay);
35
36           m_servo = ServoIfc::capture_instance(); //Indica comunicacion con la camara
37           m_servo->init(jscfg);
38
39           m_camera = Camera::capture_instance(); //Indica comunicacion con el servomotor
40           m_camera->init(jscfg);
41
42           info_msg("done");
43      }
44
45      void wait_for_camera_ready()
46      {
47
48      }
49
50      void Butterfly::measure() //Medicion y acondicionamiento de las variables de estado
51      {
52           int64_t t_servo;
53           int status = m_servo->get_state(t_servo, m_theta, m_dtheta, true); //Toma theta de la aplicacion del
     servo
54           if (status < 0)
55           throw runtime_error("servo disconnected");
56
57           double theta_delayed = m_delay_theta.process(t_servo, m_theta);
58           double dtheta_delayed = m_delay_dtheta.process(t_servo, m_dtheta);
59
60           int64_t t_cam;
61           status = m_camera->get(t_cam, m_x, m_y); //Toma x, y de la aplicacion de la camara
62
63           switch (status)
64           {
65                case 1:
66                {
67                     m_vx = m_diff_x.process(t_cam, m_x);
68                     m_vy = m_diff_y.process(t_cam, m_y);
69
70                     double alpha = atan2(m_x, m_y);
71                     double dalpha = (m_y * m_vx - m_x * m_vy) / (m_x * m_x + m_y * m_y);
72
73                     m_phi = theta_delayed + alpha; //Calcula varphi
74                     m_dphi = dtheta_delayed + dalpha;
75                     m_ball_found = true;
76                     break;
77                }
78                case 0:
79                {
80                     break;
81                }
82                default:
83                {
84                     if (m_ball_found) //Identifica que la pelota no es reconocida por el sistema de vision
85                     info_msg("ball was lost");
86                     m_ball_found = false;
87                     break;
88                }
89           }
90      }
91
92      void Butterfly::stop()
93      {
94           m_stop = true;
95      }
96
```

```
97    void Butterfly::get_signals(int64_t const& t, BflySignals& signals) //Asigna las senales de entrada y
      salida
98    {
99        signals.t = t * 1e-6;
100       signals.ball_found = m_ball_found;
101       signals.theta = m_theta;
102       signals.dtheta = m_dtheta;
103       signals.phi = m_phi;
104       signals.dphi = m_dphi;
105       signals.x = m_x;
106       signals.vx = m_vx;
107       signals.y = m_y;
108       signals.vy = m_vy;
109       signals.torque = 0;
110   }
111
112   void Butterfly::start(callback_t const& cb) //Programa Main para el Butterfly Robot
113   {
114       if (!m_camera || !m_servo) //Confirma la interconexion de aplicaciones
115       throw runtime_error("Butterfly not initialized yet");
116
117       m_camera->start();
118       m_servo->start();
119
120       int status;
121       int64_t t, t0;
122       t0 = get_time_usec();
123       //Codigo agregado por mi para obtener datos practicos
124       ofstream filetime("/home/butterfly/dev/nuc_controller/out/timeOut.txt");//New
125       ofstream filetorque("/home/butterfly/dev/nuc_controller/out/torqueOut.txt");//New
126       ofstream filetheta("/home/butterfly/dev/nuc_controller/out/thetaOut.txt");//New
127       ofstream filephi("/home/butterfly/dev/nuc_controller/out/phiOut.txt");//New
128       ofstream filedtheta("/home/butterfly/dev/nuc_controller/out/dthetaOut.txt");//New
129       ofstream filedphi("/home/butterfly/dev/nuc_controller/out/dphiOut.txt");//New
130       ofstream filex("/home/butterfly/dev/nuc_controller/out/xOut.txt");//New
131       ofstream filey("/home/butterfly/dev/nuc_controller/out/yOut.txt");//New
132
133       while (!m_stop) //Si se reconoce la pelota se mantiene en el while
134       {
135           t = get_time_usec();
136           measure();
137
138           BflySignals signals;
139           get_signals(t - t0, signals);
140           status = cb(signals);
141           if (!status)
142           m_stop = true;
143
144           m_servo->set_torque(signals.torque); //Manda al servomotor la senal de control
145           filetime << signals.t << ",";//New
146           filetorque << signals.torque << ",";//New
147           filetheta << signals.theta << ",";//New
148           filephi << signals.phi << ",";//New
149           filedtheta << signals.dtheta << ",";//New
150           filedphi << signals.dphi << ",";//New
151           filex << signals.x << ",";//New
152           filey << signals.y << ",";//New
153
154       }
155
156       m_servo->stop(); //Detiene el proceso del servo
157       m_camera->stop(); //Detiene el proceso de la camara
158       //Se cierran los archivos de escritura de datos
159       filetime.close();//New
160       filetorque.close();//New
161       filetheta.close();//New
162       filephi.close();//New
163       filedtheta.close();//New
164       filedphi.close();//New
165       filex.close();//New
166       filey.close();//New
167
168
169       info_msg("stopped");
170   }
```

Código A.2 Código `butterfly.cpp`.

A.3. Código overturn_control.cpp

```
#include "matrix.h"
#include "splines.h"
#include "filters.h"
#include <fstream> //New


using namespace std;

ofstream fileky("/home/butterfly/dev/nuc_controller/out/kyOut.txt");//New
ofstream filekdy("/home/butterfly/dev/nuc_controller/out/kdyOut.txt");//New
ofstream filekz("/home/butterfly/dev/nuc_controller/out/kzOut.txt");//New
ofstream fileyco("/home/butterfly/dev/nuc_controller/out/ycOut.txt");//New
ofstream filezco("/home/butterfly/dev/nuc_controller/out/zcOut.txt");//New
ofstream filedyco("/home/butterfly/dev/nuc_controller/out/dycOut.txt");//New
ofstream filedphis("/home/butterfly/dev/nuc_controller/out/dphisOut.txt");//New
ofstream filethetas("/home/butterfly/dev/nuc_controller/out/thetasOut.txt");//New
ofstream filedthetas("/home/butterfly/dev/nuc_controller/out/dthetasOut.txt");//New
ofstream filevc1("/home/butterfly/dev/nuc_controller/out/vc1Out.txt");//New
ofstream filevc2("/home/butterfly/dev/nuc_controller/out/vc2Out.txt");//New
ofstream filerho("/home/butterfly/dev/nuc_controller/out/rho.txt");//New
ofstream filerhod1("/home/butterfly/dev/nuc_controller/out/rhod1.txt");//New
ofstream filerhod2("/home/butterfly/dev/nuc_controller/out/rhod2.txt");//New
ofstream filew("/home/butterfly/dev/nuc_controller/out/w.txt");//New



inline double spline_dphi(double arg, int der=0)
{
     static spline s(5, {-0.0191204608723,-0.00714628929586,-0.00315436770211,0.0,0.00399199796585...
     3.14159265359,3.14558465156,3.14957654151,3.16155048192},...
     {0.999689303094,0.999767222325,0.999854795574,0.999933607176,0.999991355644,...
     0.999922895581,0.99983492871,0.999742802199,0.999661451433}, "none");
     return s(arg, der);
}

inline double spline_vc(double arg, int der=0)
{
     static spline s(5, {-0.1,-0.0750627413911,-0.0667503218548,-0.0584379023185,-0.0501254827822,...
     3.20003055591,3.20834297544,3.21665539498,3.24159265359},...
     {0.0565874772989,0.0537475080241,0.0499654500309,0.0452447495562,0.0395888173845,...
     3.09634790402,3.09162720355,3.08784514556,3.08500517628}, "none");
     return s(arg, der);
}

inline double spline_rho(double arg, int der=0)
{
     static spline s(5, {-0.0259916157659,-0.0103954212891,-0.00519762315263,3.90788876463e
-12,0.00519762315912,...
     6.28318530718,6.28838293034,6.29358072847,6.30917692295},...
     {0.085375715712,0.0853652143387,0.0853533184262,0.0853419526262,0.085333562149,...
     0.0853419526262,0.0853533184261,0.0853652143386,0.0853757157119}, "none");
     return s(arg, der);
}

inline double spline_kz(double arg, int der=0)
{
     static spline s(5, {-0.0156766100479,-0.00627064401914,-0.00313532200957,0.0,0.00313532200957,...
     3.14159265359,3.1447279756,3.14786329761,3.15726926364},...
     {3.4913601002,3.49871460191,3.52335647926,3.52805836488,3.56937826914,...
     3.65310707103,3.66248707926,3.69275277838,3.70477341229}, "none");
     return s(arg, der);
}

inline double spline_ky(double arg, int der=0)
{
     static spline s(5, {-0.0156766100479,-0.00627064401914,-0.00313532200957,0.0,0.00313532200957,...
     3.14159265359,3.1447279756,3.14786329761,3.15726926364},...
     {-52.9159280529,-52.9681260893,-53.1720358053,-53.1897045855,-53.5338075304,...
     -54.1849616041,-54.2409425062,-54.4920188832,-54.5829046771}, "none");
     return s(arg, der);
}

inline double spline_kdy(double arg, int der=0)
{
```

```
74          static spline s(5, {-0.0156766100479,-0.00627064401914,-0.00313532200957,0.0,0.00313532200957,...
75          3.14159265359,3.1447279756,3.14786329761,3.15726926364},...
76          {-12.4067600279,-12.4125450747,-12.4547040303,-12.4450940912,-12.5162666361,...
77          -12.6165758102,-12.6118296382,-12.6607198337,-12.6717573447}, "none");
78          return s(arg, der);
79      }
80
81      inline Mat2x2 sub_M(double const& theta, double const& phi)
82      {
83          auto rho_val = spline_rho(phi);
84          auto rho_d1_val = spline_rho(phi, 1);
85          //return Mat2x2(0.003*pow(rho_val, 2)*pow(sin(phi), 2) + 0.003*pow(rho_val, 2)*pow(cos(phi), 2) +
    0.0015816125, 0.003*sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*
    sin(phi), 2))*(-rho_val*(rho_d1_val*sin(phi) + rho_val*cos(phi))*cos(phi)/sqrt(pow(rho_d1_val*sin(phi) +
    rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)) + rho_val*(rho_d1_val*cos(phi) -
    rho_val*sin(phi))*sin(phi)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) -
    rho_val*sin(phi), 2)) - 0.0183347079152333), 0.003*sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow
    (rho_d1_val*cos(phi) - rho_val*sin(phi), 2))*(-rho_val*(rho_d1_val*sin(phi) + rho_val*cos(phi))*cos(phi)/
    sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)) +
    rho_val*(rho_d1_val*cos(phi) - rho_val*sin(phi))*sin(phi)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi),
    2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)) - 0.0183347079152333), 0.00793951612903226*pow(
    rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + 0.00793951612903226*pow(rho_d1_val*cos(phi) - rho_val*sin(phi),
     2));
86          return Mat2x2(0.003*pow(rho_val, 2)*pow(sin(phi), 2) + 0.003*pow(rho_val, 2)*pow(cos(phi), 2) +
    0.0015816125, 0.003*sqrt(pow(rho_d1_val,2)+pow(rho_val,2))*(-pow(rho_val,2)/sqrt(pow(rho_d1_val,2)+pow(
    rho_val,2)) - 0.0183347079152333), 0.003*sqrt(pow(rho_d1_val,2)+pow(rho_val,2))*(-pow(rho_val,2)/sqrt(pow(
    rho_d1_val,2)+pow(rho_val,2)) - 0.0183347079152333), 0.00793951612903226*(pow(rho_d1_val,2)+pow(rho_val,2)))
    ; //New
87      }
88
89      inline Mat2x2 sub_C(double const& theta, double const& phi, double const& dtheta, double const& dphi)
90      {
91          auto rho_val = spline_rho(phi);
92          auto rho_d1_val = spline_rho(phi, 1);
93          auto rho_d2_val = spline_rho(phi, 2);
94          filerho << rho_val << ",";//New
95          filerhod1 << rho_d1_val << ",";//New
96          filerhod2 << rho_d2_val << ",";//New
97          //return Mat2x2(0.003*dphi*(rho_val*(rho_d1_val*sin(phi) + rho_val*cos(phi))*sin(phi)/sqrt(pow(
    rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)) + rho_val*(
    rho_d1_val*cos(phi) - rho_val*sin(phi))*cos(phi)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(
    rho_d1_val*cos(phi) - rho_val*sin(phi), 2)))*sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(
    rho_d1_val*cos(phi) - rho_val*sin(phi), 2)), 0.003*dphi*(((rho_d1_val*sin(phi) + rho_val*cos(phi))*(2*
    rho_d1_val*cos(phi) + rho_d2_val*sin(phi) - rho_val*sin(phi)) + (-rho_d1_val*cos(phi) + rho_val*sin(phi))
    *(2*rho_d1_val*sin(phi) - rho_d2_val*cos(phi) + rho_val*cos(phi)))*(-rho_val*(rho_d1_val*sin(phi) + rho_val*
    cos(phi))*cos(phi)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*
    sin(phi), 2)) + rho_val*(rho_d1_val*cos(phi) - rho_val*sin(phi))*sin(phi)/sqrt(pow(rho_d1_val*sin(phi) +
    rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)) - 0.0183347079152333)/sqrt(pow(
    rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)) + (-rho_val*((
    rho_d1_val*sin(phi) + rho_val*cos(phi))*(-(rho_d1_val*sin(phi) + rho_val*cos(phi))*(2*rho_d1_val*cos(phi) +
    rho_d2_val*sin(phi) - rho_val*sin(phi)) - (-rho_d1_val*cos(phi) + rho_val*sin(phi))*(2*rho_d1_val*sin(phi) -
     rho_d2_val*cos(phi) + rho_val*cos(phi)))/pow(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(
    rho_d1_val*cos(phi) - rho_val*sin(phi), 2), 3.0L/2.0L) + (2*rho_d1_val*cos(phi) + rho_d2_val*sin(phi) -
    rho_val*sin(phi))/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*
    sin(phi), 2)))*cos(phi)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) -
    rho_val*sin(phi), 2)) + rho_val*((rho_d1_val*cos(phi) - rho_val*sin(phi))*(-(rho_d1_val*sin(phi) + rho_val*
    cos(phi))*(2*rho_d1_val*cos(phi) + rho_d2_val*sin(phi) - rho_val*sin(phi)) - (-rho_d1_val*cos(phi) + rho_val
    *sin(phi))*(2*rho_d1_val*sin(phi) - rho_d2_val*cos(phi) + rho_val*cos(phi)))/pow(pow(rho_d1_val*sin(phi) +
    rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2), 3.0L/2.0L) + (-2*rho_d1_val*sin(phi)
    + rho_d2_val*cos(phi) - rho_val*cos(phi))/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(
    rho_d1_val*cos(phi) - rho_val*sin(phi), 2)))*sin(phi)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) +
    pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)))*(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(
    rho_d1_val*cos(phi) - rho_val*sin(phi), 2))) + 0.003*dtheta*(rho_val*(rho_d1_val*sin(phi) + rho_val*cos(phi)
    )*sin(phi)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi),
     2)) + rho_val*(rho_d1_val*cos(phi) - rho_val*sin(phi))*cos(phi)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(
    phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)))*sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi),
     2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)), 0.003*dtheta*(-rho_val*(rho_d1_val*sin(phi) + rho_val
    *cos(phi))*sin(phi)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*
    sin(phi), 2)) - rho_val*(rho_d1_val*cos(phi) - rho_val*sin(phi))*cos(phi)/sqrt(pow(rho_d1_val*sin(phi) +
    rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)))*sqrt(pow(rho_d1_val*sin(phi) +
    rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)), 0.00793951612903226*dphi*((
    rho_d1_val*sin(phi) + rho_val*cos(phi))*(2*rho_d1_val*cos(phi) + rho_d2_val*sin(phi) - rho_val*sin(phi)) +
    (-rho_d1_val*cos(phi) + rho_val*sin(phi))*(2*rho_d1_val*sin(phi) - rho_d2_val*cos(phi) + rho_val*cos(phi))))
    ;
98          return Mat2x2(0.003*dphi*rho_val*rho_d1_val, 0.003*dphi*((rho_val*rho_d1_val+rho_d1_val*rho_d2_val)*(-
    pow(rho_val,2)/sqrt(pow(rho_d1_val,2)+pow(rho_val,2)) - 0.0183347079152333)/sqrt(pow(rho_d1_val,2)+pow(
    rho_val,2)) + ((pow(rho_val,3)*rho_d1_val+pow(rho_val,2)*rho_d1_val*rho_d2_val)/(pow(rho_d1_val,2)+pow(
```

```
     rho_val,2))-2*rho_val*rho_d1_val)) + 0.003*dtheta*rho_val*rho_d1_val, -0.003*dtheta*rho_val*rho_d1_val,
     0.00793951612903226*dphi*(rho_val*rho_d1_val+rho_d1_val*rho_d2_val)); //New
99   }
100
101  inline Vec2 sub_G(double const& theta, double const& phi)
102  {
103       auto rho_val = spline_rho(phi);
104       auto rho_d1_val = spline_rho(phi, 1);
105       //return Vec2(0.02943*rho_val*sin(phi)*cos(theta) - 0.02943*rho_val*sin(theta)*cos(phi), 0.02943*((
     rho_d1_val*sin(phi) + rho_val*cos(phi))*sin(theta)/sqrt(pow(rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow
     (rho_d1_val*cos(phi) - rho_val*sin(phi), 2)) + (rho_d1_val*cos(phi) - rho_val*sin(phi))*cos(theta)/sqrt(pow(
     rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2)))*sqrt(pow(
     rho_d1_val*sin(phi) + rho_val*cos(phi), 2) + pow(rho_d1_val*cos(phi) - rho_val*sin(phi), 2))); \\Matriz dada
      por PhD. Maksim Surov
106       return Vec2(0.02943*rho_val*sin(phi)*cos(theta) - 0.02943*rho_val*sin(theta)*cos(phi), 0.02943*((
     rho_d1_val*sin(phi) + rho_val*cos(phi))*sin(theta) + (rho_d1_val*cos(phi) - rho_val*sin(phi))*cos(theta)));
     //New (Simplificacion)
107  }
108
109
110  // double get_torque(double theta, double phi, double dtheta, double dphi, double phi2)
111  double get_torque(double theta, double phi, double dtheta, double dphi)
112  {
113       auto n = int(floor(phi / _PI));
114       phi -= _PI * n;
115       theta -= _PI * n;
116       //Asigna las variables deseadas con los splines (todos dependientes de varphi -phi-)
117       auto dphi_s = spline_dphi(phi);
118       auto theta_s = spline_vc(phi);
119       auto vc1 = spline_vc(phi, 1);
120       auto vc2 = spline_vc(phi, 2);
121       auto kz = spline_kz(phi);
122       auto ky = spline_ky(phi);
123       auto kdy = spline_kdy(phi);
124       auto dtheta_s = vc1 * dphi_s;
125       //Calcula las coordenadas transversales
126       auto z = dphi - dphi_s;
127       auto y = theta - theta_s;
128       auto dy = dtheta - vc1 * dphi;
129       //Entrada de control w
130       auto w = z * kz + y * ky + dy * kdy;
131       //Calculo de las matrices y vectores
132       Vec2 dq_s(dtheta_s, dphi_s);
133       Mat2x2 invL(1, -vc1, 0, 1);
134       auto M = sub_M(theta, phi);
135       auto C = sub_C(theta_s, phi, dtheta_s, dphi_s);
136       auto G = sub_G(theta, phi);
137       auto invM = inv(M);
138       auto K = invL * invM;
139       //Adquisicion de datos
140       fileky << ky << ",";//New
141       filekdy << kdy << ",";//New
142       filekz << kz << ",";//New
143       fileyco << y << ",";//New
144       filedyco << dy << ",";//New
145       filezco << z << ",";//New
146       filedphis << dphi_s << ",";//New
147       filethetas << theta_s << ",";//New
148       filedthetas << dtheta_s << ",";//New
149       filevc1 << vc1 << ",";//New
150       filevc2 << vc2 << ",";//New
151       filew << w << ",";//New
152       //Senal de control tau(u)
153       auto tau = (w + (K * (G + C*dq_s)).at(0,0) + vc2 * pow(dphi_s, 2)) / K.at(0,0);
154       return tau;
155
156  }
157
158  double get_torque_sync( //Sincroniza la senal de control en caso de perdida de comunicacion
159  double theta, double phi, double dtheta, double dphi,
160  double theta2, double phi2, double dtheta2, double dphi2
161  )
162  {
163       auto n = int(floor(phi / _PI));
164       phi -= _PI * n;
165       theta -= _PI * n;
166
```

```
167         auto dphi_s = spline_dphi(phi);
168         auto theta_s = spline_vc(phi);
169         auto vc1 = spline_vc(phi, 1);
170         auto vc2 = spline_vc(phi, 2);
171         auto kz = spline_kz(phi);
172         auto ky = spline_ky(phi);
173         auto kdy = spline_kdy(phi);
174         auto dtheta_s = vc1 * dphi_s;
175
176         double k_sync = 1.0;
177         double sync_max = 0.1;
178
179         auto z = dphi - (dphi_s - clamp(k_sync * excess(phi - phi2, _PI), -sync_max, sync_max));
180         auto y = theta - theta_s;
181         auto dy = dtheta - vc1 * dphi;
182
183
184         auto v = z * kz + y * ky + dy * kdy;
185
186         Vec2 dq_s(dtheta_s, dphi_s);
187         Mat2x2 invL(1, -vc1, 0, 1);
188         auto M = sub_M(theta, phi);
189         auto C = sub_C(theta_s, phi, dtheta_s, dphi_s);
190         auto G = sub_G(theta, phi);
191
192         auto invM = inv(M);
193         auto K = invL * invM;
194
195         auto tau = (v + (K * (G + C * dq_s)).at(0,0) + vc2 * pow(dphi_s, 2)) / K.at(0,0);
196         return tau;
197     }
```

Código A.3 Código `overturn_control.cpp`.

Referencias

[Antsaklis y Michel, 2007] Antsaklis, P. J. & Michel, A. N. (2007). *A Linear Systems Primer*. Birkhäuser, 1° edition. (Citado en páginas 7, 8, 19, 20, 23 y 37.)

[Cefalo, *et al.*, 2006] Cefalo, M., Lanari, L., & Oriolo, G. (2006). Energy-based control of the butterfly robot. *IFAC Proceedings Volumes*, *39*(15), 1–6. (Citado en páginas 2 y 43.)

[Chen, 1999] Chen, C.-T. (1999). *Linear System Theory and Design*. Oxford University Press, 3° edition. (Citado en páginas 10, 20 y 31.)

[Hamed y Grizzle, 2013] Hamed, K. A. & Grizzle, J. W. (2013). Robust event-based stabilization of periodic orbits for hybrid systems: Application to an underactuated 3d bipedal robot. In *2013 American Control Conference* (pp. 6206–6212). (Citado en página 1.)

[Hauser, *et al.*, 1992] Hauser, J., Sastry, S., & Kokotovic, P. (1992). Nonlinear control via approximate input-output linearization: The ball and beam example. *IEEE transactions on automatic control*, *37*(3), 392–398. (Citado en páginas 2 y 43.)

[Hendricks, *et al.*, 2008] Hendricks, E., Jannerup, O., & Sørensen, P. H. (2008). *Linear Systems Control*. Springer. (Citado en páginas 8, 14, 16, 17, 18, 21, 22 y 23.)

[Lobontiu, 2010] Lobontiu, N. (2010). *System Dynamics for Engineering Students*. Elsevier. (Citado en página 7.)

[Lynch, *et al.*, 1998] Lynch, K., Shiroma, N., Arai, H., & Tanie, K. (1998). The roles of shape and motion in dynamic manipulation: the butterfly example. *3*, 1958–1963 vol.3, https://doi.org/10.1109/ROBOT.1998.680600. (Citado en páginas 2 y 42.)

[Lynch y Mason, 1999] Lynch, K. M. & Mason, M. T. (1999). Dynamic nonprehensile manipulation: Controllability, planning, and experiments. *The International Journal of Robotics Research*, *18*(1), 64–92. (Citado en páginas 2, 42 y 43.)

[Morales, *et al.*, 2013] Morales, D. O., Hera, P. L., & Réhman, S. U. (2013). Generating periodic motions for the butterfly robot. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2527–2532). (Citado en página 3.)

[Nise, 2015] Nise, N. S. (2015). *Control Systems Engineering*. Wiley, 7° edition. (Citado en páginas 10, 11, 16, 17 y 32.)

[Ogata, 2010] Ogata, K. (2010). *Ingeniería de Control Moderna*. Pearson Educación, 5° edition. (Citado en páginas 13, 16 y 19.)

[Poznyak, 2008] Poznyak, A. S. (2008). *Advanced Mathematical Tools for Automatic Control Engineers Volume 1: Deterministic Techniques*, volume 1. Elsevier. (Citado en página 23.)

[Ryu, *et al.*, 2013] Ryu, J.-C., Ruggiero, F., & Lynch, K. M. (2013). Control of nonprehensile rolling manipulation: Balancing a disk on a disk. *IEEE Transactions on Robotics*, *29*(5), 1152–1161. (Citado en páginas 2 y 43.)

[Shiriaev, *et al.*, 2005] Shiriaev, A., Perram, J., & Canudas-de Wit, C. (2005). Constructive tool for orbital stabilization of underactuated nonlinear systems: virtual constraints approach. *IEEE Transactions on Automatic Control*, *50*(8), 1164–1176, https://doi.org/10.1109/TAC.2005.852568. (Citado en páginas 3, 4, 26, 27, 28, 29, 31 y 52.)

[Shiriaev, *et al.*, 2010] Shiriaev, A. S., Freidovich, L. B., & Gusev, S. V. (2010). Transverse linearization for controlled mechanical systems with several passive degrees of freedom. *IEEE Transactions on Automatic Control*, *55*(4), 893–906, https://doi.org/10.1109/TAC.2010.2042000. (Citado en páginas 26, 27, 28, 42, 54 y 67.)

[Shtessel, *et al.*, 2014] Shtessel, Y., Edwards, C., Fridman, L., & Levant, A. (2014). *Sliding Mode Control and Observation*. Springer New York. (Citado en páginas 35, 36 y 80.)

[Slotine y Li, 1991] Slotine, J.-J. E. & Li, W. (1991). *Applied Nonlinear Control*. Prentice Hall. (Citado en páginas 25, 33, 34 y 37.)

[Spong, 1994] Spong, M. W. (1994). Swing up control of the acrobot using partial feedback linearization*. *IFAC Proceedings Volumes*, *27*(14), 833–838, https://doi.org/https://doi.org/10.1016/S1474-6670(17)47404-0. Fourth IFAC Symposium on Robot Control, Capri, Italy, September 19-21, 1994. (Citado en página 2.)

[Surov, *et al.*, 2015] Surov, M., Shiriaev, A., Freidovich, L., Gusev, S., & Paramonov, L. (2015). Case study in non-prehensile manipulation: planning and orbital stabilization of one-directional rollings for the "butterfly" robot. (pp. 1484–1489)., https://doi.org/10.1109/ICRA.2015.7139385. (Citado en páginas 2, 3, 4, 31, 43, 48, 49, 52, 55, 56, 57, 58, 63, 67 y 79.)

[Sætre, 2022] Sætre, C. F. (2022). *Orbital Stabilization of Nonlinear Underactuated Systems*. Phd thesis, Norwegian University of Science and Technology https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/3034741/Christian%20Fredrik%20S%C3%A6tre.pdf. (Citado en páginas 1, 15, 19, 26, 27, 32, 34, 38, 39, 40, 41 y 51.)

[Zhang y Tarn, 2002] Zhang, M. & Tarn, T.-J. (2002). Hybrid control of the pendubot. *IEEE/ASME Transactions on Mechatronics*, *7*(1), 79–86, https://doi.org/10.1109/3516.990890. (Citado en página 2.)