



UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO  
DE INGENIERO EN MECATRÓNICA

TEMA:

“TÉCNICAS PARA LA IMPLEMENTACIÓN DE CONTROL  
AUTO-DISPARADO INSPIRADO EN MUESTREO ÓPTIMO”

AUTOR: JUAN PABLO BENAVIDES PIEDRA

DIRECTOR: CARLOS XAVIER ROSERO

IBARRA-ECUADOR

SEPTIEMBRE 2017



**UNIVERSIDAD TÉCNICA DEL NORTE**  
**BIBLIOTECA UNIVERSITARIA**  
**AUTORIZACIÓN DE USO Y PUBLICACIÓN**

**A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE**

## 1. IDENTIFICACIÓN DE LA OBRA

La Universidad Técnica del Norte dentro del proyecto *Repositorio Digital Institucional*, determinó la necesidad de disponer de textos completos en formato digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la Universidad.

Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto, para lo cual pongo a disposición la siguiente información:

<b>DATOS DEL AUTOR</b>			
<b>CÉDULA DE IDENTIDAD:</b>	1003019211		
<b>APELLIDOS Y NOMBRES:</b>	BENAVIDES PIEDRA JUAN PABLO		
<b>DIRECCIÓN:</b>	1003019211		
<b>EMAIL:</b>	jpbnavidesp@utn.edu.ec - juanpablo82@gmail.com		
<b>TELÉFONO FIJO:</b>	062612754	<b>TELÉFONO MÓVIL:</b>	0999326824
<b>DATOS DE LA OBRA</b>			
<b>TÍTULO:</b>	“TÉCNICAS PARA LA IMPLEMENTACIÓN DE CONTROL AUTO-DISPARADO INSPIRADO EN MUESTREO ÓPTIMO”		
<b>AUTOR:</b>	JUAN PABLO BENAVIDES PIEDRA		
<b>FECHA (AAAA-MM-DD):</b>	2017-09-25		
<b>SÓLO PARA TRABAJOS DE GRADO</b>			
<b>PROGRAMA:</b>	PREGRADO		
<b>TÍTULO POR EL QUE OPTA:</b>	INGENIERO EN MECATRÓNICA		
<b>ASESOR/DIRECTOR:</b>	CARLOS XAVIER ROSERO C.		

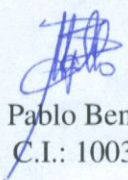
## 2. AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

Yo, Juan Pablo Benavides Piedra con cédula de identidad Nro. 1003019211, en calidad de autor y titular de los derechos patrimoniales de la obra o trabajo de grado descrito anteriormente, hago entrega del ejemplar respectivo en formato digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y uso del archivo digital en la Biblioteca de la Universidad con fines académicos, para ampliar la disponibilidad del material y como apoyo a la educación, investigación y extensión; en concordancia con la Ley de Educación Superior, Artículo 144.

### 3. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló sin violar derechos de autor de terceros, por lo tanto la obra es original, y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 25 días del mes de septiembre de 2017.



Juan Pablo Benavides Piedra  
C.I.: 1003019211



**UNIVERSIDAD TÉCNICA DEL NORTE**  
**FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS**  
**CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE GRADO A**  
**FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE**

Yo, Juan Pablo Benavides Piedra con cédula de identidad Nro. 1003019211, manifiesto mi voluntad de ceder a la Universidad Técnica del Norte los derechos patrimoniales consagrados en la Ley de Propiedad Intelectual del Ecuador, artículos 4, 5 y 6, en calidad de autor (es) de la obra o trabajo de grado denominado “TÉCNICAS PARA LA IMPLEMENTACIÓN DE CONTROL AUTO-DISPARADO INSPIRADO EN MUESTREO ÓPTIMO”, que ha sido desarrollado para optar por el título de Ingeniero en Mecatrónica, en la Universidad Técnica del Norte, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente. En mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Técnica del Norte.

Ibarra, septiembre de 2017

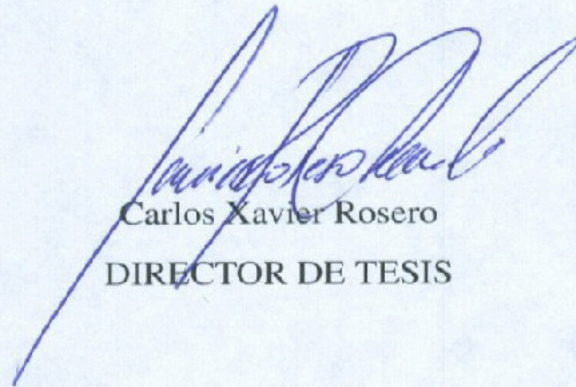
Juan Pablo Benavides Piedra  
C.I.: 1003019211



UNIVERSIDAD TÉCNICA DEL NORTE  
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS  
CERTIFICACIÓN

En calidad de director del trabajo de grado "TÉCNICAS PARA LA IMPLEMENTACIÓN DE CONTROL AUTO-DISPARADO INSPIRADO EN MUESTREO ÓPTIMO", presentado por el egresado JUAN PABLO BENAVIDES PIEDRA, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, septiembre de 2017



Carlos Xavier Rosero  
DIRECTOR DE TESIS



UNIVERSIDAD TÉCNICA DEL NORTE  
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS  
DECLARACIÓN

Yo, Juan Pablo Benavides Piedra con cédula de identidad Nro. 1003019211, declaro bajo juramento que el trabajo aquí escrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Universidad Técnica del Norte - Ibarra, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normativa institucional vigente.

Ibarra, septiembre de 2017

Juan Pablo Benavides Piedra  
C.I.: 1003019211

## **Agradecimiento**

Agradezco a Dios ya que gracias a Él todo es posible. Ha sido quien ha guiado mis pasos y me ha dado sabiduría, amor y paciencia para sobrellevar los momentos difíciles.

El presente trabajo ha sido finalizado con éxito debido al apoyo y dirección de mi director Carlos Xavier Rosero. Le agradezco por su conocimiento, tiempo y entrega profesional que han hecho que este difícil reto se termine con éxito.

A mi familia por ser fuente de motivación ya que es seguro que sin su apoyo y tenacidad no hubiese conseguido lo ahora logrado.

A la Universidad Técnica del Norte, Facultad de Ingeniería en Ciencias Aplicadas, Carrera de Ingeniería Mecatrónica y a sus respectivas autoridades, por su entera colaboración para culminar una etapa más de mi vida.

Al Grupo de Investigación en Sistemas Inteligentes por brindarme su apoyo y guía durante el desarrollo del presente trabajo. A los docentes que fueron parte fundamental de mi formación, brindándome su conocimiento y consejos.

Al Hospital del IESS de Ibarra, Institución en la que laboro, por darme las facilidades y permisos para lograr culminar esta meta. Este logro permitirá que crezca profesionalmente en mi noble trabajo.

*Juan Pablo*

## **Dedicatoria**

El presente trabajo está dedicado principalmente a mi amada madre Lucita Piedra, quien hoy descansa en la paz del Señor y que ha sabido encaminarme por sendas de justicia y de amor a Dios; a mi hermana Anita por apoyarme siempre y ser como mi madre luego del fallecimiento de mi Lucita; a mi Padre y demás hermanas y hermanos, ya que me han brindado siempre su apoyo para obtener el título de Ingeniero en Mecatrónica.

*Juan Pablo*



# Resumen

En un escenario de disparo automático, el controlador puede elegir cuándo debe producirse el siguiente tiempo de muestreo y qué acción de control se mantendrá hasta que aquello suceda. El emergente control con disparo automático tiene como objetivo disminuir el uso de recursos computacionales (procesador y red), mientras se mantiene el mismo rendimiento que el obtenido a través de un controlador con muestreo periódico. Dentro de este marco, se ha desarrollado recientemente una nueva técnica de control auto-disparado, inspirada en un patrón de muestreo cuya densidad óptima minimiza el costo de control; este enfoque se denomina control auto-disparado inspirado en muestreo óptimo (OSISTC, Optimal Sampling-inspired Self-Triggered Control). Sin embargo, las estrategias utilizadas para implementarlo en sistemas reales que trabajen bajo perturbaciones, controlados por microprocesadores, aún no están claras; este documento aborda algunas pautas de implementación para hacer esta teoría aplicable sobre controladores reales. La solución propuesta comprende una nueva concepción de esta técnica en base a un observador de lazo cerrado, así como la elaboración de estrategias para la implementación de procesos computacionalmente costosos mediante polinomios ligeros ajustados en la fase de diseño. Simulaciones y experimentos prácticos confirman que la solución es efectiva y que podría haber un campo de investigación abierto relacionado con la observación en técnicas de control auto-disparado con muestreo óptimo.

# Abstract

In a self-triggered scenario the controller is allowed to choose when the next sampling time should occur and which control action will be maintained until that happens. This emerging control type is aimed at decreasing the use of computational resources (processor and network) while preserving the same performance control as the one obtained via a controller with periodic sampling. Within this framework it has been developed recently a self-triggered control technique inspired by a sampling pattern whose optimal density minimizes the control cost; this approach is called "optimal-sampling inspired self-triggered control". However the strategies used to implement it on real microprocessor-controlled systems working under disturbances are still unclear, then this paper addresses some implementation guidelines to make this theory applicable to actual controllers. The proposed solution comprises a new conception of this technique based on a closed-loop observer as well as making strategies for implementation of computationally expensive processes by lightweight polynomials fitted at design stage. Simulations and practical experiments confirm the solution is effective and there could be an open research topic concerning observation in optimal-sampling self-triggered control techniques.

# Índice general

<b>Índice general</b>	<b>XI</b>
<b>Índice de figuras</b>	<b>XIV</b>
<b>Índice de cuadros</b>	<b>XV</b>
<b>Lista de Programas</b>	<b>XVI</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Alcance . . . . .	4
1.3. Objetivos . . . . .	4
1.4. Estructura del documento . . . . .	4
<b>2. Revisión Literaria</b>	<b>6</b>
2.1. Antecedentes . . . . .	6
2.2. Diferencias entre los tipos de disparo . . . . .	8
2.2.1. Control disparado por tiempo . . . . .	8
2.2.2. Control disparado por eventos . . . . .	9
2.2.3. Control auto-disparado . . . . .	9
2.2.4. Panorama . . . . .	10
2.3. Observadores . . . . .	10
2.3.1. Estimación de estados . . . . .	11

2.3.2.	Observador de Luenberger . . . . .	12
2.3.3.	Asignación de valores propios del observador de estados . . . . .	12
<b>3.</b>	<b>Metodología</b>	<b>13</b>
3.1.	Antecedentes . . . . .	13
3.1.1.	Dinámica de tiempo continuo . . . . .	13
3.1.2.	Muestreo . . . . .	14
3.1.3.	Dinámica de tiempo discreto . . . . .	14
3.1.4.	Regulador lineal cuadrático . . . . .	15
3.1.5.	Reformulando el control auto-disparado inspirado en muestreo óptimo .	15
3.1.6.	Observador en tiempo discreto . . . . .	16
3.2.	Directrices de implementación . . . . .	17
3.2.1.	El control auto-disparado inspirado en muestreo óptimo en la práctica .	17
3.2.2.	Problemas considerados . . . . .	18
3.2.3.	Conjunto de intervalos de muestreo $T$ . . . . .	19
3.2.4.	Estrategia para calcular la matriz de ganancias del controlador $K_d(\tau_k)$ . .	20
3.2.5.	Estrategia para calcular la matriz de ganancias del observador $L_d(\tau_k)$ . .	21
3.2.6.	Directrices de implementación . . . . .	23
<b>4.</b>	<b>Implementación y pruebas</b>	<b>25</b>
4.1.	Descripción del sistema . . . . .	25
4.1.1.	Planta . . . . .	25
4.1.2.	Controlador y observador . . . . .	26
4.1.3.	Sistema operativo de tiempo real en el controlador . . . . .	27
4.2.	Implementación sobre un microprocesador . . . . .	28
4.3.	Análisis de resultados . . . . .	29
<b>5.</b>	<b>Conclusiones y trabajo futuro</b>	<b>32</b>
5.1.	Conclusiones . . . . .	32

5.2. Trabajo futuro . . . . .	32
<b>Bibliografía</b>	<b>34</b>
<b>Apéndice</b>	<b>38</b>
.A. Software . . . . .	38
.A.1. Diseño del controlador (controllerDesign.m) . . . . .	38
.A.2. Interfaz para obtener datos desde microcontrolador (MatlabInterface.m)	42
.A.3. Simulación de OSISTC (simulation.m) . . . . .	45
.B. Firmware . . . . .	50
.B.1. Código implementado sobre microcontrolador (code.c) . . . . .	50

# Índice de figuras

3.1. Observador de Luenberger de tiempo discreto . . . . .	17
3.2. Arquitectura del sistema de control de retroalimentación auto-activado con observación. Las líneas sólidas denotan señales de tiempo continuo y las líneas discontinuas indican señales actualizadas solamente en cada momento de muestreo. . . . .	17
4.1. Ajuste polinómico de las ganancias: controlador (arriba), y observador (abajo) .	27
4.2. Implementación real de los algoritmos (nótese las placas FLEX) . . . . .	29
4.3. Evolución de los estados y secuencias de muestreo de la prueba del escalón unitario: simulación (arriba), implementación real (abajo) . . . . .	30

# Índice de cuadros

4.1. Ajustes del experimento . . . . .	26
4.2. Ajustes generales para el RTOS Erika Enterprise en los DSCs . . . . .	28

# Lista de Programas

1.	Diseño del controlador . . . . .	38
2.	Interfaz para obtener datos desde microcontrolador . . . . .	42
3.	Simulación de OSISTC . . . . .	45
4.	Código implementado sobre microcontrolador . . . . .	50



# Capítulo 1

## Introducción

Este trabajo de grado ha sido realizado con el *Grupo de Investigación en Sistemas Inteligentes de la Universidad Técnica del Norte (GISI-UTN)*.

### 1.1. Motivación

Hoy en día los controladores se implementan en sistemas digitales compuestos por microprocesadores y redes de comunicación. Las funciones generales de un controlador consisten en muestrear los estados, calcular la acción de control y finalmente aplicarla sobre la planta, todo esto a lo largo del tiempo de ejecución.

A menudo las tareas de control tienen que compartir recursos de procesamiento y de red con otras tareas y debido a que su ejecución es periódica, no consideran la escasez de estos recursos usándolos de manera no óptima. Así, la ejecución de diversas tareas de control que garanticen el funcionamiento en lazo cerrado requiere microprocesadores de mayor desempeño y/o redes que trabajen a velocidades más altas.

Como alternativas para asegurar el consumo eficiente de recursos, dentro del paradigma de control no periódico, se encuentran el control disparado por eventos (ETC, Event-Triggered

Control), y el control auto-disparado (STC, Self-Triggered Control). Los dos resuelven el problema fundamental de la determinación tanto del muestreo óptimo, como de las estrategias de procesamiento/comunicación. La optimalidad se ve reflejada en: a) el costo de implementación, relacionado con el número de disparos necesarios para el procesamiento, comunicaciones y/o cambios en el actuador, b) el desempeño del controlador.

STC, propuesto por [1], [2], [3] y [4], consiste en que cada vez que se dispara la tarea de control, se estime el momento en que el siguiente muestreo se debe realizar (regla de muestreo), y también la acción de control que debe mantenerse hasta que se produzca este próximo muestreo. El diseño de la regla de muestreo sigue varias estrategias manejadas por limitaciones de control y/o restricciones de recursos. Dentro de estos paradigmas se encuentran las reglas de muestreo óptimo que están orientadas al rendimiento de control óptimo, en el sentido de producir una secuencia de entradas de control, que sea capaz de proporcionar un coste de control menor que el control óptimo. Se debe considerar este último como el correspondiente al coste mínimo producido por el regulador cuadrático lineal de tiempo discreto (LQR, Linear Quadratic Regulator), en [5].

Recientemente se han abordado varios enfoques dirigidos a resolver el problema de determinación de reglas óptimas de muestreo. Entre las obras realizadas hasta el momento, se encuentran [6], [7], [8] y [9]. El enfoque en [8], que a su vez se inspiró en un patrón de muestreo óptimo propuesto en [7], describe una regla que en cada momento de muestreo, genera entradas de control de tal manera que la señal de control a trozos resultante, se aproxima a la entrada de control LQR de tiempo continuo. La garantía de rendimiento se basa en la cantidad de muestras a lo largo de un intervalo de tiempo, con una restricción de muestreo mínimo. El tiempo de muestreo se calcula mediante la derivación de un problema LQR de tiempo continuo, y la regla produce tiempos de muestreo más pequeños mientras la acción de control tenga más variación.

Aunque el muestreo óptimo en [7] y [8] tiene un costo estándar inferior al obtenido por las técnicas de muestreo periódico, e incluso que otras técnicas de muestreo óptimo como en [9], todavía tiene muchas debilidades. Dado que la investigación es todavía nueva, hay muchos problemas abiertos entre los que destaca la ausencia de normas para su aplicación en sistemas microprocesados reales.

Para resolver el problema, en [8] se describe una simulación y una configuración experimental, que consideran una planta a ser controlada a través de cierto hardware que trabaja con un kernel de tiempo real. Sin embargo, no se establece una explicación más profunda del paradigma que un diseñador de sistemas de control debe usar, para implementar [8] sobre un sistema microprocesado.

Para superar los problemas antes mencionados, la contribución de este trabajo consiste en:

- Implementar el algoritmo en [8] sobre microcontroladores reales, sustituyendo el uso del regulador cuadrático lineal, por un polinomio ligero ajustado en la fase de diseño (fuera de línea).
- Implementar un observador de estados en lazo cerrado, como en [10], con una ganancia que varíe en cada tiempo de muestreo, mediante un polinomio ajustado fuera de línea.

Una solución al problema se presenta en [11], en la que se describe el control de una planta lineal en presencia de perturbaciones desconocidas, considerando su salida muestreada, y utilizando una estrategia de control auto-disparado. El enfoque propuesto se construye sobre la base de dos resultados previos, referentes a control auto-disparado por realimentación de estados, en [3] y [12]. Sin embargo, el STC analizado es diferente al usado como objetivo de estudio en este trabajo.

## **1.2. Alcance**

El trabajo comprende tres partes:

1. Desarrollo teórico, que consiste en la generación de un algoritmo que especifique la manera de implementar la técnica de control.
2. Simulaciones a través de herramientas de software matemático, para las cuales se desarrollan scripts en software matemático.
3. Implementación sobre microcontroladores, usando criterios referentes a tiempo real.

## **1.3. Objetivos**

El objetivo principal de este proyecto consiste desarrollar técnicas para la implementación del control auto-disparado inspirado en muestreo óptimo sobre sistemas microprocesados reales. Los siguientes objetivos específicos son también realizados:

- Abordar el estado del arte concerniente al control auto-disparado inspirado en muestreo-óptimo.
- Desarrollar el procedimiento para la implementación del control en base al fundamento teórico.
- Probar el procedimiento de implementación a través de simulaciones.
- Validar la solución obtenida mediante su implementación física sobre un sistema microprocesado real.

## **1.4. Estructura del documento**

Además de la presente introducción, el documento está conformado por cuatro capítulos y un apéndice. El Capítulo 2 presenta el estado del arte y en él se realiza una breve definición de

ETC y STC así como también una descripción detallada de OSISTC.

El marco de referencia que ha sido usado para crear la solución (firmware, software y hardware), y una descripción del algoritmo y software desarrollados, se realizan en el Capítulo 3. Esta es la parte neurálgica y más extensa del documento y se analiza desde los puntos de vista de ciencias computacionales y automática.

En el Capítulo 4 se desarrollan tanto la simulación como la implementación de la solución. Además se explica el análisis de rendimiento.

El Capítulo 5 muestra las conclusiones de este desarrollo y además se bosquejan algunas posibles líneas de trabajo futuro.

Finalmente el Apéndice muestra el código y fotografías del proyecto completo.

# Capítulo 2

## Revisión Literaria

Recientes avances en las tecnologías de la computación y de las comunicaciones han dado lugar a un nuevo tipo de sistemas de control embebidos de gran escala con limitaciones de recursos. En estos sistemas es deseable restringir la computación y/o comunicación de sensores y controladores, sólo a instancias cuando el sistema necesita atención. Sin embargo, el control clásico de datos muestreados se basa en realizar el sensado y la actuación periódicamente en lugar de cuando el sistema necesita atención. En este capítulo se describen los sistemas de control por eventos y auto-disparados en los que se realiza la detección y la actuación sólo cuando es necesario.

### 2.1. Antecedentes

En control estándar, por ejemplo [5] y [13], el control periódico se presenta como la única opción para implementar leyes de control por realimentación sobre plataformas digitales. Aunque este paradigma de control disparado por tiempo ha demostrado ser extremadamente exitoso en muchas aplicaciones de control digital, los recientes desarrollos en las tecnologías de la computación y de las comunicaciones han llevado a un nuevo tipo de sistemas de control embebido de gran escala que exigen una reconsideración de este paradigma tradicional. En particular, la creciente popularidad de los sistemas de control (compartidos) conectados por cable

e inalámbricos aumenta la importancia de abordar explícitamente las limitaciones de energía, computación y comunicación al diseñar circuitos de control de retroalimentación. Las estrategias de control aperiódico que permiten que los tiempos de interoperabilidad de las tareas de control varíen en el tiempo ofrecen ventajas potenciales con respecto al control periódico al manejar estas restricciones, pero también introducen muchos nuevos desafíos teóricos y prácticos interesantes.

Aunque las discusiones sobre la implementación periódica versus aperiódica de bucles de control de retroalimentación se remontan al principio de sistemas controlados por ordenador [14], a finales de los años 90, dos artículos influyentes [15] y [16] resaltaron las ventajas del control de retroalimentación basado en eventos. Estos dos artículos estimularon el desarrollo de los primeros diseños sistemáticos de implementaciones basadas en eventos de leyes estabilizadoras de control de retroalimentación, por ejemplo [17, 18, 19]. Desde entonces, varios investigadores han mejorado y generalizado estos resultados y han aparecido enfoques alternativos. Mientras tanto, también surgió el llamado control auto-disparado [1].

Los sistemas de control disparados por eventos y auto-disparados constan de dos elementos a saber, un controlador de realimentación que calcula la entrada de control, y un mecanismo de activación que determina cuándo la entrada de control debe ser actualizada nuevamente. La diferencia entre el control disparado por eventos y el control auto-disparado es que el primero es reactivo, mientras que el segundo es proactivo. De hecho, en el control disparado por eventos se controla continuamente una condición de disparo basada en mediciones de corriente y cuando se mantiene la condición, se desencadena un evento. En el control auto-disparado, el tiempo de actualización siguiente es precalculado en un tiempo de actualización de control basado en predicciones usando datos previamente recibidos y conocimiento de la dinámica de la planta. En algunos casos, es ventajoso combinar un control disparado por eventos y uno auto-disparado que da como resultado un sistema de control reactivo a perturbaciones impredecibles y proactivo al predecir el uso futuro de los recursos.

## 2.2. Diferencias entre los tipos de disparo

Para indicar las diferencias entre las implementaciones digitales de las leyes de control de realimentación, considere el control de la planta no lineal

$$\dot{x} = f(x, u) \quad (2.1)$$

siendo  $x \in \mathbb{R}^n$  la variable de estado y  $u \in \mathbb{R}^m$  la variable de entrada. El sistema está controlado por una ley de retroalimentación de estados no lineal

$$u = h(x) \quad (2.2)$$

donde  $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$  se implementará en una plataforma digital. El re-cálculo del valor de control y la actualización de las señales del actuador se producirán en momentos indicados por  $t_0, t_1, t_2, \dots$  con  $t_0 = 0$ . Si se asume que las entradas se mantendrán constantes entre los sucesivos re-cálculos de la ley de control (denominados como muestreo y retención u orden cero), se tiene

$$u_{(t)} = u_{(t_k)} = h(x_{(t_k)}) \quad \forall t \in [t_k, t_{k+1}), k \in \mathbb{N}. \quad (2.3)$$

Se hace referencia a los instantes  $\{t_k\}_{k \in \mathbb{N}}$  como los tiempos de disparo o tiempos de ejecución. En base a estos tiempos, se puede explicar fácilmente la diferencia entre el control disparado por tiempo, el control disparado por eventos y el control auto-disparado.

### 2.2.1. Control disparado por tiempo

En el control disparado por tiempo se tiene la igualdad  $t_k = kT_s$  siendo  $T_s > 0$  el periodo de muestreo. Por lo tanto, las actualizaciones se llevan a cabo de manera equidistante en el tiempo independientemente de cómo se comporte el sistema. No existe un mecanismo de retroalimentación en la determinación de los tiempos de ejecución; éstos son determinados a priori y en lazo abierto. Otra forma de escribir el mecanismo de disparo en el control disparado por tiempo



es

$$t_{k+1} = t_k + T_s, \quad k \in \mathbb{N} \quad (2.4)$$

con  $t_0 = 0$ .

### 2.2.2. Control disparado por eventos

En el control disparado por eventos, el siguiente tiempo de ejecución del controlador está determinado por un mecanismo de disparo de eventos que verifica continuamente si cierta condición basada en la variable de estado real se convierte en verdadera. Esta condición incluye a menudo información sobre la variable de estado  $x_{(t_k)}$  en el tiempo de ejecución anterior  $t_k$  y puede escribirse, por ejemplo, como  $C(x_{(t)}, x_{(t_k)}) > 0$ . Formalmente, los tiempos de ejecución son entonces determinados por

$$t_{k+1} = \inf\{t > t_k | C(x_{(t)}, x_{(t_k)}) > 0\} \quad (2.5)$$

con  $t_0 = 0$ . Por lo tanto, está claro a partir de (2.4) que existe un mecanismo de realimentación presente en la determinación del siguiente tiempo de ejecución ya que se basa en la variable de estado medida. En este sentido, el control desencadenado por eventos es reactivo.

### 2.2.3. Control auto-disparado

Finalmente, en el control auto-disparado el siguiente tiempo de ejecución se determina proactivamente en base al estado medido  $x_{(t_k)}$  en el tiempo de ejecución anterior. En particular, existe una función  $M : \mathbb{R}^n \rightarrow \mathbb{R} \geq 0$  que especifica el siguiente tiempo de ejecución como

$$t_{k+1} = t_k + M(x_{(t_k)}) \quad (2.6)$$

con  $t_0 = 0$ . Como consecuencia, en el control auto-disparado tanto el valor de control  $u_{(t_k)}$  como el siguiente tiempo de ejecución  $t_{k+1}$  se calculan en el tiempo de ejecución  $t_k$ . Entre  $t_k$  y  $t_{k+1}$  no se requieren más acciones del controlador. Obsérvese que la implementación tempo-

rizada puede ser vista como un caso especial de la implementación auto-disparada tomando  $M(x) = T_s \quad \forall x \in \mathbb{R}^n$ .

Claramente, en todos los tres esquemas de implementación  $T_s$ ,  $C$  y  $M$  se eligen junto con la ley de retroalimentación proporcionada por  $h$ . Esto proporciona garantías de estabilidad y rendimiento mientras se realiza cierta utilización de recursos informáticos y de comunicación.

#### **2.2.4. Panorama**

Muchos resultados experimentales y de simulación muestran que las estrategias de control disparado por eventos y auto-disparado son capaces de reducir el número de ejecuciones de las tareas de control, manteniendo al mismo tiempo un rendimiento satisfactorio en lazo cerrado. A pesar de estos resultados, el despliegue real de estos nuevos paradigmas de control en aplicaciones relevantes sigue siendo bastante marginal. Para fomentar el desarrollo futuro de controladores disparados por eventos y auto-disparados es importante validar estas estrategias en la práctica, junto a la construcción de una teoría completa del sistema. Es justo decir que a pesar de que actualmente existen muchos resultados interesantes, la teoría del sistema para el control disparado por eventos y auto-disparado está lejos de ser madura, ciertamente comparada con la vasta literatura sobre control disparado por tiempo. Así, existen todavía muchos desafíos teóricos y prácticos dentro de este atractivo campo de investigación.

### **2.3. Observadores**

Un observador es un sistema dinámico usado para estimar el estado o estados de un sistema [5]. El observador puede ser diseñado ya sea en tiempo continuo o en tiempo discreto; las características son parecidas y con propósitos de objetividad, este apartado se centrará en la observación en tiempo discreto.

### 2.3.1. Estimación de estados

El problema de estimación de estados consiste en construir una predicción  $\hat{x}_{(k)}$  del estado  $x_{(k)}$ , sólomente midiendo la salida  $y_{(k)}$  y la entrada  $u_{(k)}$ . El observador más elemental es el de lazo abierto que consiste en construir una copia artificial del sistema, alimentada en paralelo con la misma señal de entrada  $u_{(k)}$ . Esta copia es un simulador numérico  $\hat{x}_{(k+1)} = A_d \hat{x}_{(k)} + B_d u_{(k)}$  que reproduce el comportamiento del sistema real. La dinámica del sistema real y de la copia numérica es

$$\begin{cases} x_{(k+1)} = A_d x_{(k)} + B_d u_{(k)} & \text{[Sistema real]} \\ \hat{x}_{(k+1)} = A_d \hat{x}_{(k)} + B_d u_{(k)} & \text{[Copia numérica]} \end{cases} \quad (2.7)$$

La dinámica del error de estimación  $\tilde{x}_{(k)} = x_{(k)} - \hat{x}_{(k)}$  es

$$\tilde{x}_{(k+1)} = A_d \tilde{x}_{(k)} \quad (2.8)$$

y por lo tanto  $\tilde{x}_{(k)} = A_d^k (x_{(0)} - \hat{x}_{(0)})$ .

Un observador en lazo abierto no es ideal debido a:

- La dinámica del error de estimación está fijada por los valores propios de  $A_d$  y no puede ser modificada.
- El error de estimación se desvanece asintóticamente sí y sólo sí  $A_d$  es asintóticamente estable.

Nótese que  $y_{(k)}$  no es usada para calcular la estimación de estados  $\hat{x}_{(k)}$ .

### 2.3.2. Observador de Luenberger

Este observador corrige la ecuación de estimación con una retroalimentación del error de estimación  $y(k) - \hat{y}(k)$

$$\begin{cases} \hat{x}_{(k+1)} = A_d \hat{x}_{(k)} + B_d u_{(k)} + L_d [y_{(k)} - \hat{y}_{(k)}] \\ y_{(k)} = C x_{(k)} \\ \text{dado } x_0 \end{cases} \quad (2.9)$$

donde  $L_d \in \mathbb{R}^{n \times p}$  es la ganancia del observador.

La dinámica del error de estimación  $\tilde{x}_{(k)} = x_{(k)} - \hat{x}_{(k)}$  es

$$\tilde{x}_{(k+1)} = (A_d - L_d C) \tilde{x}_{(k)} \quad (2.10)$$

y entonces  $\tilde{x}_{(k)} = (A_d - L_d C)^k (x_{(0)} - \tilde{x}_{(0)})$ .

### 2.3.3. Asignación de valores propios del observador de estados

Si el par  $(A_d, C)$  es observable, entonces los valores propios de  $(A_d - L_d C)$  pueden ser ubicados arbitrariamente. Varios enunciados permiten comprobar esto:

1. Si el par  $(A_d, C)$  es completamente observable, el sistema dual  $(A'_d, C', B'_d)$  es completamente accesible.
2. Entonces se puede diseñar un compensador  $L_d$  para el sistema dual y ubicar los valores propios de  $(A'_d + C' L_d)$  arbitrariamente.
3. Los valores propios de la matriz  $(A'_d + C' L_d)$  son los valores propios de su traspuesta  $(A_d + L'_d C)$ .

# Capítulo 3

## Metodología

En este capítulo se realiza una revisión condensada del control auto-disparado inspirado en muestreo óptimo y posteriormente se establecen las directrices de implementación del mismo.

### 3.1. Antecedentes

#### 3.1.1. Dinámica de tiempo continuo

Considerérese el sistema lineal invariante de tiempo continuo descrito en representación de espacio de estados por

$$\begin{cases} \dot{x}(t) = A_c x(t) + B_c u(t) \\ y(t) = C x(t) \\ \text{dado } x(0) = x_0 \end{cases} \quad (3.1)$$

donde  $x(t) \in \mathbb{R}^n$  es el vector de estados y  $u(t) \in \mathbb{R}^m$  es la señal de entrada de control.  $A_c \in \mathbb{R}^{n \times n}$  y  $B_c \in \mathbb{R}^{n \times m}$  describen la dinámica del sistema, y  $C \in \mathbb{R}^{q \times n}$  es la matriz de pesos usada para leer los estados;  $x_0$  es el valor inicial del estado. Las variables  $m$ ,  $n$  y  $q$  denotan las dimensiones de los estados, entradas y salidas respectivamente.

### 3.1.2. Muestreo

La entrada de control  $u(t)$  en (3.1) es constante por partes, lo que significa que permanece con el mismo valor entre dos instantes de muestreo consecutivos

$$u(t) = u(k) \quad \forall t \in [t_{k-1}, t_k), \quad (3.2)$$

donde la entrada de control  $u(k)$  se actualiza a instantes discretos  $k \in \mathbb{N}$  y los instantes de muestreo están representados por  $t_k \in \mathbb{R}$ ; los instantes de muestreo consecutivos se separan por intervalos de muestreo  $\tau_k$  por lo que la relación entre instantes e intervalos se describe por

$$\tau_k = t_{k+1} - t_k, \quad t_k = \sum_{i=0}^{k-1} \tau_i \quad \text{for } k \geq 1. \quad (3.3)$$

### 3.1.3. Dinámica de tiempo discreto

En el muestreo periódico se considera un intervalo de muestreo constante  $\tau$ , entonces la dinámica de tiempo continuo de (3.1) se discretiza usando métodos tomados de [5] por

$$A_d = e^{A_c \tau}, \quad B_d = \int_0^{\tau} e^{A_c(\tau-t)} dt B_c, \quad (3.4)$$

resultando en el siguiente sistema lineal invariante de tiempo discreto:

$$\begin{cases} x_{(k+1)} = A_d x_{(k)} + B_d u_{(k)} \\ y_{(k)} = C x_{(k)} \end{cases}, \quad \text{given } x_{(0)} = x_0 \quad (3.5)$$

donde el estado  $x_{(k)}$  es muestreado cada  $t_k$ .

La ubicación de los polos del sistema (o valores propios de la matriz de dinámica cuando se usa la representación del espacio de estados) es fundamental para determinar/cambiar la estabilidad del sistema. Los polos en tiempo continuo  $p_c$  se convierten en polos en tiempo

discreto  $p_d$  a través de

$$p_d = e^{p_c * \tau}, \quad (3.6)$$

técnica también tomada de [5].

### 3.1.4. Regulador lineal cuadrático

El problema de control óptimo LQR permite encontrar una señal de entrada óptima que minimice las funciones de costo en tiempo continuo y discreto en horizonte infinito de (3.7) y (3.8) respectivamente, como en

$$J_c = \int_0^{\infty} (x_{(t)}^T Q_c x_{(t)} + 2x_{(t)}^T S_c u_{(t)} + u_{(t)}^T R_c u_{(t)}) dt, \quad (3.7)$$

$$J_d = \sum_0^{\infty} (x_{(k)}^T Q_d x_{(k)} + 2x_{(k)}^T S_d u_{(k)} + u_{(k)}^T R_d u_{(k)}). \quad (3.8)$$

Atendiendo a la dimensionalidad de (3.7) y (3.8), las matrices de peso  $Q_c, Q_d \succeq 0 \in \mathbb{R}^{n \times n}$  son positivas semidefinidas,  $R_c, R_d \succ 0 \in \mathbb{R}^{m \times m}$  son positivas definidas, y  $S_c, S_d \in \mathbb{R}^{n \times m}$ .

### 3.1.5. Reformulando el control auto-disparado inspirado en muestreo óptimo

El enfoque en [8] implica diseñar una regla de muestreo y una entrada de control por partes al mismo tiempo, de manera que el costo LQR se minimice. Además, la periodicidad de ejecución del controlador se reduce, disminuyendo así el consumo de recursos. La *regla de muestreo* es

$$\tau_k = \tau_{max} \frac{1}{\frac{\tau_{max}}{\eta} |K_c(A_c + B_c K_c)x_{(k)}|^{\alpha} + 1} \quad (3.9)$$

donde el límite superior en los intervalos de muestreo está dado por  $\tau_{max}$ ; de manera similar  $\eta$  modifica el grado de densidad de la secuencia de muestreo ( $\eta$  más pequeño produce instantes de muestreo más densos y viceversa). Minimizando la función de costo de tiempo continuo (3.7) se obtiene una ganancia de retroalimentación continua óptima  $K_c$  una sola vez. De acuerdo con [7]

y [8] existen configuraciones óptimas para el exponente  $\alpha \geq 0$  que influyen en la densidad del conjunto de muestras; con  $\alpha = 0$  el muestreo se hace regular (periódico).

Adicionalmente, en [8] la *señal de control óptima por partes* expresada en forma de retroalimentación de estado lineal es

$$u_{(k)} = -K_{d(\tau_k)}x_{(k)}, \quad (3.10)$$

donde  $K_{d(\tau_k)}$  es calculado en cada ejecución del controlador. Su valor es obtenido resolviendo el problema LQR de tiempo discreto con (3.8), considerando un período de muestreo fijo  $\tau_k$ .

### 3.1.6. Observador en tiempo discreto

Para implementar cualquier controlador de realimentación de estados como el que se ve en (3.10) se requiere conocimiento completo del vector de estados  $x_{(k)}$ , sin embargo, a menudo los sensores sólo proporcionan mediciones de la salida  $y_{(k)}$  sin considerar perturbaciones, como se muestra en (3.5). Por lo tanto, aparece la necesidad de reconstruir la información faltante de las variables de estado; esto se logra mediante la estimación del estado  $x_{(k)}$ , midiendo solamente la salida  $y_{(k)}$  y conociendo la entrada de control  $u_{(k)}$  aplicada al sistema.

Un observador constituye una copia informática del sistema dinámico alimentada en paralelo por la misma señal  $u_{(k)}$ ; por ejemplo el observador de Luenberger [10] es un estimador de estados que funciona correctamente en presencia de perturbaciones desconocidas. Véase [5] para una mejor comprensión.

El observador de Luenberger de la Fig. 3.1 es ampliamente usado para corregir la estimación  $\hat{x}_{(k+1)}$  con una retroalimentación del error de salida del observador  $\tilde{e}_{(k)}$ . La ecuación deducida es

$$\hat{x}_{(k+1)} = A_d\hat{x}_{(k)} + B_d u_{(k)} + L_d [y_{(k)} - \hat{y}_{(k)}], \quad (3.11)$$

donde  $\hat{x}_{(k+1)} \in \mathbb{R}^n$  es el estado estimado y  $\hat{y}_{(k)} \in \mathbb{R}^q$  es la salida estimada;  $A_d$ ,  $B_d$ , y  $C$  tienen las



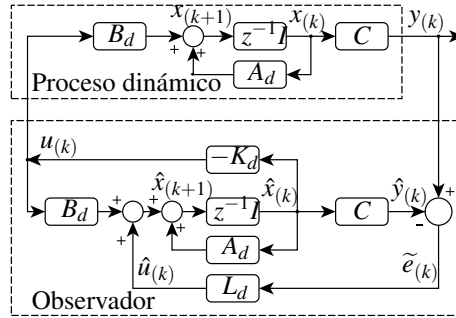


Figura 3.1: Observador de Luenberger de tiempo discreto

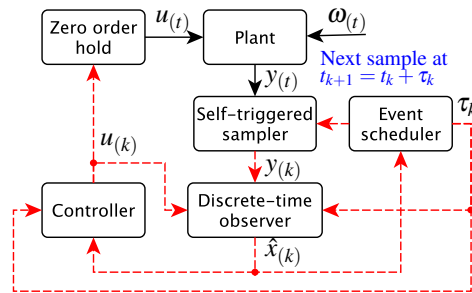


Figura 3.2: Arquitectura del sistema de control de retroalimentación auto-activado con observación. Las líneas sólidas denotan señales de tiempo continuo y las líneas discontinuas indican señales actualizadas solamente en cada momento de muestreo.

dimensiones apropiadas y se explican en mayor detalle a través de (3.4)–(3.5).  $L_d \in \mathbb{R}^{n \times q}$  es la matriz de ganancias del observador.

## 3.2. Directrices de implementación

### 3.2.1. El control auto-disparado inspirado en muestreo óptimo en la práctica

La teoría sobre OSISTC no considera ruido ( $\omega(t)$  en la Fig. 3.2), por lo que es necesario utilizar algunas estrategias adicionales para su aplicación práctica en una planta bajo perturbaciones.

La Fig. 3.2 muestra la arquitectura de control auto-disparado propuesta, en la que destaca

el uso de un *observador de tiempo discreto*. Esta configuración tiene como desventaja que el observador necesita resolver una nueva ubicación de polos en cada ejecución, ya que las matrices de la dinámica discreta y los polos discretos son dependientes del intervalo de muestreo  $\tau_k$ . Esto implica que el observador tenga una matriz de ganancia diferente  $L_d$  en cada ejecución. Considerando la dinámica cambiante, el sistema en (3.11) se convierte en

$$\hat{x}_{(k+1)} = A_{d(\tau_k)}\hat{x}_{(k)} + B_{d(\tau_k)}u_{(k)} + L_{d(\tau_k)} [y_{(k)} - \hat{y}_{(k)}], \quad (3.12)$$

donde  $A_{d(\tau_k)}$  y  $B_{d(\tau_k)}$  son matrices discretizadas para un intervalo de muestreo  $\tau_k$ ,  $u_{(k)}$  es la acción de control linear a trozos calculada a través de la estrategia de auto-disparo en (3.10), y  $L_{d(\tau_k)} \in \mathbb{R}^{n \times q}$  es la matriz de ganancias del observador dependiente del muestreo.

### 3.2.2. Problemas considerados

Hay varios inconvenientes al ensamblar tanto el controlador OSISTC como el observador variable en el tiempo, en un sistema de control en tiempo real.

El primer problema tiene que ver con el cálculo de la *matriz de ganancias del controlador*  $K_{d(\tau_k)}$  in (3.10) resolviendo el problema LQR discreto en (3.8) a través del cálculo recursivo de la ecuación algebraica discreta de Ricatti equation (DARE, Discrete Algebraic Ricatti Equation) hasta su convergencia [5]. El segundo problema es la ubicación de polos resuelta por la fórmula de Ackermann [20] con el fin de obtener la *matriz de ganancias del observador*  $L_{d(\tau_k)}$ .

Ambos procesos son costosos desde el punto de vista computacional y deben realizarse en cada ejecución del controlador; son difíciles de implementar en tiempo real en un sistema embebido con baja capacidad computacional. Si el tiempo de ejecución de la tarea de control está demasiado cerca del intervalo mínimo de muestreo, aparecen efectos indeseables como el jitter [17]. En particular, en OSISTC se presenta el peor escenario cuando la tasa de cambio de la acción de control es máxima, lo que causa una densidad más alta en la aparición de muestras

(mínimo  $\tau_k$ ).

Para resolver los problemas anteriores se propone usar ciertos mecanismos fuera de línea que produzcan aproximaciones a ser usadas en línea, con el fin de optimizar la utilización del procesador.

### 3.2.3. Conjunto de intervalos de muestreo $T$

La estrategia propuesta incluye el cálculo fuera de línea de un conjunto de todas las posibles matrices de ganancia del controlador  $K_d(\tau_k)$ , para luego inferir y usar funciones polinomiales para imitar su comportamiento.

El primer paso se basa en la descripción del *conjunto de intervalos de muestreo*  $T \in \mathbb{R}^{1 \times s}$  dentro de un intervalo cerrado  $[\tau_{min}, \tau_{max}]$  como sigue

$$T = \{\tau_{min}, \tau_{min} + \tau_g, \tau_{min} + 2\tau_g, \dots, \tau_{max}\} \quad (3.13)$$

donde  $\tau_g$  es la *granularidad de muestreo* definida como la unidad de menor incremento para los intervalos de muestreo. Cada elemento del conjunto  $T$  puede ser direccionado en esta manera

$$\tau_h = T_{[h]} \quad \forall h \in \mathbb{N} : 1 \leq h \leq s \quad (3.14)$$

siendo  $s$  el largo de  $T$ .

Los tiempos de muestreo mínimo y máximo,  $\tau_{min}$  y  $\tau_{max}$ , además de  $\tau_g$ , son escogidos siguiendo las condiciones detalladas en [8]

$$\begin{cases} \frac{\beta^\alpha}{\eta} \leq \frac{1}{\tau_{min}} - \frac{1}{\tau_{max}}, & \beta := \sup_{x \in X} |K_c(A_c + B_c K_c)x| \\ \tau_{RTOS} \leq \tau_g \end{cases} \quad (3.15)$$

donde  $X$  es el espacio entero del estado tomado de la restricción física de la planta, y  $\tau_{RTOS}$  es la granularidad de muestreo del sistema operativo de tiempo real (RTOS, Real Time Operating

System) en el cual el algoritmo será implementado.

### 3.2.4. Estrategia para calcular la matriz de ganancias del controlador

$$K_d(\tau_k)$$

La matriz  $K_d(\tau_h)$  es calculada por fuerza bruta para cada elemento  $h$  en el conjunto de intervalos de muestreo  $T$  en (3.13) direccionado por (3.14). Lo anterior comprende cada vez primero calcular las matrices discretas  $A_d(\tau_h)$ ,  $B_d(\tau_h)$ ,  $Q_d(\tau_h)$  y  $R_d(\tau_h)$  por (3.4) con las cuales el problema LQR de tiempo discreto es resuelto posteriormente; lo anterior estipula la minimización de la función de costo en (3.8). Por lo tanto, se obtiene un total de  $s$  matrices de ganancias de control del tipo  $K_d(\tau_h) \in \mathbb{R}^{m \times n}$  ya que son evaluadas para cada uno de los posibles  $s$  valores de  $\tau_h$  dentro del conjunto  $T$ . Estas matrices tienen la forma

$$K_d(\tau_h) = dlqr(A_d(\tau_h), B_d(\tau_h), Q_d(\tau_h), R_d(\tau_h)) = \begin{bmatrix} kd_{11}^{(\tau_h)} & \dots & kd_{1n}^{(\tau_h)} \\ \vdots & \ddots & \vdots \\ kd_{m1}^{(\tau_h)} & \dots & kd_{mn}^{(\tau_h)} \end{bmatrix} \quad (3.16)$$

donde el superíndice ( $\tau_h$ ) indica la pertenencia del elemento correspondiente de ganancia a la matriz  $K_d(\tau_h)$ .

Si los elementos de todas las matrices de ganancia se reagrupan según su posición se tiene un grupo  $S_{K_d}$ , de  $m \cdot n$  conjuntos de entrenamiento de largo, donde  $n$  y  $m$  son las dimensiones de los estados y las entradas respectivamente, entonces

$$S_{K_d} = \{[kd_{11}^{(\tau_1)}, \dots, kd_{11}^{(\tau_s)}], \dots, [kd_{1n}^{(\tau_1)}, \dots, kd_{1n}^{(\tau_s)}], \dots, [kd_{m1}^{(\tau_1)}, \dots, kd_{m1}^{(\tau_s)}], \dots, [kd_{mn}^{(\tau_1)}, \dots, kd_{mn}^{(\tau_s)}]\}. \quad (3.17)$$

Cada conjunto de entrenamiento en (3.17) está definido en  $\mathbb{R}^{1 \times s}$  y es usado para desarrollar un *ajuste de curva polinómico* para encontrar los coeficientes  $\theta$  de los polinomios de grado  $d$ ,

$K_{ij}(\tau_k)$ . Por lo tanto, se tiene un total de  $m \cdot n$  polinomios, cada uno siguiendo la forma

$$K_{ij}(\tau_k) = \theta_1^{(ij)} \tau_k^d + \theta_2^{(ij)} \tau_k^{d-1} + \dots + \theta_d^{(ij)} \tau_k + \theta_{d+1}^{(ij)}, \quad (3.18)$$

donde el subíndice  $(ij)$  indica la pertenencia de los coeficientes  $\theta$  al polinomio  $K_{ij}(\tau_k)$ ; las filas  $i$  y las columnas  $j$  denotan la posición de los polinomios en la matriz de ganancias. Note el uso de  $\tau_k$  en lugar de  $\tau_h$  ya que el primero será el intervalo de muestreo actual calculado en línea por la ecuación (3,9) en el controlador real. Así, (3.16)–(3.18) se convierten en

$$K_d(\tau_k) = \begin{bmatrix} K_{11}(\tau_k) & \dots & K_{1n}(\tau_k) \\ \vdots & \ddots & \vdots \\ K_{m1}(\tau_k) & \dots & K_{mn}(\tau_k) \end{bmatrix}, \quad (3.19)$$

donde

$$\begin{aligned} K_{11}(\tau_k) &= \theta_1^{(11)} \tau_k^d + \theta_2^{(11)} \tau_k^{d-1} + \dots + \theta_d^{(11)} \tau_k + \theta_{d+1}^{(11)} \\ &\dots \\ K_{1n}(\tau_k) &= \theta_1^{(1n)} \tau_k^d + \theta_2^{(1n)} \tau_k^{d-1} + \dots + \theta_d^{(1n)} \tau_k + \theta_{d+1}^{(1n)} \\ &\dots \\ K_{m1}(\tau_k) &= \theta_1^{(m1)} \tau_k^d + \theta_2^{(m1)} \tau_k^{d-1} + \dots + \theta_d^{(m1)} \tau_k + \theta_{d+1}^{(m1)} \\ &\dots \\ K_{mn}(\tau_k) &= \theta_1^{(mn)} \tau_k^d + \theta_2^{(mn)} \tau_k^{d-1} + \dots + \theta_d^{(mn)} \tau_k + \theta_{d+1}^{(mn)}. \end{aligned}$$

### 3.2.5. Estrategia para calcular la matriz de ganancias del observador $L_d(\tau_k)$

Éste es un proceso similar al descrito en la subsección 3.2.4. Todas las posibles matrices de ganancias del observador  $L_d(\tau_k)$  son evaluadas fuera de línea como funciones de los intervalos de muestreo  $\tau_h$ .

La dinámica del error del observador está dada por los polos de  $[A_d(\tau_h) - L_d(\tau_h)C]$ . Una regla general consiste en colocar los polos del observador por lo menos cinco a diez veces más hacia la izquierda del plano-s que los polos dominantes del sistema. Por lo tanto, la ubicación de los polos de tiempo continuo se asigna estáticamente y se discretizan utilizando (3.6) para todos los  $\tau_h$ , obteniendo  $P_d(\tau_h)$ .

Calculando nuevamente  $A_d(\tau_h)$  mediante (3.4) para todos los  $\tau_h$ , asignando estáticamente los polos en tiempo continuo y discretizándolos mediante (3.6) también para todos los  $\tau_h$  para obtener  $P_d(\tau_h)$ , y finalmente considerando  $C$  que permanece constante, se obtiene un total de  $s$  matrices de ganancias del observador  $L_d(\tau_h) \in \mathbb{R}^{n \times q}$  por el método de ubicación de polos en [20], con la forma

$$L_d(\tau_h) = \text{ackermann} \left( A_d(\tau_h)^T, C^T, P_d(\tau_h)^T \right) = \begin{bmatrix} ld_{11}^{(\tau_h)} & \dots & ld_{1q}^{(\tau_h)} \\ \vdots & \ddots & \vdots \\ ld_{n1}^{(\tau_h)} & \dots & ld_{nq}^{(\tau_h)} \end{bmatrix}. \quad (3.20)$$

Usando el criterio de reagrupación de (3.17) se obtiene un grupo  $S_{L_d}$  conformado por  $n \cdot q$  conjuntos de datos para entrenamiento

$$S_{L_d} = \{ [ld_{11}^{(\tau_1)}, \dots, ld_{11}^{(\tau_s)}], \dots, [kd_{1q}^{(\tau_1)}, \dots, kd_{1q}^{(\tau_s)}], \dots, [kd_{n1}^{(\tau_1)}, \dots, kd_{n1}^{(\tau_s)}], \dots, [kd_{nq}^{(\tau_1)}, \dots, kd_{nq}^{(\tau_s)}] \}. \quad (3.21)$$

Subsecuentemente un total de  $n \cdot q$  polinomios se calcula con la forma

$$L_{ij}(\tau_k) = \theta_1^{(ij)} \tau_k^d + \theta_2^{(ij)} \tau_k^{d-1} + \dots + \theta_d^{(ij)} \tau_k + \theta_{d+1}^{(ij)}, \quad (3.22)$$

tal como en (3.18). Finalmente, usando (3.22) se obtiene

$$L_d(\tau_k) = \begin{bmatrix} L_{11}(\tau_k) & \dots & l_{1q}(\tau_k) \\ \vdots & \ddots & \vdots \\ L_{n1}(\tau_k) & \dots & l_{nq}(\tau_k) \end{bmatrix}, \quad (3.23)$$

donde

$$\begin{aligned}
L_{11}(\tau_k) &= \theta_1^{(11)} \tau_k^d + \theta_2^{(11)} \tau_k^{d-1} + \dots + \theta_d^{(11)} \tau_k + \theta_{d+1}^{(11)} \\
&\dots \\
L_{1q}(\tau_k) &= \theta_1^{(1q)} \tau_k^d + \theta_2^{(1q)} \tau_k^{d-1} + \dots + \theta_d^{(1q)} \tau_k + \theta_{d+1}^{(1q)} \\
&\dots \\
L_{n1}(\tau_k) &= \theta_1^{(n1)} \tau_k^d + \theta_2^{(n1)} \tau_k^{d-1} + \dots + \theta_d^{(n1)} \tau_k + \theta_{d+1}^{(n1)} \\
&\dots \\
L_{nq}(\tau_k) &= \theta_1^{(nq)} \tau_k^d + \theta_2^{(nq)} \tau_k^{d-1} + \dots + \theta_d^{(nq)} \tau_k + \theta_{d+1}^{(nq)}.
\end{aligned}$$

Entonces, en cada ejecución del controlador real, después de calcular el siguiente intervalo de muestreo  $\tau_k$  mediante (3.9), cada elemento de la matriz de ganancia del observador se calcula a través de un polinomio diferente en la matriz  $L_d(\tau_k)$ .

### 3.2.6. Directrices de implementación

A través del Algoritmo 1 se resume lo que se dijo en las subsecciones 3.2.4 y 3.2.5; este programa puede ser realizado fuera de línea por cualquier programa de computación numérica. El Algoritmo 2 muestra cómo implementar OSISTC en cualquier procesador que tenga características de rendimiento reducido.

**Data:**  $A_c, B_c, C, Q_c, R_c, \tau_{min}, \tau_{max}, \tau_g, \alpha, \beta, \eta, P_c$   
**Result:**  $K_c, K_d(\tau_k), L_d(\tau_k)$   
 $K_c = f(A_c, B_c, Q_c, R_c)$  por LQR continuo (3.7);  
 $T = f(\tau_{min}, \tau_{max}, \tau_g)$  por (3.13) y (3.15);  
**for**  $\tau_h \in T$  **do**  
    Calcula  $A_d(\tau_h), B_d(\tau_h), Q_d(\tau_h), R_d(\tau_h)$  por (3.4);  
     $K_d(\tau_h) = f(A_d(\tau_h), B_d(\tau_h), Q_d(\tau_h), R_d(\tau_h))$  por LQR discreto (3.16);  
    Calcula  $P_d(\tau_h)$  por (3.6);  
     $L_d(\tau_h) = f(A_d^T(\tau_h), C^T, P_d(\tau_h))$  por Ackermann (3.20);  
**end**  
Crea  $S_{K_d}$  basado en todos los  $K_d(\tau_h)$  por (3.17);  
Crea  $S_{L_d}$  basado en todos los  $L_d(\tau_h)$  por (3.21);  
**for**  $i \leq m$  **and**  $j \leq n$  **do**  
     $K_{ij}(\tau_k) = f(S_{K_d(i,j)})$  por ajuste de curva polinómico;  
**end**  
**for**  $i \leq n$  **and**  $j \leq q$  **do**  
     $L_{ij}(\tau_k) = f(S_{L_d(i,j)})$  por ajuste de curva polinómico;  
**end**  
Crea  $K_d(\tau_k)$  al ordenar todos los  $K_{ij}(\tau_k)$  como en (3.19);  
Crea  $L_d(\tau_k)$  al ordenar todos los  $L_{ij}(\tau_k)$  como en (3.23);  
**Algoritmo 1:** Diseño fuera de línea

**Data:**  $A_c, B_c, C, K_c, \tau_{max}, \tau_g, \alpha, \beta, \eta, K_d(\tau_k), L_d(\tau_k)$   
**Result:**  $\tau_k, u(k)$   
Inicializa hardware, RTOS y variables;  
 $x(k) := lee\_input();$   
 $\tau_k := f(\tau_{max}, K_c, \alpha, \beta, \eta, \hat{x}(k))$  por (3.9);  
Configura RTOS para disparar el siguiente tiempo luego de  $\tau_k$ ;  
Calcula  $K_d(\tau_k)$  por (3.19);  
 $u(k) := f(K_d(\tau_k), \hat{x}(k))$  por (3.10);  
Ajusta actuador con  $u(k)$ ;  
Calcula  $L_d(\tau_k)$  por (3.23);  
Calcula  $A_d(\tau_k), B_d(\tau_k)$  por (3.4);  
 $\hat{x}(k) := f(A_d(\tau_k), B_d(\tau_k), K_d(\tau_k), L_d(\tau_k), \hat{x}(k), x(k))$  por (3.12)  
**Algoritmo 2:** Implementación en línea



# Capítulo 4

## Implementación y pruebas

En este capítulo, la teoría desarrollada hasta ahora es comprobada a través de un ejemplo numérico, específicamente sobre un circuito electrónico doble integrador. Se describe tanto la metodología adoptada como las matemáticas utilizadas. Además, se explican y analizan las pruebas para corroborar su funcionamiento adecuado.

### 4.1. Descripción del sistema

Se presenta un experimento sobre una planta real para ilustrar la teoría introducida en la sección anterior.

#### 4.1.1. Planta

Como planta experimental con forma (1) se utiliza un circuito electrónico de doble integración (críticamente estable); se aconseja usar [4] para más información. La representación en el espacio de estados es

$$A_c = \begin{bmatrix} 0 & -23,81 \\ 0 & 0 \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ -23,81 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \quad (4.1)$$

En la Tabla 4.1 se detallan las configuraciones más importantes utilizadas para diseñar tanto

Tabla 4.1: Ajustes del experimento

Parámetro	Valor
$Q_c$	$0,0025 \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
$R_c$	0,1
$\tau_{min}$	15ms
$\tau_{max}$	50ms
$\tau_g$	1ms
$\alpha$	0,667
$\beta$	9,4
$\eta$	0,11
$P_c$	$[-5 + 2j, -5 - 2j]$

el controlador como el observador. Estos valores se han basado en las recomendaciones de la literatura en [8].

#### 4.1.2. Controlador y observador

En la Fig. 4.1 las curvas ajustadas (líneas continuas) describen aproximadamente el comportamiento de las ganancias. Las ganancias tanto del controlador como del observador evaluadas para el conjunto de intervalos de muestreo se muestran mediante círculos. A continuación se resumen los resultados numéricos adicionales:

- Ganancia del controlador en tiempo continuo:  $K_c = [0,1581, -0,5841]$ .

- La matriz de ganancias del controlador está formada de dos polinomios:

$$K_d(\tau_k) = [K_{11}(\tau_k), K_{12}(\tau_k)] \text{ donde } K_{11}(\tau_k) = -8,8668\tau_k + 0,1548 \text{ y } K_{12}(\tau_k) = 1,8819\tau_k - 0,5799.$$

- La matriz de ganancias del observador está formada de dos polinomios:

$$L_d(\tau_k) = [L_{11}(\tau_k), L_{21}(\tau_k)]^T \text{ donde } L_{11}(\tau_k) = 8,7071\tau_k + 0,0185 \text{ y } L_{21}(\tau_k) = -0,8751\tau_k - 0,0048.$$

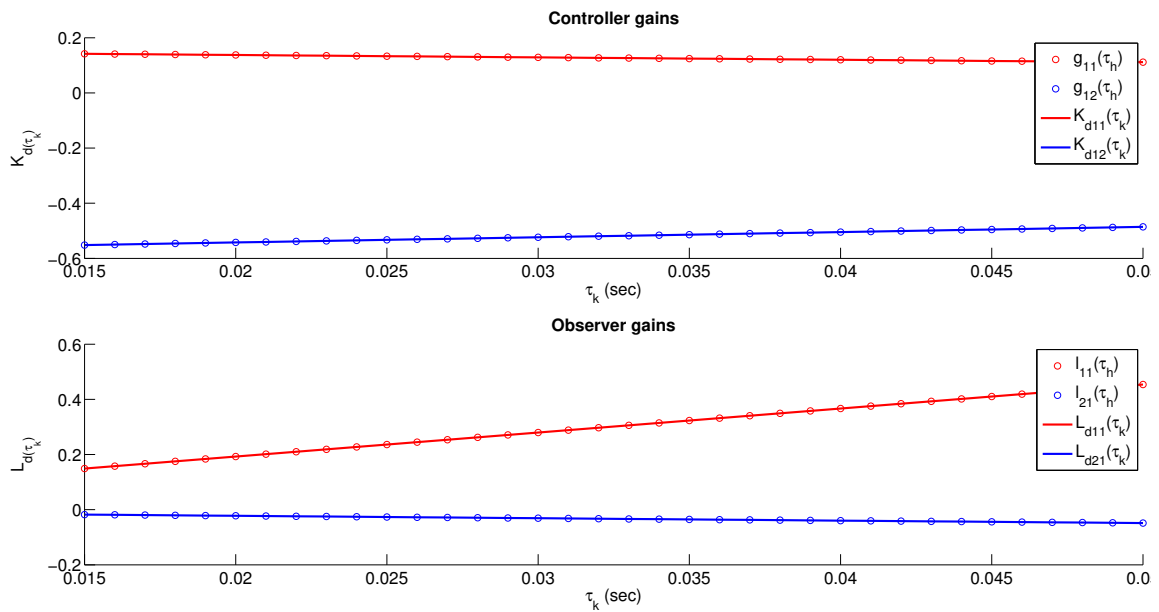


Figura 4.1: Ajuste polinómico de las ganancias: controlador (arriba), y observador (abajo)

### 4.1.3. Sistema operativo de tiempo real en el controlador

La estructura multitarea requiere procesos de dibujo bajo el paradigma de una política de programación. Así, en el microcontrolador existirán varios procesos activados secuencialmente, permaneciendo latentes incluso cuando no se están ejecutando.

El kernel multitarea de tiempo real *Erika Enterprise*, usado aquí, fue descrito brevemente en el capítulo 2. La Tabla 4.2 puede ser usada para conocer las configuraciones generales del RTOS implementado en el DSC.

Tabla 4.2: Ajustes generales para el RTOS Erika Enterprise en los DSCs

Parámetro	Ajuste	Descripción
Manejo de pilas	Monopila	Sólo hay una pila en el sistema. No se admiten primitivas de bloqueo y todas las tareas e interrupciones se ejecutan en la misma pila.
Manejo de interrupciones	ISRs <sup>1</sup> rápidas	Existen ISRs regulares que pueden llamar a primitivas RTOS (por ejemplo, la recepción CAN o UART puede llamar al comando <i>ActivateTask</i> para activar una tarea periódica). Estos ISR siempre tienen una gran prioridad de interrupción de hardware.
Tipo de kernel	EDF <sup>2</sup> o menor tiempo para ejecutarse	Siempre que se produzca un evento planificado (finalización de tareas, nueva tarea liberada, etc.) se buscará en la cola el proceso con tiempo límite más cercano. Este proceso es el siguiente en ejecutarse.
Petición de interrupción	Nested	Una interrupción de mayor prioridad puede ser atendida antes de completar la rutina de servicio de interrupción actual.
Largo de período mínimo de ejecución (tick)	25 nanosegundos	Esta referencia de temporización depende de la configuración del reloj. En este caso: $F_{cy} = 40$ MIPS, $Tick_{length} = 1/F_{cy} = 25ns$ .

## 4.2. Implementación sobre un microprocesador

La plataforma de desarrollo (véase la Fig. 4.2) comprende el controlador digital de señales (DSC) dsPIC33FJ256MC710A de Microchip el cual internamente ejecuta el kernel de tiempo real Erika Enterprise. Para conocer más acerca de este ambiente se recomienda revisar el trabajo original en [22] y sus referencias.

El controlador auto-activado usa la regla (3.9) para calcular cuándo se activará la próxima vez; este valor se utiliza para configurar el RTOS para activar el siguiente instante de muestreo. Otras funciones del controlador son leer los estados de la planta  $x_{(k)}$  a través del convertidor analógico/digital del DSC, estimar los estados  $\hat{x}_{(k)}$  a través del observador y calcular la acción de control  $u_{(k)}$  que se aplica directamente a la planta mediante modulación de ancho de pulso (PWM).

<sup>1</sup>Interrupt Service Routines

<sup>2</sup>Earliest Deadline First

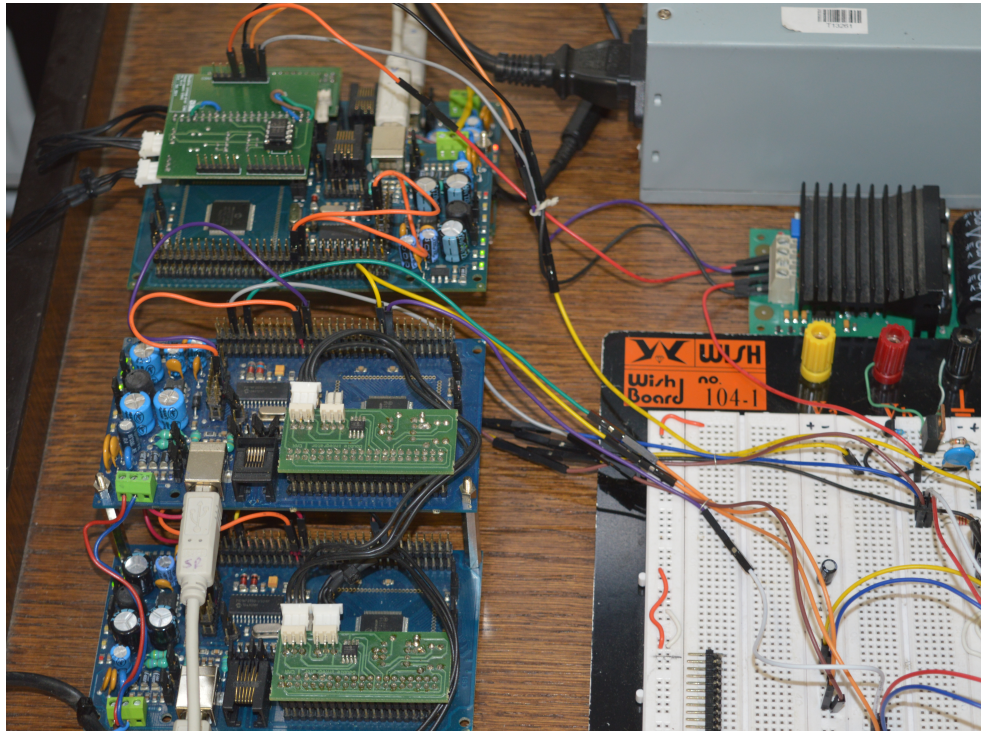


Figura 4.2: Implementación real de los algoritmos (nótese las placas FLEX)

La matriz de ganancias del controlador, en lugar de minimizar la DARE, usa dos polinomios de primer grado que son funciones de  $\tau_k$  y están representados como  $K_{11}(\tau_k)$  y  $K_{12}(\tau_k)$ , agrupados en  $K_d(\tau_k)$ .

Finalmente, en lugar de usar un método de ubicación de polos, por ejemplo Ackermann, la matriz de ganancias del observador es reemplazada por el par de polinomios de primer grado  $L_{11}(\tau_k)$  y  $L_{21}(\tau_k)$ , enmarcados dentro de  $L_d(\tau_k)$ .

### 4.3. Análisis de resultados

La Figura 4.3 muestra la evolución de los estados y el patrón de muestreo tanto en la simulación como en la implementación real, cuando el OSISTC se somete a un paso unitario. Los tiempos de establecimiento, los rebasamientos y los errores de estado estacionario son muy

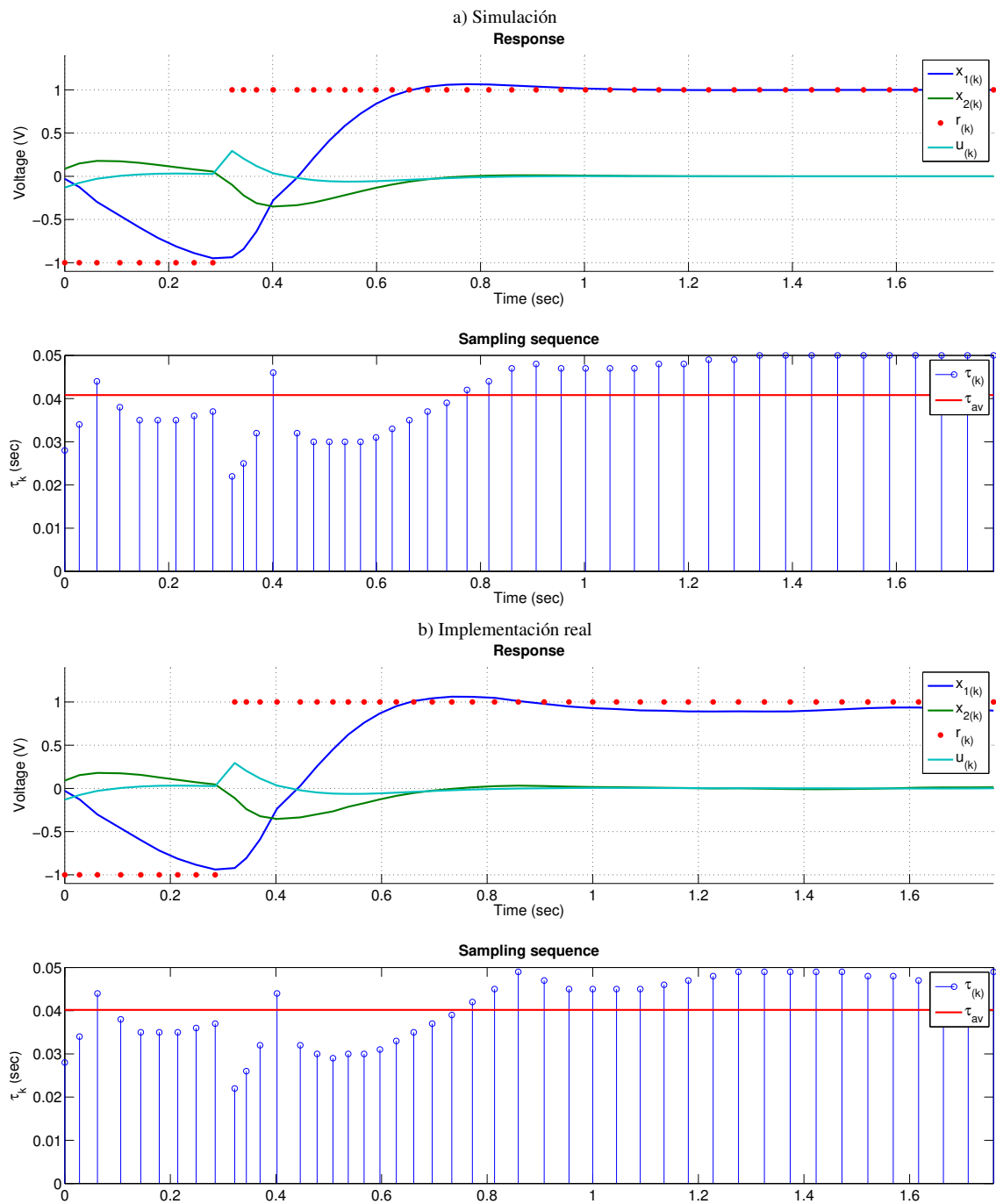


Figura 4.3: Evolución de los estados y secuencias de muestreo de la prueba del escalón unitario: simulación (arriba), implementación real (abajo)

cercanos.

Los intervalos de muestreo (3.15) en la simulación se mantienen dentro del rango  $[22, 50]$ ms, mientras en el sistema real están dentro de  $[21, 50]$ ms. En este último, cuando el sistema se aproxima al estado estacionario, los intervalos de muestreo  $\tau_{(k)}$  tienden a oscilar alrededor de  $\tau_{max}$ . Se aseguró que los dos se encuentren dentro del rango  $[\tau_{min}, \tau_{max}]$ .

La métrica de muestreo promedio  $\tau_{av}$  [8] establece

$$\tau_{av} = \frac{1}{N} \sum_{k=0}^{N-1} \tau_k, \quad (4.2)$$

donde  $N$  es el número de muestras dentro del tiempo de experimento/simulación; valores grandes de  $\tau_{av}$  indican menor utilización de recursos. En la simulación  $\tau_{av} = 40,82$ ms y en la implementación  $\tau_{av} = 39,96$ ms. Su diferencia pequeña se debe a incertidumbre.

# Capítulo 5

## Conclusiones y trabajo futuro

Este capítulo muestra las conclusiones del presente proyecto y esboza algunas líneas posibles de trabajo futuro.

### 5.1. Conclusiones

Se presentaron algunas pautas de implementación para hacer la teoría en [8] aplicable a controladores reales. Para ello, la implementación del algoritmo sobre controladores reales se realizó reemplazando el uso en línea del problema LQR por un polinomio ligero instalado en la fase de diseño (fuera de línea). Además, un observador de lazo cerrado variable en el tiempo se ha implementado mediante técnicas de ajuste polinómico, evitando al mismo tiempo el uso en línea de métodos de colocación de polos como Ackermann. La coherencia entre la teoría y la práctica ha sido demostrada para que la implementación pueda ser asumida como efectiva.

### 5.2. Trabajo futuro

Simulaciones y experimentos prácticos sobre un procesador real confirman que la solución es efectiva y podría haber un tema de investigación abierto con respecto a las técnicas de observación en OSISTC. Hay interesantes medidas de desempeño en la literatura que podrían



convertirse en trabajo futuro para esta implementación; las métricas de [8] y [23] permitirían una mejor evaluación de un sistema real. Se puede hacer una comparación entre la implementación con y sin observador para determinar la verdadera contribución de esta última. Además, se podría sustituir al observador de Luenberger por otro observador con información de incertidumbre como Kalman.

# Bibliografía

- [1] M. Velasco, J.M. Fuertes, and P. Marti, “The Self Triggered Task Model for Real-Time Control Systems”, in *Proc. IEEE Real-Time Systems Symposium*, pp. 67-70, 2003.
- [2] A. Anta and P. Tabuada, “To Sample or Not to Sample: Self-Triggered Control for Non-linear Systems”, *IEEE Transactions on Automatic Control*, vol. 55, no. 9, pp. 2030-2042, 2010.
- [3] M. Mazo Jr., A. Anta and P. Tabuada, “An ISS Self-Triggered Implementation of Linear Controllers”, in *Automatica*, vol. 46, pp. 1310-1314, 2010.
- [4] J. Almeida, C. Silvestre, A.M. Pascoal, “Self-Triggered Output Feedback Control of Linear Plants”, in *American Control Conference*, pp. 2831–2836, 2011.
- [5] K.J. Åström and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*, Prentice Hall, third edition, 1997.
- [6] A. Molin and S. Hirche, “On the Optimality of Certainty Equivalence for Event-Triggered Control Systems”, in *IEEE Transactions on Automatic Control*, vol. 58, no. 2, pp. 470-474, February 2013.
- [7] E. Bini and G.M. Buttazzo, “The Optimal Sampling Pattern For Linear Control Systems”, in *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 78-90, January 2014.

- [8] M. Velasco, P. Martí, and E. Bini, “Optimal-Sampling-Inspired Self-Triggered Control”, in *1st IEEE International Conference on Event-based Control, Communication, and Signal Processing*, Krakow, Poland, June 2015.
- [9] T. Gommans, D. Antunes, T. Donkers, P. Tabuada, and M. Heemels, “Self-Triggered Linear Quadratic Control”, in *Automatica*, vol. 50, no. 4, pp. 1279-1287, 2014.
- [10] D. Luenberger, “An introduction to observers”, in *IEEE Transactions on Automatic Control*, vol. 16, no. 6, pp. 596-602, December 1971.
- [11] J. Almeida, C. Silvestre, A.M. Pascoal, “Observer Based Self-Triggered Control of Linear Plants with Unknown Disturbances”, in *American Control Conference*, pp. 5688-5693, 2012.
- [12] X. Wang and M.D. Lemmon, “Self-Triggering Under State-Independent Disturbances”, in *IEEE Transactions on Automatic Control*, vol. 55, no. 6, pp. 1494-1500, 2010.
- [13] G. Franklin, J. Powell, and A. Emami-Naeini, *Feedback Control of Dynamical Systems*, Prentice Hall, 2010.
- [14] S. Gupta, “Increasing the sampling efficiency for a control system”, in *IEEE Transactions on Automatic Control*, vol. 8, no. 3, pp. 263-264, 1963.
- [15] K.J. Åström and B.M. Bernhardsson, “Comparison of periodic and event based sampling for first order stochastic systems”, in *Proceedings IFAC World Conference*, pp. 301-306, 1999.
- [16] K. Arzén, “A Simple Event-Based PID Controller”, in *Preprints IFAC World Conference*, vol. 8, pp. 423-428, 1999.
- [17] J.K. Yook, D.M. Tilbury and N.R. Soparkar, “Trading Computation for Bandwidth: Reducing Communication in Distributed Control Systems Using State Estimators”, in *IEEE Transactions on Control Systems Technology*, vol. 10, no. 4, pp. 503-518, 2012.

- [18] P. Tabuada, “Event-Triggered Real-Time Scheduling of Stabilizing Control Tasks in *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680-1685, 2007.
- [19] W.P.M.H. Heemels, J.H. Sandee and P.P.J. van den Bosch, “Analysis of event-driven controllers for linear systems”, in *International Journal of Control*, vol. 81, pp. 571-590.
- [20] J. Ackermann, “On the synthesis of linear control systems with specified characteristics”, in *Automatica*, vol. 13, pp. 89-94, 1977.
- [21] E. Weisstein, “Least Squares Fitting-Polynomial”, from *MathWorld—A Wolfram Web Resource*, <http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html>
- [22] C. Lozoya, P. Martí, M. Velasco, J. Fuertes and E. Martín, “Resource and performance trade-offs in real-time embedded control systems”, in *Real-Time Systems*, vol. 49, no. 3, pp. 267-307, 2013.
- [23] C. Rosero, C. Vaca, L. Tobar and F. Rosero, “Performance of Self-Triggered Control Approaches”, in *Enfoque UTE*, vol. 8, no. 2, pp. 107-120, 2017.

# Apéndice

Este apéndice incluye el software y el firmware desarrollados en el proyecto. Sólo se han anexado los archivos más importantes. Para obtener el código completo, consulte los medios adjuntos.

## **.A. Software**

### **.A.1. Diseño del controlador (controllerDesign.m)**

---

#### Programa 1: Diseño del controlador

---

```
%%TWO-THIRDS CONTROLLER
clear all, close all, clc,

A = [0 -23.8095; 0 0]; %dynamic system definition
B = [0; -23.8095];
C = [1 0];
D = 0;
sys = ss(A, B, C, D);

Q = 1/(23.8095^2)*eye(size(A,1));
R = 2/23.8095;
%S = 1e-4*[3, -5; -5, 20];

%%Continuous time controller-gain

L = lqr(A,B,Q,R); %continuous time control gain
eigenvalues = eig(A-B*L) %stability test

%%Settings
x = [0; 0]; %initial state
x_hat = [0; 0]; % initial observed states
tau_max = 38e-3;
tau_min = 5e-3;
granularity = 1e-3;
beta = 9.4;
alpha = 0.667;
eta = 0.0277;
Ns = 400; % Number of samples for simulation
```

```

refer = 0.5; %reference for simulation
snr = 50; %signal to noise ratio

%pole1 = -Inf; %poles in continuous time for the observer's poles placement
%pole2 = -Inf;
% pole1 = -3+2i;
% pole2 = -3-2i;
% pole1 = -5+2i;
% pole2 = -5-2i;
pole1 = -10+5i;
pole2 = -10-5i;

%% guarantee that all sampling periods will be in the range [tau_min, tau_max]

(beta^alpha)/eta
1/tau_min - 1/tau_max

%% L(tau_k) for all possible tau
L_hat_tk = [];

for tau_k = tau_min : granularity : tau_max
    L_hat_tk = [L_hat_tk; lqrd(A,B,Q,R,tau_k)]; %gain matrices
end

%% Calculating functions to fit the controller-gains behaviour
tau_k = [tau_min: granularity: tau_max]';
figure;
subplot(1, 2, 1);
%plot(tau_k, L_hat_tk(:,1), tau_k, L_hat_tk(:,2));

% new polynomials to fit elements of gain matrix along tau_k
polyGain1 = polyfit(tau_k, L_hat_tk(:,1), 1) %1st element
polyGain2 = polyfit(tau_k, L_hat_tk(:,2), 1) %2nd element

L_hat_tk_1 = polyval(polyGain1, tau_k);
L_hat_tk_2 = polyval(polyGain2, tau_k);

hold on
plot(tau_k, L_hat_tk_1, 'o', tau_k, L_hat_tk_2, '*');
plot(tau_k, L_hat_tk_1, 'Color','g');
plot(tau_k, L_hat_tk_2, 'Color','g');
title('L_{\tau k}');
hold off

```

```

%%Implementation into the microcontroller (C language)

syms A00 A01 A10 A11
syms B00 B10
syms L00 L01
syms x00 x10
syms r

a = [A00 A01; A10 A11];
b = [B00; B10];
l = [L00 L01];
X = [(x00-r); x10];

l*(a+b*l)*X

%%Observer design

Adisc = [];
Bdisc = [];

L_obs_tau_k = [];
i = 0;

for tau_k = tau_min : granularity : tau_max
    i = i+1;
    z1 = exp(pole1*tau_k); %poles in discrete time
    z2 = exp(pole2*tau_k);
    sys_d = c2d(sys, tau_k);
    Adisc = [Adisc; sys_d.a];
    Bdisc = [Bdisc, sys_d.b];
    L_obs_tau_k = [L_obs_tau_k, acker(sys_d.a', sys_d.c', [z1 z2])']; %gain matrices
    if tau_k == tau_min || tau_k == tau_max %check stability for minimal and maximal gains
        eig(A-L_obs_tau_k(:,i)*C)
    end
end

end

%% Calculating functions to fit the observer-gains behaviour
tau_k = [tau_min: granularity: tau_max]';
subplot(1,2,2);

% new polynomials to fit elements of gain matrix along tau_k
polyGainObs1 = polyfit(tau_k, L_obs_tau_k(1,:)', 1) %1st element

```



```

polyGainObs2 = polyfit(tau_k, L_obs_tau_k(2,:), 1) %2nd element

L_obs_tk_1 = polyval(polyGainObs1, tau_k);
L_obs_tk_2 = polyval(polyGainObs2, tau_k);

hold on
plot(tau_k, L_obs_tk_1, 'o', tau_k, L_obs_tk_2, '*');
plot(tau_k, L_obs_tk_1, 'Color', 'g');
plot(tau_k, L_obs_tk_2, 'Color', 'g');
title('Lobs_{\tau k}');
hold off

%%Observer implementation into the microcontroller (C language)
syms Ad00 Ad01 Ad10 Ad11
syms Bd00 Bd10
syms L00 L01
syms Lobs_k00 Lobs_k10
syms x00 x10
syms x_hat00 x_hat10
syms u r

a = [Ad00 Ad01; Ad10 Ad11];
b = [Bd00; Bd10];
l = [L00 L01];
X = [x00; x10];
l_obs = [Lobs_k00; Lobs_k10];
x_hat_ = [x_hat00; x_hat10];

u_ = -l*[x_hat_(1)-r; x_hat_(2)]
y_ = C*X + D*u;
y_hat_ = C*x_hat_ + D*u;

x_hat_new = a*x_hat_ + l_obs*(y_ - y_hat_) + b*u
%x_hat_new = (a-b*l)*x_hat_ + l_obs*(X - x_hat_)
%%

simulation

```

---

## .A.2. Interfaz para obtener datos desde microcontrolador (MatlabInterface.m)

### Programa 2: Interfaz para obtener datos desde microcontrolador

---

```
%m-file to acquire ERIKA-FLEX FULL data from the serial port using RS232
%asar esto para cambiar el nombre del puerto usb: sudo ln -s /dev/ttyACM0 /dev/ttyUSB0

clear all
close all

number_of_samples = 500;           %set the number of acquisitions
delete(instrfind)                 %clean all open ports

%configuring serial port
s = serial('/dev/ttyUSB0');        %creates a matlab object from the serial port
s.BaudRate = 115200;              %baudrate=115200bps
s.Parity = 'none';               %no parity
s.DataBits = 8;                  %data sended in 8bits format
s.StopBits = 1;                  %1 bit to stop
s.FlowControl = 'none';          %no flowcontrol
s.Terminator = 'LF';             %LineFeed character as terminator
s.Timeout = 1;                   %maximum time in seconds since the data is readed
s.InputBufferSize = 100000;      %increment this value when data is corrupted

q = quantizer('float',[32 8]);    %cast 32bits hex to float

%%Experiment data
try
    disp('Triggering experiment!')
    fopen(s)                       %open serial port object
    error = 0;
catch
    fclose(s)
    %close all
    disp('Error, port could not be opened!')
    error = 1;
end

if error == 0
```

```

data = zeros(23,1);
disp('Starting acquisition of data from experiment... ')
fwrite(s, '1', 'char');
%figure;
for n = 1:number_of_samples %get data
    data = fread(s,23,'char');

    t(n,1) = hex2dec([dec2hex(data(2),2) dec2hex(data(3),2) dec2hex(data(4),2) dec2hex(
        data(5),2)])*25e-9;%25e-9 is 1/Fcy=1/40e6
    r(n,1) = hex2num(q,[dec2hex(data(9),2) dec2hex(data(8),2) dec2hex(data(7),2) dec2hex(
        data(6),2)]);%joint four bytes, float value
    x1(n,1) = hex2num(q,[dec2hex(data(13),2) dec2hex(data(12),2) dec2hex(data(11),2)
        dec2hex(data(10),2)]);
    x2(n,1) = hex2num(q,[dec2hex(data(17),2) dec2hex(data(16),2) dec2hex(data(15),2)
        dec2hex(data(14),2)]);
    u(n,1) = hex2num(q,[dec2hex(data(21),2) dec2hex(data(20),2) dec2hex(data(19),2)
        dec2hex(data(18),2)]);
    %a(n,1) = bitshift(data(23),24) + bitshift(data(22),16) + bitshift(data(21),8) + data
        (20); %joint two bytes, unsigned int value
end

fwrite(s, '0', 'char');
fclose(s)

%%Sampling data
try
    fopen(s) %open serial port object
    error = 0;
catch
    fclose(s)
    %close all
    disp('Error, port could not be opened!')
    error = 1;
end

if error == 0
    data = zeros(23,1);
    disp('Starting acquisition of data concerning sampling... ')
    %figure;
    fwrite(s, '2', 'char');
    warning('off','all'); %suppress all warnings
    try
        for n = 1:number_of_samples %get data

```

```

        data = fread(s,23,'char');
        for i = 2 : 1 : 23
            a = n*22 - 22 + i - 1;
            tau(a,1) = hex2dec(dec2hex(data(i),1))/1000;
        end
    end
catch

end

fclose(s)

%%Plotting

tt = [];
for i=1:length(t)
    tt = [tt; t(i,1)-t(1,1)];
end

figure
subplot(2,1,1)

hold on
grid on
plot(tt,r,'g.',tt,x1,'b');
%axis([max(tt)-3 max(tt) -1.75 1.75]);
xlim([0 max(tt)]);
stairs(tt,u,'r') %Plots u as stair
legend('r','x_1','u')
xlabel('t (s)')
ylabel('Voltage (V)')
title('Acquisition')

subplot(2,1,2)
hold on
stem([0; cumsum(tau(1:end-1))],tau,'r')
tauValid = tau(1:number_of_samples-1)
plot([0; cumsum(tauValid(1:end))], mean(tauValid),'b*')
xlim([0 max(tt)]);
grid on
disp('Done!')

end

```

end

---

### .A.3. Simulación de OSISTC (simulation.m)

---

#### Programa 3: Simulación de OSISTC

---

```
%%Simulation ideal system

%%Initial values

history_tau_k = [];
history_x = [];
history_ref = [];
history_u = [];
history_time = 0;

r = 1;
reference = r*refer;
%%
for n=1:Ns % iterate through time

    % reference evolution
    if mod(n,(Ns/4)) == 0
        r = ~r;
        reference = r*refer;
    end
    history_ref = [history_ref , r];

    tau_k = tau_max/(1+(tau_max/eta)*(abs(L*(A+B*L)*[x(1)-r; x(2)]))^alpha); % tau(k)
    tau_k = round(tau_k*1000); % transform into milliseconds in order to round
    tau_k = tau_k/1000; % go back to seconds
    history_tau_k = [history_tau_k , tau_k];
    history_time = [history_time , history_time(end)+tau_k];

    % control signal
    L_hat_tk = lqrd(A,B,Q,R,tau_k); %L(tau(k))
    u = L_hat_tk*[r-x(1); -x(2)]; % control u(k)
    history_u = [history_u , u];
```

```

    %dynamics evolution
    [Ad, Bd] = c2d(A, B, tau_k); %new discrete plant considering tau(k)
    x = Ad*x + Bd*u; % x(k+1)
    history_x = [history_x , x];
end

%%Plotting ideal system

figure
subplot(3,2,1)
stem(history_time(1:Ns), history_tau_k)
xlim([0 history_time(Ns)])
title('Ideal system')
xlabel('t')
ylabel('{\tau}_k')

subplot(3,2,3)
grid on, hold on
plot(history_time(1:Ns), history_x(1,:), history_time(1:Ns), ...
      history_x(2,:), history_time(1:Ns), history_ref(1,:), 'lineWidth', 2);
xlim([0 history_time(Ns)])%, ylim([-1.7 1.7])
title('Response')
xlabel('t')
ylabel('x_1, x_2')
hold off

subplot(3,2,5)
hold on
plot(history_time(1:Ns), history_tau_k)
plot(history_time(1:Ns), mean(history_tau_k), '-')
hold off
xlim([0 history_time(Ns)])

subplot(3,2,[2,4,6])
plot(history_x(1,:), history_x(2,:));

%%Simulation of noisy system without observer

figure
history_tau_k = [];
history_x = [];
history_ref = [];

```

```

history_u = [];
history_time = 0;

for n=1:Ns % iterate through time
    % reference evolution
    % reference evolution
    if mod(n,(Ns/4)) == 0
        r = ~r;
        reference = r*refer;
    end
    history_ref = [history_ref , r];

    %tau_k = tau_max/(1+(tau_max/eta)*(abs(L*(A+B*L)*[x(1)-r; x(2)]))^alpha); % tau(k)

    tau_k = tau_max/(1+(tau_max/eta)*(abs(L*(A+B*L)*[r-x(1); -x(2)]))^alpha); % tau(k)

    tau_k = round(tau_k*1000); % transform into milliseconds in order to round
    tau_k = tau_k/1000; % go back to seconds
    history_tau_k = [history_tau_k , tau_k];
    history_time = [history_time , history_time(end)+tau_k];

    % control gain and control signal
    L_hat_tk = lqrd(A, B, Q, R, tau_k); % L(tau(k))
    u = -L_hat_tk * [x(1)-r; x(2)]; % control u(k)
    history_u = [history_u , u];

    %dynamics evolution
    sys_d = c2d(sys , tau_k);
    x = sys_d.a*x + sys_d.b*u; % x(k+1)
    x = awgn(x, snr); % add noise
    history_x = [history_x , x];
end

%%Plotting noisy system without observer

subplot(3,2,1)
grid on
% plot(history_time(1:Ns), history_x(1,:), history_time(1:Ns) ,...
%     history_x(2,:), history_time(1:Ns), history_ref(1,:), 'lineWidth', 2);

plot(history_time(1:Ns), history_x(1,:), history_time(1:Ns), history_u(1,:), ...
     history_time(1:Ns), history_ref(1,:), 'lineWidth', 2);

```

```

        xlim([0 history_time(Ns)])%, ylim([-1.7 1.7])
    title('Response')
    xlabel('t')
    ylabel('x_1, x_2')

    subplot(3,2,3)
    stem(history_time(1:Ns), history_tau_k)
    xlim([0 history_time(Ns)])
    title('Noisy without observer')
    xlabel('t')
    ylabel('{\tau}_k')

    subplot(3,2,5)
    hold on
    plot(history_time(1:Ns), history_tau_k)
    plot(history_time(1:Ns), mean(history_tau_k), '-')
    hold off
    xlim([0 history_time(Ns)])

%%Simulation of noisy system with observer

history_tau_k = [];
history_x = [];
history_x_hat = [];
history_ref = [];
history_u = [];
history_time = 0;

for n=1:Ns % iterate through time

    % reference evolution
    % reference evolution
    if mod(n,(Ns/4)) == 0
        r = ~r;
        reference = r*refer;
    end
    history_ref = [history_ref, r];

    tau_k = tau_max/(1+(tau_max/eta)*(abs(L*(A+B*L)*[r-x_hat(1); -x_hat(2)]))^alpha); % tau(k)
    tau_k = round(tau_k*1000); % transform into milliseconds in order to round
    tau_k = tau_k/1000; % go back to seconds
    history_tau_k = [history_tau_k, tau_k];
    history_time = [history_time, history_time(end)+tau_k];

```



```

% control gain and control signal
L_hat_tk = lqrd(A,B,Q,R,tau_k); %L(tau(k))
u = L_hat_tk*[r-x_hat(1); -x_hat(2)]; % control u(k)

history_u = [history_u , u];

% observer gain
sys_d = c2d(sys , tau_k);
%sys_d = c2d(sys , tau_max);
z1 = exp(pole1*tau_k); %poles in discrete time
z2 = exp(pole2*tau_k);
L_obs_tau_k = acker(sys_d.a', sys_d.c' , [z1 z2]); %gain matrices

y = C*x; %it is the same C or Cd
y_hat = C*x_hat;
x_hat = sys_d.a*x_hat + L_obs_tau_k*(y - y_hat) + sys_d.b*u;
history_x_hat = [history_x_hat, x_hat];

%dynamics evolution
sys_d = c2d(sys, tau_k);
x = sys_d.a*x + sys_d.b*u; % x(k+1)
x = awgn(x, snr); %add noise
history_x = [history_x, x];
end

%% Plotting noisy systems

subplot(3,2,2)
grid on
% plot(history_time(1:Ns),history_x(1,:),history_time(1:Ns),history_x(2,:),...
%     history_time(1:Ns),history_x_hat(1,:),history_time(1:Ns),history_x_hat(2,:),...
%     history_time(1:Ns),history_ref(1,:),'lineWidth',2);

plot(history_time(1:Ns),history_x(1,:),history_time(1:Ns),history_x_hat(1,:),...
     history_time(1:Ns),history_u(1,:),history_time(1:Ns),history_ref(1,:),...
     'lineWidth',2);

xlim([0 history_time(Ns)])%, ylim([-1.7 1.7])
title('Response')
xlabel('t')
ylabel('x_1, x_2')

```

```

subplot(3,2,4)
stem(history_time(1:Ns), history_tau_k)
xlim([0 history_time(Ns)])
title('Noisy with observer')

xlabel('t')
ylabel('\tau_k')

subplot(3,2,6)
hold on
plot(history_time(1:Ns), history_tau_k)
plot(history_time(1:Ns), mean(history_tau_k), '-')
hold off
xlim([0 history_time(Ns)])

```

---

## .B. Firmware

### .B.1. Código implementado sobre microcontrolador (code.c)

---

#### Programa 4: Código implementado sobre microcontrolador

---

```

/* CONTROLLER BOARD */

#include "ee.h"
#include "cpu/pic30/inc/ee_irqstub.h"
#include "setup.h"
#include "uart_dma.h"
#include "math.h"

_FOSCSEL(FNOSC_PRIPLL); // Primary (XT, HS, EC) Oscillator with PLL
_FOSC(OSCIOFNC_ON & POSCMD_XT); // OSC2 Pin Function: OSC2 is Clock Output
// Primary Oscillator Mode: XT
// Crystal

_FWDT(FWDTEN_OFF); // Watchdog Timer Enabled/disabled by user software
_FGS(GCP_OFF); // Disable Code Protection

```

```

/*      Variables      */

static float r=-1;

static float v_max=3.3; //dsPIC voltage reference
static float offset=0.0;//0.06;//OPAMP offset

////////////////////////////////////
//Two thirds
static float A[2][2] = {{0,-23.8095},{0,0}};
static float B[2][1] = {{0},{-23.8095}};
static float L[1][2] = {{0.1449, -0.5575}};
static float x[2][1] = {{0},{0}};
static float alpha = 0.66;
static float eta = 0.0277;
static float tau_max = 38e-3;
static float tau_min = 15e-3;
static float tau_k = 15e-3;
static float u = 0;
static float gain1[2] = {-0.7995, 0.1430};
static float gain2[2] = {1.7644, -0.5550};
static float L_k[1][2] = {{0, 0}};

//Observer
static float Ad[2][2] = {{1,-0.3571},{0,1}};
static float Bd[2][1] = {{0.0638},{-0.3571}};

static float gainObs1[2] = {0, 2};
static float gainObs2[2] = {68.4053, -3.5213};

static float x_hat[2][1] = {{0},{0}};
static float Lobs_k[2][1] = {{0},{0}};

#define lengthBuffer 450
static unsigned char bufferTau1[lengthBuffer];
static unsigned char bufferTau2[lengthBuffer];

#define buffer1 1
#define buffer2 2
#define experiment 1
#define tauAcquisition 2
#define none 0
static unsigned char whichBuffer = buffer1;

```

```

unsigned char processState = none; //by default , system turned-off
static unsigned int countBufferTau = 0;
static unsigned int countReadBuffer = 0;

static unsigned long sys_time=0;

/* Change the reference value between -0.5 and 0.5V*/
TASK(TaskReferenceChange)
{
    if (r == -1)
    {
        r = 1;
        //LATBbits.LATB14 = 1;//Orange led switch on
    }
    else
    {
        r = -1;
        //LATBbits.LATB14 = 0;//Orange led switch off
    }
}

/* Read RCRC circuit output voltages */
void Read.State(void)
{
    AD1CHS0 = 22;           // Input ADC channel selection
    AD1CON1bits.SAMP = 1; // Start conversion
    while (!IFS0bits.AD1IF); // Wait till the EOC
    IFS0bits.AD1IF = 0;    // Reset ADC interrupt flag
    x[0][0] = (ADC1BUF0*v_max/4096)-v_max/2-offset; // Acquire data and scale

    AD1CHS0 = 23;           // Input ADC channel selection
    AD1CON1bits.SAMP = 1; // Start conversion
    while (!IFS0bits.AD1IF); // Wait till the EOC
    IFS0bits.AD1IF = 0;    // Reset ADC interrupt flag
    x[1][0] = (ADC1BUF0*v_max/4096)-v_max/2-offset; // Acquire data and scale
}

void Actuate(void)
{
    PDC1 = (u/v_max)*0x7fff+0x3FFF;
}

/* Controller Task */

```

```

TASK(TaskController)
{
    float accum1 = 0, accum2 = 0;
    static unsigned char *p_r = (unsigned char *)&r;

    //////////////////////////////////
    //static unsigned char *p_r = (unsigned char *)&L_k[0][0];
    //static unsigned char *p_x0 = (unsigned char *)&L_k[0][1];
    //////////////////////////////////
    //static unsigned char *p_x0 = (unsigned char *)&x_hat[0][0];
    static unsigned char *p_x0 = (unsigned char *)&x[0][0];
    //static unsigned char *p_x1 = (unsigned char *)&x_hat[1][0];
    static unsigned char *p_x1 = (unsigned char *)&x[1][0];
    static unsigned char *p_u = (unsigned char *)&u;

    LATBbits.LATB14 = 1; //Turn on orange led

    sys_time=GetTime();

    Read_State();

    //L*(A+B*L)*x =
    // =(x00-r)*(L00*(A00 + B00*L00) + L01*(A10 + B10*L00)) +
    // x10*(L00*(A01 + B00*L01) + L01*(A11 + B10*L01))

    accum1 = L[0][0]*(A[0][0] + B[0][0]*L[0][0]);
    accum1 += L[0][1]*(A[1][0] + B[1][0]*L[0][0]);
    //standard controller
    accum1 *= (x[0][0]-r);
    //controller with observer
    //accum1 *= (x_hat[0][0]-r);

    accum2 = L[0][0]*(A[0][1] + B[0][0]*L[0][1]);
    accum2 += L[0][1]*(A[1][1] + B[1][0]*L[0][1]);
    //standard controller
    accum2 *= x[1][0];
    //controller with observer
    //accum2 *= x_hat[1][0];

    accum1 += accum2;

```

```

//abs(L*(A+B*L)*x)
if (accum1<0)
    accum1 = -accum1;

//((abs(L*(A+B*L)*x))^alpha
accum2 = pow(accum1, alpha);

//[(tau_max/eta)*(abs(L*(A+B*L)*x))^alpha]+1
accum1 = (tau_max/eta)*accum2 + 1;

//tau_k
tau_k = tau_max/accum1; //value in milliseconds

if (tau_k > tau_max)
    tau_k = tau_max;
else if (tau_k < tau_min)
    tau_k = tau_min;

//fix the value to be used to trigger the next process
accum1 = tau_k*1000;
//////////
//AQUI APLICAR ROUND
//////////
SetRelAlarm(AlarmController ,(unsigned int)(accum1),0);

if (whichBuffer == buffer1)
{
    bufferTau1[countBufferTau] = (unsigned char)(accum1);
    countBufferTau++;
    if (countBufferTau > lengthBuffer)
    {
        countBufferTau = 0;
        whichBuffer = buffer2;
    }
}
else if (whichBuffer == buffer2)
{
    bufferTau2[countBufferTau] = (unsigned char)(accum1);
    countBufferTau++;
    if (countBufferTau > lengthBuffer)
    {
        countBufferTau = 0;
        whichBuffer = none; //buffers out, more information will not be saved
    }
}

```

```

        }
    }
    else if (whichBuffer == none)
    {

    }

    //L_k is calculated with two polynomial taken from polyfits made
    //with the set of L_k computed off-line using matlab
    L_k[0][0] = gain1[0]*tau_k + gain1[1];
    L_k[0][1] = gain2[0]*tau_k + gain2[1];

    //standard controller
    u = -L_k[0][0]*(x[0][0] - r) - L_k[0][1]*x[1][0];

    //controller with observer
    //u = -L_k[0][0]*(x_hat[0][0] - r) - L_k[0][1]*x_hat[1][0];

    //Check for saturation
    if (u > v_max/2) u = v_max/2;
    if (u < -v_max/2) u = -v_max/2;

    //Lobs_k is calculated with two polynomial taken from polyfits made
    //with the set of Lobs_k computed off-line using matlab
    Lobs_k[0][0] = gainObs1[0]*tau_k + gainObs1[1];
    Lobs_k[1][0] = gainObs2[0]*tau_k + gainObs2[1];

    //Discretize A and B using the current sampling and the approximations shown
    //in Computer Controlled Systems by Astrom

    Ad[0][1] = -23.8095*tau_k;
    Bd[0][0] = 283.4461*tau_k*tau_k;
    Bd[1][0] = Ad[0][1];

    //Using accumulators because both calculations must be done at the same time
    accum1 = Ad[0][0]*x_hat[0][0] + Ad[0][1]*x_hat[1][0];
    accum1 += Lobs_k[0][0]*(x[0][0] - x_hat[0][0]) + Bd[0][0]*u;

    accum2 = Ad[1][0]*x_hat[0][0] + Ad[1][1]*x_hat[1][0];
    accum2 += Lobs_k[1][0]*(x[0][0] - x_hat[0][0]) + Bd[1][0]*u;

    x_hat[0][0] = accum1;
    x_hat[1][0] = accum2;

```

```

Actuate();

//LATBbits.LATB10 = 1; //To get time with the oscilloscope

//sys_time=GetTime(); //Get system time (EDF Scheduler)

OutBuffer[0]=0x01; //header;
OutBuffer[1]=(unsigned char)(sys_time>>24); //4th byte of unsigned long
OutBuffer[2]=(unsigned char)(sys_time>>16); //3rd byte of unsigned long
OutBuffer[3]=(unsigned char)(sys_time>>8); //2nd byte of unsigned long
OutBuffer[4]=(unsigned char)sys_time; //1st byte of unsigned long
OutBuffer[5]=*p_r; //4th byte of float (32 bits)
OutBuffer[6]=*(p_r+1); //3rd byte of float (32 bits)
OutBuffer[7]=*(p_r+2); //2nd byte of float (32 bits)
OutBuffer[8]=*(p_r+3); //1st byte of float (32 bits)
OutBuffer[9]=*p_x0;
OutBuffer[10]=*(p_x0+1);
OutBuffer[11]=*(p_x0+2);
OutBuffer[12]=*(p_x0+3);
OutBuffer[13]=*p_x1;
OutBuffer[14]=*(p_x1+1);
OutBuffer[15]=*(p_x1+2);
OutBuffer[16]=*(p_x1+3);
OutBuffer[17]=*p_u;
OutBuffer[18]=*(p_u+1);
OutBuffer[19]=*(p_u+2);
OutBuffer[20]=*(p_u+3);

//Force sending data
DMA4CONbits.CHEN = 1; // Re-enable DMA4 Channel
DMA4REQbits.FORCE = 1; // Manual mode: Kick-start the first transfer

//LATBbits.LATB10 = 0; //To get time with the oscilloscope

LATBbits.LATB14 = 0; //Turn off orange led

}

/* Send data using the UART port 1 via RS-232 to the PC
 * Note: This Task takes less than 0.0002 seconds */

TASK(TaskSupervision)
{

```



```

static unsigned int i;
static unsigned char noThisBuffer = 0;

if (processState == tauAcquisition)
{
    if (whichBuffer == buffer1)
    {
        OutBuffer[0] = 0xff; //header

        noThisBuffer = 0;

        for (i=1; i<=22; i++)
        {
            if (noThisBuffer == 0)
            {
                OutBuffer[i] = bufferTau1[countReadBuffer];
                countReadBuffer++;
                if (countReadBuffer >= lengthBuffer)
                {
                    countReadBuffer = 0;
                    noThisBuffer = 1;
                    whichBuffer = buffer2;
                }
            }
            else
            {
                OutBuffer[i] = bufferTau2[countReadBuffer];
                countReadBuffer++;
            }
        }
        //Force sending data
        DMA4CONbits.CHEN = 1; //Re-enable DMA4 Channel
        DMA4REQbits.FORCE = 1; //Manual mode: Kick-start the first transfer

        //LATBbits.LATB10 = 0; //To get time with the oscilloscope
    }
    else if (whichBuffer == buffer2)
    {
        OutBuffer[0] = 0xff; //header

```

```

noThisBuffer = 0;

for (i=1; i<=22; i++)
{
    if (noThisBuffer == 0)
    {
        OutBuffer[i] = bufferTau2[countReadBuffer];
        countReadBuffer++;
        if (countReadBuffer >= lengthBuffer)
        {
            countReadBuffer = 0;
            noThisBuffer = 1;
            whichBuffer = none;
        }
    }
    else
    {
        OutBuffer[i] = 0;
    }
}
//Force sending data
DMA4CONbits.CHEN = 1; //Re-enable DMA4 Channel
DMA4REQbits.FORCE = 1; //Manual mode: Kick-start the first transfer
//LATBbits.LATB10 = 0; //To get time with the oscilloscope
}
else //auto-stop supervision
{
    processState = none;
    countBufferTau = 0;
    countReadBuffer = 0;
    CancelAlarm(AlarmSupervision);

    for(i=0; i<=lengthBuffer; i++)
    {
        bufferTau1[i] = 0;
        bufferTau2[i] = 0;
    }
}
}

ISR2(_DMA5Interrupt) //UART RX interruption
{

```

```

//LATBbits.LATB14 ^= 1; //To get time with the oscilloscope

switch (InBufferA [0])
{
    case 0x30: //stop
        processState = none;
        //reset values
        r = -1;
        u = 0;
        x_hat [0][0] = 0;
        x_hat [1][0] = 0;
        x [0][0] = 0;
        x [1][0] = 0;
        CancelAlarm (AlarmReferenceChange);
        CancelAlarm (AlarmController);
        CancelAlarm (AlarmSupervision);

        break;

    case 0x31: //start running experiment
        processState = experiment;
        whichBuffer = buffer1;
        countBufferTau = 0;
        SetRelAlarm (AlarmReferenceChange ,50,1500); //ref changes every 1500ms
        SetRelAlarm (AlarmController ,50,0);
        break;

    case 0x32: //start reading buffer - it is auto-stopped
        processState = tauAcquisition;
        whichBuffer = buffer1;
        countReadBuffer = 0;
        SetRelAlarm (AlarmSupervision , 1, 5); //Data is sent to computer every 5
            ms
        break;

    default :
        processState = none;
        break;
}

IFS3bits.DMASIF = 0; // Clear the DMAI Interrupt Flag
}

```

```
int main(void)
{
    Sys_init();//Initialize clock , devices and peripherals

    //SetRelAlarm(AlarmReferenceChange,10,1000); //ref changes every 1000ms
    //SetRelAlarm(AlarmController ,(unsigned int)(tau_min*1000),0);
    //SetRelAlarm(AlarmSupervision , 1, 5);//Data is sent to computer every 5ms

    /* Forever loop: background activities (if any) should go here */
    for (;);

    return 0;
}
```

---