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:

"LAZOS DE CONTROL EN RED CON SINCRONIZACIÓN EN LOS INSTANTES DE ACTUACIÓN"

AUTOR: JORGE EDISON QUILUMBANGO VACA

DIRECTOR: CARLOS XAVIER ROSERO CHANDI

IBARRA-ECUADOR

FEBRERO 2019

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN

A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

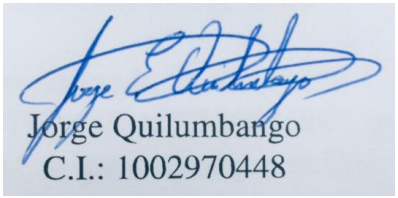
En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL AUTOR			
CÉDULA DE IDENTIDAD:	1002970448		
APELLIDOS Y NOMBRES:	QUILUMBANGO VACA JORGE EDISON		
DIRECCIÓN:	ATUNTAQUI-JULIO MIGUEL AGUINAGA		
EMAIL:	jequilumbangov@utn.edu.ec - edisn15@gmail.com		
TELÉFONO FIJO:	062908429	**TELÉFONO MÓVIL:**	0990476738
DATOS DE LA OBRA			
TÍTULO:	"LAZOS DE CONTROL EN RED CON SINCRONIZACIÓN EN LOS INSTANTES DE ACTUACIÓN"		
AUTOR:	JORGE EDISON QUILUMBANGO VACA		
FECHA:	15-03-2019		
SÓLO PARA TRABAJOS DE GRADO			
PROGRAMA:	PREGRADO		
TÍTULO POR EL QUE OPTA:	INGENIERO EN MECATRÓNICA		
ASESOR/DIRECTOR:	CARLOS XAVIER ROSERO C.		

2. 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 15 días del mes de marzo de 2019.



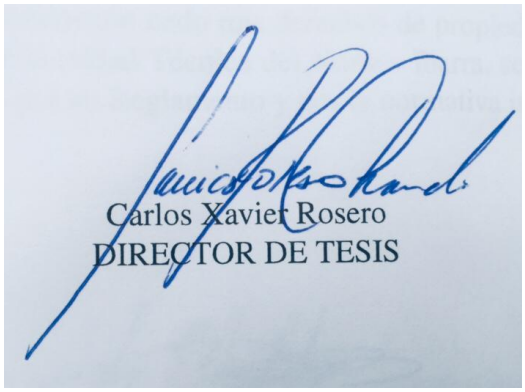
UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado “LAZOS DE CONTROL EN RED CON SINCRONIZACIÓN EN LOS INSTANTES DE ACTUACIÓN”, presentado por el egresado JORGE EDISON QUILUMBANGO VACA, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, febrero de 2019



UNIVERSIDAD TÉCNICA DEL NORTE

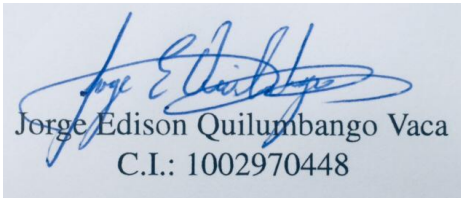
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

DECLARACIÓN

Yo, Jorge Edison Quilumbango Vaca con cédula de identidad Nro. 1002970448, 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, febrero de 2019



Agradecimiento

A mis padres Enrique Quilumbango y María Vaca, quienes me han apoyado en cada paso de mi formación como profesional y como persona.

A mis hermanos Jeaneth, Johana y Alex y a Jhadira por brindarme su apoyo en los momentos de necesidad.

A mi director Carlos Xavier Rosero, quien con sus conocimientos, hizo posible culminar el presente trabajo.

A la Universidad Técnica del Norte, por abrirme sus puertas y poder cumplir una meta más en mi vida académica y profesional.

Al personal docente de la Facultad de Ingeniería en Ciencias Aplicadas, quienes con paciencia y empeño han impartido conocimientos fundamentales para la culminación de este trabajo y mi formación moral y ética.

Jorge Quilumbango

Dedicatoria

Dedico presente trabajo a mis queridos padres quienes con amor, paciencia y esfuerzo lucharon diariamente olvidándose muchas veces de sí mismos para educar a sus hijos. A mis hermanos y familiares quienes me han apoyado para obtener el título de Ingeniero en Mecatrónica.

Jorge Quilumbango

Resumen

Los sistemas conocidos como lazos de control en red (NCS, Networked Control Systems), emplean una red de comunicación que permite a los nodos sensor, controlador y actuador compartir información a través de la misma. Dichos sistemas se caracterizan por trabajar con dos puntos de sincronización (muestreo y actuación) para realizar una acción de control en una planta. Además, debido al uso de una red de comunicación, se insertan retardos de tiempo, que producen problemas de sincronización como retardos de control, errores transitorios y jitter. La mayoría de los análisis realizados para hacer frente a estos problemas requieren que los retardos de control sean constantes y que los puntos de sincronización esten bien definidos en el tiempo, lo cual es difícilmente conseguido debido a la latencia de la red y a factores externos que afectan al desempeño del sistema. Dentro de este marco, se ha desarrollado un nuevo modelo matemático el cual soluciona los problemas de sincronización y adicionalmente basa su funcionamiento en un solo punto de sincronización, en los instantes de actuación. El presente trabajo aborda algunas pautas de implementación para hacer esta nueva técnica aplicable sobre microcontroladores reales; donde los resultados de simulación y experimentos prácticos confirman que la técnica de sincronización en los instantes de actuación es efectiva.

Abstract

The systems known as networked control systems (NCS), use a communication network that allows the sensor, controller and actuator nodes to share information through it. These systems are characterized by working with two synchronization points (sampling and actuation) to perform a control action in a plant, in addition, due to the use of a communication network time delays are inserted, which produce synchronization problems such as control delays, transient errors and jitter. Most of the analyzes carried out to deal with these problems require both constant control delays and well defined in time synchronization points. This is difficult to achieve due to the latency of the network and external factors that affect the performance of the system. Within this framework, a new mathematical model has been developed which solves synchronization problems and also bases its operation on a single synchronization point at actuation instants. This paper addresses some implementation guidelines to make this new technique applicable to real microcontrollers; where the results of simulation and practical experiments confirm that the technique of synchronization at the actuation instants is effective.

Índice general

Índice de figuras

Lista de Programas

Capítulo 1

1.1. Motivación

Los avances en las tecnologías de la computación y comunicaciones han dado origen a un tipo de sistema de control, el cual emplea una red de comunicación que permite a los nodos tales como sensor, controlador y actuador intercambiar información por medio de la misma. Estos sistemas son comúnmente conocidos como NCS (Networked Control Systems)[1].

Al aplicarse en una NCS un modelo de control convencional, se requiere una invarianza de tiempo con un muestreo equidistante. La invarianza de tiempo requiere que los retardos de control sean constantes y que las acciones de control y muestreo sean bien definidos en el tiempo. Como los modelos de control convencional no tienen el mismo desempeño con una NCS, se genera entonces variaciones de tiempo. Para solucionar los problemas que generan las variaciones de tiempo, se las incluye en el diseño del controlador o deben de ser despreciables en comparación con la dinámica del proceso controlado. Estas variaciones de tiempo dan origen a problemas de sincronización tales como retrasos de control, errores transitorios, y jitter [2].

Este nuevo enfoque se centra en que todas las operaciones de control sean sincronizadas en los instantes de actuación, lo cual permite instantes de muestreo irregulares y ofrece robustez frente a los jitters y otros problemas de sincronización.

1.2. Objetivos

El objetivo principal de este proyecto consiste en desarrollar una técnica de control en red con sincronización en los instantes de actuación. Los siguientes objetivos especificos son también realizados:

- Abordar el estado del arte referente a los lazos de control basados en red.
- Desarrollar una estrategia que permita controlar plantas en los instantes de actuación.
- Simular la estrategia para comprobar su desempeño.
- Implementar el algoritmo de control para validarlo sobre un microcontrolador real.

1.3. Antecedentes

El modelo de espacio de estado para un sistema de control discreto, lineal e invariable en el tiempo con retardo de tiempo, implica un tiempo de sincronización en los instantes de muestreo y actuación suponiendo que el muestreo y la actuación ocurren al inicio y al final de cada operación de tarea.

Para una NCS la sincronización llega a ser el aspecto más crítico puesto que los retardos de tiempo variables evitan el correcto funcionamiento del sistema. Para hacer frente a estos problemas se han realizado investigaciones como en [3] y [4], donde se obtiene modelos de control matemáticos que son usados para el análisis y diseño de controladores con muestreo periódico.

Este algoritmo de sistemas de control en red sincronizado en los instantes de actuación ofrece un modelo de espacio de estado para tareas de control en tiempo real, con el fin de resolver los problemas que se generan en un sistema de control donde afecta los retardos de tiempo y los problemas que conllevan los mismos.

Sin embargo, la aplicación de esta solución propuesta basa su operación en mediciones de tiempo absoluto, y por lo tanto requiere sincronización de reloj precisa entre los nodos de la red, aprovechando el protocolo PTP [5] la sincronización de reloj está garantizada, este requisito se discute y se analiza para el caso específico de NCS basado en CAN (Control Area Network)[6].

1.4. Problema

Cuando los sensores y actuadores se comunican con un controlador remoto a través de una red multipropósito es necesario usar técnicas mejoradas para la estimación del estado, la determinación de la estabilidad del lazo cerrado y la síntesis del controlador. Los sistemas de control en red son sistemas en los que la comunicación entre sensores, actuadores y controladores se produce a través de una red de comunicación digital de banda limitada compartida[4].

Debido a que las NCS tienen una arquitectura flexible y los costos en instalación y mantenimiento son reducidos, sus aplicaciones se encuentran en una amplia gama de áreas tales como redes de sensores móviles [7], cirugía remota [8], y vehículos aéreos no tripulados (UAVs)[9], etc.

Los retardos de tiempo presentes en los controladores de lazo cerrado basados en NCS, generan problemas de sincronización como retrasos de control, jitter y errores transitorios, los cuales evitan su correcto funcionamiento, y, para eliminar estos efectos negativos, muchas investigaciones se centran en encontrar soluciones basados en el supuesto de que la operación de los lazos de control están sincronizados en los instantes de muestreo.

1.5. Justificación

Al suponer que un sistema tiene muestreo periódico, se asume una invariancia de tiempo, la cual requiere que los retardos de control sean constantes y que las acciones de control y muestreo tengan lugar en instantes bien definidos en el tiempo [4]. Al no cumplirse estas exigencias, es cuando se generan los problemas de sincronización.

La relevancia de este nuevo enfoque de sincronización está en eliminar los problemas antes mencionados implementando este nuevo algoritmo en los lazos de control basados en NCS.

1.6. Alcance

El presente proyecto se lo simulará en Matlab aplicándolo a una planta doble integrador para visualizar su funcionamiento y posteriormente se lo implementará en microcontrolador.

Capítulo 2

Revisión Literaria

Para hacer frente a los problemas generados por los retardos de tiempo variables, se han desarrollado varias técnicas como las que se presentan a continuación.

2.1. Enfoque integrado

Para comenzar con el análisis de [10], se implementa el lazo de control con una arquitectura distribuida, con tres nodos que se comunican a través de una red de comunicación $fieldbus$. En la Figura 2.1, el nodo sensor muestrea al sistema $y(t)$ con un período de muestreo h y envía los datos al nodo controlador introduciendo un retardo de tiempo de comunicación τ_{sc} (retardo de tiempo del sensor a controlador). El nodo controlador por su parte ejecuta el cálculo de control previamente diseñado, introduciendo un retardo de tiempo de cálculo τ_c, el cual se supondrá constante para cada ejecución del controlador. Del controlador se produce la salida $u(t)$, la cual se envía al nodo actuador, introduciendo otro retardo de tiempo de comunicación τ_{ca} (retardo de tiempo del controlador al actuador).

La acumulación de los retardos de tiempo inducidos por la arquitectura distribuida llamada

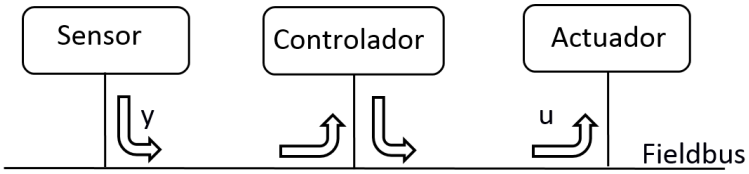


Figura 2.1: Arquitectura distribuida basada en Fieldbus.

retardo de tiempo de muestreo a actuación se describe por

$$\tau = \tau_{sc} + \tau_c + \tau_{ca}. \tag{2.1}$$

Para propósitos de control, se describe un modelo de espacio de estados para el sistema invariante de tiempo continuo descrito por las ecuaciones 2.2 y 2.3, incluyendo un delay constante τ,

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t-\tau) \tag{2.2}$$

$$y(t) = Cx(t) + Du(t), \tag{2.3}$$

donde se supone que τ es menor que el período de muestreo h, describiendo así el sistema de datos muestreado en un espacio de estados por 2.4 y 2.5.

$$x(kh+h) = \Phi(h)x(kh) + \Gamma_0(h,\tau)u(kh) + \Gamma_1(h,\tau)u(kh-h) \tag{2.4}$$

$$y(kh) = Cx(kh) + Du(kh) \tag{2.5}$$

donde

$$\Phi(h) = e^{Ah}$$
$$\Gamma_0(h,\tau) = \int_0^{h-\tau} e^{As} ds B$$
$$\Gamma_1(h,\tau) = e^{A(h-\tau)} \int_0^{\tau} e^{As} ds B.$$

El sistema especificado por 2.4 y 2.5 puede controlarse mediante retroalimentación de estado, expresados en 2.6, donde L, la ley de control de realimentación de estado se puede obtener mediante una estrategia de diseño de control, como la ubicación de polos o enfoque de optimización.

$$u(kh) = -L(h,\tau)x(kh) \tag{2.6}$$

Luego del proceso de diseño, el nodo controlador ejecutará un algoritmo de control que dependerá del período de muestreo h y del retardo de tiempo constante de muestreo a actuación τ. Aquí surgen dos problemas de implementación:

1. Al asumir τ constante, τ_{sc} y τ_{ca} variarán según las características de la red fieldbus, la política de control de acceso al medio y el tráfico de datos. Por lo tanto τ variará en cada ejecución del lazo de control. Representando un problema, debido a que 2.6 basada en 2.5 y 2.4 se ha diseñado suponiendo que τ es constante.

2. El algoritmo de control exige que el período de muestreo y los retardos de tiempo de muestreo-actuación se conozcan al comienzo de cada ejecución del controlador. El nodo sensor realiza un muestreo periódico con h y el retardo de tiempo de cálculo τ_c son conocidos y constantes y se los conoce al comienzo de cada ejecución del controlador. Los retardos de tiempo variables de muestreo a actuación inducidos por la red no pueden ser conocidos completamente, así τ_{sc} se puede conocer usando técnicas de marcas de tiempo si se asume un tiempo global para la sincronización de reloj en la arquitectura, pero τ_{ca} no se puede conocer al inicio de la ejecución del controlador.

La propuesta para hacer frente a estos problemas es el de modificar la arquitectura distribuida, ejecutando el controlador inmediatamente antes de que la salida se envíe a la planta, es decir eliminar el nodo controlador y mover el cálculo de control al nodo actuador, como se observa en la Figura 2.2.

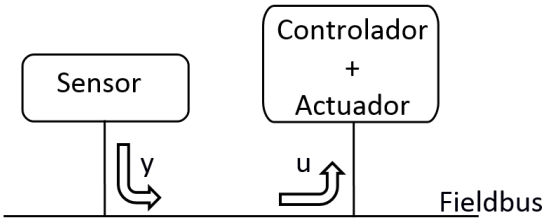


Figura 2.2: Nueva arquitectura distribuida.

Con esta nueva arquitectura, cada retardo de tiempo de actuación a muestreo puede conocerse al inicio de cada ejecución de calculo del controlador, representado por

$$\tau = \tau_{sa} + \tau_c \tag{2.7}$$

donde τ_c será un retardo de tiempo fijo pero τ_{sa} variara en cada ejecución del lazo de control debido a las características de comunicación de fieldbus y la dinámica. Así, la estrategia de diseño de control deberá tener en cuenta esta variación, sabiéndose que esta será limitada, es decir, τ_{sa} tomara valores dentro de un rango discreto conocido determinado en la etapa de diseño según las latencia de mensaje del peor de los casos de fieldbus. Por lo tanto, en cada ejecución k del cálculo de control, el retardo de tiempo de muestreo a actuación (τ_k) tomará valores dentro del rango generalizado descrito por

$$0 \leq \tau_k \leq \text{Latencia de mensaje del peor caso de Fieldbus.} \tag{2.8}$$

Sabiendo esto, el cálculo de control puede compensar el tiempo de ejecución para cada retardo de tiempo en cada ejecución del lazo de control si implementa el sistema de datos muestreados con un retardo de tiempo descrito en 2.4 y 2.5 formuladas como 2.9 y 2.10, donde

τ_k varía desde la ejecución en ejecución.

$$x(kh+h) = \Phi(h)x(kh) + \Gamma_0(h,\tau_k)u(kh) + \Gamma_1(h,\tau_k)u(kh-h) \tag{2.9}$$

$$y(kh) = Cx(kh) + Du(kh) \tag{2.10}$$

donde

$$\Phi(h) = e^{Ah}$$
$$\Gamma_0(h,\tau_k) = \int_0^{h-\tau_k} e^{As}ds\,B$$
$$\Gamma_1(h,\tau_k) = e^{A(h-\tau_k)}\int_0^{\tau_k} e^{As}ds\,B.$$

2.2. Sincronización de muestreo y actuación en ausencia de tiempo global

Los enfoques efectivos que se utilizan en una NCS a menudo requieren el imponer una ejecución periódica de operaciones de muestreo y/o actuación, que imponen retrasos de tiempo constantes sincronizados. Teniendo en cuenta que el muestreo y la actuación se realizan en diferentes nodos de la red, el tiempo global $GT(t)$ entre los nodos logrado por la sincronización del reloj es la estrategia estándar que permite tales operaciones sincronizadas. Esta técnica presentada en esta investigación, permite implementar las operaciones de muestreo y actuación en una NCS con la ausencia de un tiempo global. Esto se logra administrando tiempos locales entre los nodos de la red en lugar de forzar el mismo tiempo global en todos los nodos.

Definiendo la progresión de tiempo para cada nodo como una función del tiempo global que permita que $S(t)$, $C(t)$, $A(t)$, indiquen los tiempos locales en el sensor, controlador y actuador

respectivamente. Dejando a $GT(t)$ sea el tiempo global definido por

$$GT(t) = t,\ t \in \mathbb{R}^{+}. \tag{2.11}$$

Por lo tanto, los tiempos locales pueden ser descritos por

$$S(t) = k_s t + \phi_s,\ \ C(t) = k_c t + \phi_c,\ \ A(t) = k_a t + \phi_a, \tag{2.12}$$

donde ϕ_s, ϕ_c, y ϕ_a modelan la desincronización dada por los diferentes tiempos de inicio en los diversos nodos, y k_s, k_c, y k_a modelan los desvíos de relojes locales con respecto al tiempo global.

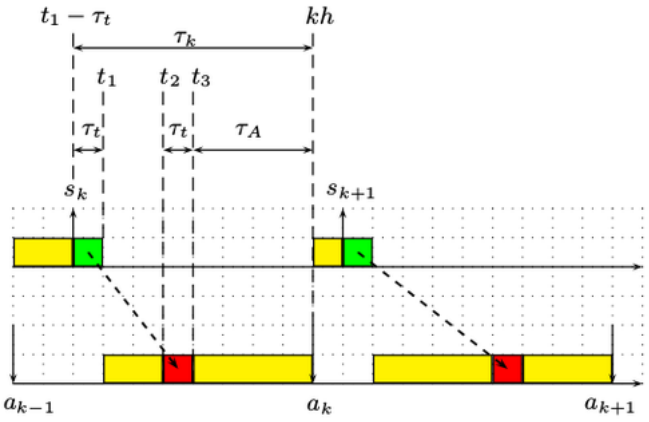


Figura 2.3: Técnica de sincronización

La técnica de sincronización esta basada en tres pasos:

1. En primer lugar, el intervalo de tiempo τ_k transcurrido desde el tiempo de muestreo "adivinado" hasta el tiempo de actuación periódica esperado kh se calcula en el nodo controlador. Aquí, se considera que en el nodo controlador el tiempo de muestreo $C(t_{s_k})$ se ha realizado en la recepción del mensaje entrante, $C(t_1)$, teniendo en cuenta su tiempo de

transmisión τ_t, es decir

$$C(t_{s_k}) = C(t_1) - \tau_t. \tag{2.13}$$

Lo cual es verdad si se impone que la planta sea muestreada justo antes de que el mensaje del sensor sea transmitido exitosamente. Esto explica por que las operaciones de muestreo s_k (flecha hacia arriba) en la Figura 2.3 aparecen justo antes de las transmisiones de mensajes del sensor (subcaja verde) y no cada cierto periodo h. Luego, el controlador, sabiendo que el actuador aplicará la señal de control periódicamente, en $C(kh)$, puede calcular el intervalo de tiempo τk que debe transcurrir entre el tiempo de muestreo y el tiempo de actuación como

$$\tau k = C(kh) - C(t_{s_k}) = C(kh) - (C(t_1) - \tau_t). \tag{2.14}$$

Este intervalo de tiempo puede variar cada operación de lazo cerrado.

2. Segundo, el controlador actualiza este intervalo de tiempo τ_k, obteniendo τ_A. Esto se debe a que ya ha sido parcialmente consumido y será consumido antes que el mensaje del controlador llegue al actuador. Básicamente, resta de τ_k el tiempo de ejecución del controlador $C(t_2) - C(t_1)$, y los tiempos de transmisión de mensaje del sensor y controlador, $2\tau_t$. Usando 2.14 se obtiene el intervalo de tiempo enviado al actuador τ_A, así

$$\tau_A = C(kh) - C(t_2) - \tau_t. \tag{2.15}$$

3. El nodo actuador esperara τ_A para aplicar la señal de control.

2.2.1. Pseudocódigo sin asumir un tiempo global

Cada nodo de un lazo de control en red que use esta técnica de sincronización, debe implementar en el sesor, controlador y actuador, los pseudocódigos presentados en el algoritmo 1

para sincronizar las operaciones sin asumir un tiempo global.

En el sensor, luego que se llevo a cabo una acción del actuador, se muestrea la planta y el mensaje del sensor se intenta enviarlo. Esto se repite hasta que se transmita exitosamente el mensaje.

Algoritmo 1.1: Pseudocódigo para el nodo sensor
begin
repeat
  $S_k := sample()$;
  send sensor message (S_k);
until $TX\ successfull$;
end

Algoritmo 1.2: Pseudocódigo para el nodo controlador
begin
$S_k := receive\ sensor\ message$;
$C(t_1) := get\ time()$;
$\tau_k := C(hk) - (C(t_1) - \tau_t)$;
$U_k := f2(S_k, \tau_k)$;
repeat
  $C(t_2) := get\ time()$;
  $\tau_A := C(hk) - (C(t_2) - \tau_t)$;
  send controller message (U_k, τ_A);
until $TX\ successfull$;
end

Algoritmo 1.3: Pseudocódigo para el nodo actuador
begin
$(U_k, \tau_A) := receive\ controller\ message$;
delay (τ_A);
apply control signal (U_k);
end

Algoritmo 1: Pseudocódigo sin asumir tiempo global

El controlador tras recibir el mensaje del sensor, obtiene el tiempo actual y calcula τ_k, y una acción de control. Particularmente, $f2(.)$ puede diseñarse usando one-shot task [11], el cual a sido desarrollado para sistemas de control con un muestreo aperiódico y una actuación pe-

riódica. Nótese que para calcular τ_A, se realiza permanentemente hasta que el mensaje se envíe exitosamente.

En el actuador, tras recibir el mensaje del controlador, se programa un timmer, el cual se disparará luego de τ_A unidades de tiempo y entonces se aplicara la señal de control a la planta [12].

2.3. Reducción del tráfico de datos sin sacrificar el rendimiento en un NCS

En [13] se presenta un mecanismo de ejecución alternativo para cada lazo de control en red que permite reducir el uso de ancho de banda de red al modificar el algoritmo del nodo sensor, mientras que garantiza el mismo o mejor rendimiento de control que el logrado por un caso periódico. Para aplicar esta nueva regla de ejecución, se utiliza un paradigma de ejecución de cada lazo de control activado por eventos, en lugar de un paradigma activado por tiempo, esto debido a que el control disparado por eventos ha demostrado ser una técnica prometedora para reducir la demanda computacional de los controladores. Esta técnica consiste en que el nodo sensor busca el tiempo de activación del lazo siguiente más lejano que se pueda aplicar, considerando que el rendimiento resultante no disminuya con respecto al desempeño dado por un controlador periódico, el peor de los casos a ejecutarse en este enfoque, es cuando el lazo de control se producirá una vez transcurrido el periódo de muestréo, de lo contrario, este se ejecutará luego, reduciendo así la tasa de tráfico en la red.

Para analizar esta técnica se considera que la planta para el sistema de control en red se describe mediante un sistema lineal de tiempo continuo

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.16}$$

$$y(t) = Cx(t)$$

donde $x \in \mathbb{R}^{(n\times 1)}$ es el estado de la planta, $u \in \mathbb{R}$ es la señal de control (entrada), $A \in \mathbb{R}^{(n\times n)}$ y $B \in \mathbb{R}^{(n\times 1)}$ son las matrices de sistema y de entrada, respectivamente, y $C \in \mathbb{R}^{(1\times n)}$ es la matriz de salida. Dejando

$$u(t) = u(k) = Lx(t_k) = Lx_k \quad \forall t \in [t_k, t_k + 1) \tag{2.17}$$

sea las actualizaciones de control dadas por un controlador de retroalimentación lineal L diseñado en el dominio de tiempo discreto usando solo muestras del estado en instantes discretos $t_0, t_1 ..... t_k$.

Por otro lado, para cada lazo de control en red, en lugar de aplicar el control periódico, se interesa lograr un patrón de ejecución no periódico, basado en:

1. ampliando en cada ejecución del controlador el intervalo de muestreo actual
2. mientras que proporciona el mismo o mejor rendimiento de control que el caso periódico
3. suponiendo que la ganancia L del controlador diseñada para el escenario periódico se mantiene constante en el nuevo enfoque de ejecución no periódica
4. y considerando que las próximas activaciones después de la actual serán periódicas.

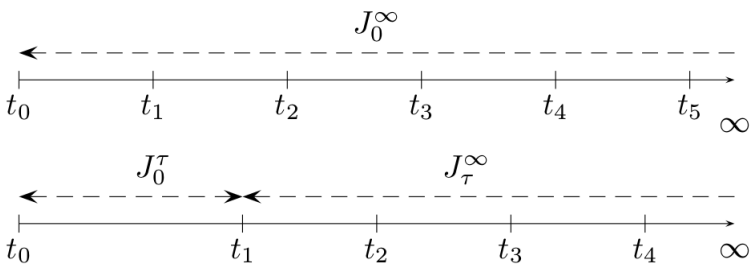


Figura 2.4: Patrones de ejecución periódica (línea superior), versus no periódica (línea inferior)

La línea superior de la Figura 2.4, ejemplifica la ejecución periódica de un lazo de control, con $t_{k+1} - t_k = h$ donde $k = 0, ..., \infty$, evaluando su desempeño con una función de costo J evaluada a partir del tiempo actual $(t_0 = 0)$ a infinito, etiquetado como J_0^∞. La línea inferior ejemplifica el enfoque propuesto. Denotando $\tau = t_1 - t_0$ el intervalo de tiempo desde el tiempo actual t_0 hasta el siguiente tiempo de activación t_1. Durante el intervalo de tiempo τ, la ejecución actual del lazo de control está teniendo lugar. Desde t_1 a infinito, se supone que las ejecuciones del lazo de control son periódicas con el período anterior, es decir, suponiendo que $t_{k+1} - t_k = h$ con $k = 1, ..., \infty$. Para este enfoque, J_0^τ denota la función de costo J evaluada durante τ, y J_τ^∞ denota el costo restante J evaluado de t_1 a infinito debido a las próximas ejecuciones periódicas asumidas. Para simplificar, se usa

$$J(\tau) = J_0^\tau + J_\tau^\infty \tag{2.18}$$

para denotar la evaluación del tiempo actual $t_0 = 0$ hasta el infinito de la función de costo J al usar el patrón de ejecución no periódica del enfoque presentado. Para un lazo de control dado, el problema a resolver en cada ejecución del lazo de control es encontrar el valor τ más largo de manera que

$$J(\tau) \leq J_0^\infty \text{ con } \tau \geq h. \tag{2.19}$$

2.3.1. Problema de optimización

Encontrar el τ más largo que satisface 2.19 se puede transformar en un problema de optimización que simplifica la búsqueda de τ. El problema de optimización puede formularse como

$$\text{maximizar} \qquad \tau \tag{2.20}$$

$$\text{sujeto a} \quad \frac{dJ(\tau)}{d\tau} = 0 \tag{2.21}$$

$$J_0^{\infty} - J(\tau) \geq 0 \tag{2.22}$$

$$\tau > h \tag{2.23}$$

$$x(\tau) = \Phi(\tau)x_0 + \Gamma(\tau)u_0 \tag{2.24}$$

$$\text{donde} \qquad u_0 = Lx_0 \tag{2.25}$$

$$\text{y} \qquad \Phi(\tau) = e^{A\tau}, \Gamma(\tau) = \int_0^{\tau} e^{As} ds B, \, \tau \in [t_k, t_{k+1}). \tag{2.26}$$

La función 2.20 indica que en cada ejecución de lazo cerrado deberíamos encontrar el mayor τ cuyo costo $J(\tau)$ es igual o menor que el costo J_0^{∞} dado por el enfoque de ejecución periódica, restricción especificada por 2.22.

La restricción 2.21 reduce el conjunto de restricciones a un conjunto de puntos aislados (puntos críticos) que puede contener un máximo y un mínimo para J.

La restricción 2.23 asegura que el siguiente tiempo de activación ocurrirá más tarde que el tiempo especificado por h, y la restricción 2.24 modela aún más la búsqueda de τ porque está restringida a la dinámica específica de ciclo cerrado en consideración.

Para solucionar el problema de optimización 2.20-2.24 con el caso de que la función de costo J y la ganancia de controlador L son las estándar usadas en un ajuste de control óptimo. Por lo tanto, para cada lazo de control 2.16, el rendimiento del control se mide mediante una

función de costo cuadrático continua de horizonte infinito

$$J = \int_0^{\infty} [x^T(t)Q_c x(t) + u^T(t)R_c u(t) + 2x^T(t)N_c u(t)]dt, \tag{2.27}$$

donde las matrices de ponderación Q_c y R_c son matrices semi-definidas positivas simétricas [14].

La evaluación de 2.27 varia si se adopta un enfoque de ejecución periódico o aperiódico. Para el caso periódico, si la ganancia L en 2.17 se diseña utilizando control lineal cuadrático (LQR), con un periodo de muestreo dado h, la evaluación de 2.27 es

$$J_0^{\infty} = x_0^T S(h) x_0 \tag{2.28}$$

donde $S(h)$ es la solución a la ecuación algebraica de Riccati, y x_0 es un estado inicial dado.

Para una ejecución aperiódica, y usando la misma ganancia L que antes, la evaluación del costo 2.27 se puede especificar en términos de J_0^{τ} y J_{τ}^{∞} como en 2.18, siendo

$$J = \int_0^{\tau} [x^T(t)Q_c x(t) + u^T(t)R_c u(t) + 2x^T(t)N_c u(t)]dt \tag{2.29}$$

y

$$J_{\tau}^{\infty} = x^T(t)S(h)x(t). \tag{2.30}$$

Al observar el problema de optimización 2.20-2.24, el primer paso es encontrar los puntos críticos de $J(\tau)$, es decir, resolver la restricción 2.21 dando como resultado

$$[x(t)\ u_0] \begin{bmatrix} \bar{Q} & \bar{N} \\ \bar{N}^T & \bar{R} \end{bmatrix} \begin{bmatrix} x(t) \\ u_0 \end{bmatrix} = 0, \tag{2.31}$$

donde

$$\bar{Q} = [2SA + Q_c],\ \bar{N} = [N_c + SB],\ \bar{R} = R_c.$$

Para resolver 2.31, $x(\tau)$ debe hacerse explícitamente en función de τ, y luego resuelve para τ. Un enfoque general puede ser el usar una aproximación de orden n de $x(\tau)$ si no existe una expresión exacta para $x(\tau)$. En primer lugar, $\Phi(\tau)$ y $\Gamma(\tau)$ en 2.24 pueden escribirse en términos de $\Psi(\tau)$ como [14]

$$\Phi(\tau) = e^{A_\tau} = I + A\Psi(\tau) \tag{2.32}$$

$$\Gamma(\tau) = \int_0^\tau e^{As} B\, ds = \Phi(\tau) B \tag{2.33}$$

donde $\Psi(\tau)$ es

$$\Psi(\tau) = \int_0^\tau e^{As} ds = \sum_{i=0}^{\infty} \frac{A^i \tau^{i+1}}{(i+1)!}. \tag{2.34}$$

Teniendo en cuenta 2.25 y sustituyendo 2.32 y 2.33 en la restricción 2.24 se obtiene

$$x(\tau) = x_0 + A\Psi(\tau)x_0 + \Psi(\tau)BLx_0. \tag{2.35}$$

Las matrices de conmutación $\Psi(\tau)$ y A, 2.35 se puede escribir como

$$x(\tau) = x_0 + \Psi(\tau)[A + BL]x_0. \tag{2.36}$$

Finalmente, se aproxima $\Psi(\tau)$ por una serie de Taylor de orden n alrededor de x_0 en 2.36, obteniendo así

$$x(\tau) = x_0 + (A + BL)x_0\tau + \frac{A[A+BL]}{2!}x_0\tau^2 + \frac{A^2[A+BL]}{3!}x_0\tau^3 + \ldots \tag{2.37}$$

2.3.2. Resumen de la nueva técnica

La solución explicita presentada anteriormente se puede resumir a continuación:

1. Dado un estado inicial x_0, calcule $x(\tau)$ como en 2.37 si no existe una solución exacta, y calcule u_0 como en 2.25 considerando que L está diseñado para un período de muestreo dado h.

2. Calcule 2.31 y resuelva para τ, y mantenga los verdaderos positivos más largos que h.

3. Calcule el costo de cada valor τ, como en 2.18 considerando 2.29 y 2.30, y calcule el costo del controlador periódico, como en 2.28. Luego mantenga esos valores τ que satisfagan la restricción 2.22 y elija el más grande.

4. Si en cualquier paso el conjunto de valores para τ está vacío, tome $\tau = h$.

2.3.3. Pseudocódigo de la nueva regla de ejecución

Suponiendo un modelo simple de ejecución para cada circuito de control en red, el algoritmo 2 ilustra los códigos básicos que se ejecutarán en los nodos de sensor, controlador y actuador si se considera un enfoque periódico.

Algoritmo 2.1: Pseudocódigo para el nodo sensor disparado por interrupción cada h

begin

$X_k := sample()$;

send sensor message (X_k);

end

Algoritmo 2.2: Pseudocódigo para el nodo controlador, disparado por recepción de mensaje

begin

$X_k := receive\ sensor\ message()$;

$U_k := compute\ control\ signal(L, X_k)$;

send controller message (U_k);

end

Algoritmo 2.3: Pseudocódigo para el nodo actuador, disparado por recepción de mensaje

begin

$(U_k) := receive\ controller\ message()$;

apply control signal (U_k);

end

Algoritmo 2: Pseudocódigo para control periódico

La implementación del enfoque presentado modifica el pseudocódigo del nodo del sensor como se ilustra en el algoritmo 3. En este caso, el nodo del sensor se dispara de acuerdo con τ, que puede variar en cada ejecución del sensor. Y la ejecución del cálculo τ, que da la solución al problema de optimización 2.20-2.24, es la nueva técnica presentada en la subsección 2.3.2.

1: **begin**
2: $X_{(k)} := sample();$
3: $\tau := compute\ \tau(X_k, U_k);$
4: $sendsensormessage\ (X_k);$
5: $resetinterrupt\ (\tau);$
6: **end**

Algoritmo 3: Pseudocódigo con la nueva regla de ejecución disparado por interrupción cada τ

2.4. Control predictivo en red basado en eventos

El método de control predictivo en red (NPC) [15], es un modelo basado en un algoritmo de control predictivo y aprovecha la característica de que un paquete de datos se puede transferir simultáneamente en la red. Este nuevo método presentado en esta investigación se basa en eventos donde un grupo de secuencias de control futuras para cada delay de tiempo posible son empaquetados y envíados a la planta a traveś de la red. En el lado de la planta se selecciona la señal de control basandose en las salidas del sistema *y* almacenadas en lugar de seleccionarse con la medición del retardo de transmisión como en [16] y posteriormente aplicada a la planta.

La Figura 2.5, muestra la estructura del sistema NPC basado en eventos, el cual se puede dividir en dos secciones. Una sección es el lado de la planta, el cual se compone de un compensador de retardo de red y un búfer del actuador. En la otra sección se encuentra el controlador basado en eventos, el cual consiste de un generador de predicción de control y un identificador en linea.

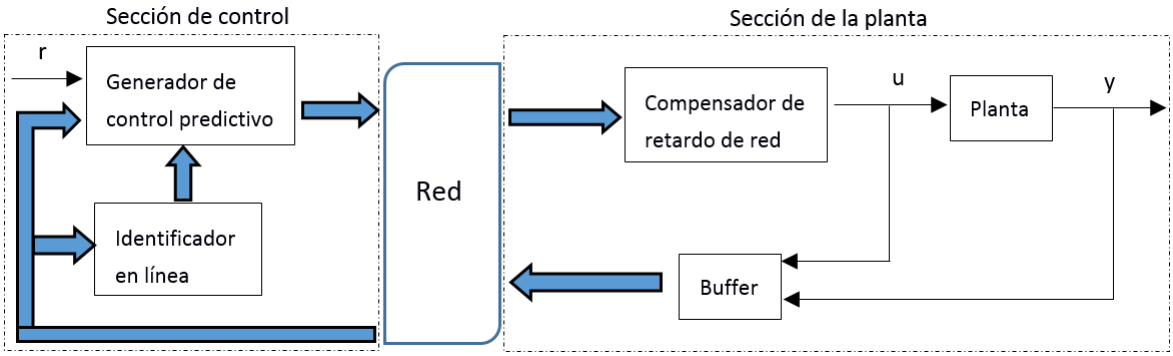


Figura 2.5: Estructura de una NPC basado en eventos

2.4.1. Generador de control predictivo basado en eventos

El generador de control trabaja solo cuando recibe del lado de la planta los datos del canal de retroalimentación, caso contrario, sin recibir datos nuevos del lado de la planta está en estado inactivo. Sea $\mathbb{R}[z^{-1},n]$ el conjunto de polinomios en el indeterminado z^{-1} con coeficientes en el campo de los números reales y con el orden n en un conjunto de números enteros no negativos. Por ejemplo, un polinomio $A(z^{-1}) \in \mathbb{R}[z^{-1},n]$, es decir, $A(z^{-1}) = a_0 + a_1 z^{-1} + ... + a_n z^{-n}$. Además considerando una planta SISO de tiempo discreto definida por el modelo de promedio móvil autoregresivo

$$A(z^{-1})y(t) = B(z^{-1})u(t-1) \tag{2.38}$$

donde $u(t)$ y $y(t)$ son la entrada y salida de lazo abierto de la planta, y $A(z^{-1}) \in \mathbb{R}[z^{-1},n]$ y $a_0 = 1$ y $B(z^{-1}) \in \mathbb{R}[z^{-1},n]$ son los polinomios del sistema.
Sin considerar el retraso de transmisión de la red, un controlador está diseñado como

$$C(z^{-1})u(t) = D(z^{-1})e(t) \tag{2.39}$$

donde los polinomios $C(z^{-1}) \in \mathbb{R}[z^{-1},n_c]$ y $c_0 = 1$ y $D(z^{-1}) \in \mathbb{R}[z^{-1},n_d]$, y

$$e(t) = r(t) - y(t) \tag{2.40}$$

donde $y(t)$ es la predicción de salida, y $r(t)$ es la entrada de referencia.

Denotando que t_{sc} es el retardo de tiempo del canal de retroalimentación. En el momento t, el lado del controlador recibe un paquete del lado de la planta. Incluido en este paquete están las secuencias de salidas de la planta y [incluyendo $y(t-t_{sc}), y(t-t_{sc}-1), ..., y(t-t_{sc}-n)$], y las secuencias de control previas u [incluyendo $u(t-t_{sc}-1), u(t-t_{sc}-2), ..., u(t-t_{sc}-n_c)$], e indica el tiempo al que se empaqueta y envia el paquete.

Para simplicidad del análisis, se asume que el máximo retardo de tiempo esta limitado por N-pasos. Definiendo la operación en las predicciones como

$$x(t+i|t) = q^{-1}x(t+i+1|t), \quad \text{for } i = 0, 1, ... \tag{2.41}$$

$$x(t-1) = q^{-1}x(t|t) \tag{2.42}$$

$$x(t-i-1) = q^{-1}x(t-i), \quad \text{for } i = 1, 2, ... \tag{2.43}$$

para $i = 0, 1, 2, ...$, donde $x(.)$ representa $y(.)$ o $u(.)$, y $x(t+i|t)$ denota la i-enésima predicción de paso futuro de $x(t)$ basado en los datos previos hasta el momento t.

Para predecir la secuencia de control futura, tenemos

$$y(t-t_{sc}|t-t_{sc}) = y(t-t_{sc}). \tag{2.44}$$

La secuencia de control correspondiente puede ser calculado como

$$u(t-t_{sc}|t-t_{sc}) = (1-C(q^{-1}))u(t-t_{sc}|t-t_{sc}) + D(q^{-1})(r(t-t_{sc}) - y(t-t_{sc}|t-t_{sc})). \tag{2.45}$$

Es muy simple mostrar que recursivamente la salida futura de la planta es

$$y(t-t_{sc}+k|t-t_{sc}) = (1-A(q^{-1}))y(t-t_{sc}+k|t-t_{sc})+B(q^{-1})(u(t-t_{sc}+k-1|t-t_{sc}) \quad (2.46)$$

donde $k = 0,1,...,N-1$, y la señal de control futura es

$$u(t-t_{sc}+k|t-t_{sc}) = (1-C(q^{-1}))u(t-t_{sc}+k|t-t_{sc})+D(q^{-1})(r(t-t_{sc}+k)-y(t-t_{sc+k}|t-t_{sc})) \quad (2.47)$$

donde $k = 0,1,...,N-1$.

Después de un cálculo de N-pasos, se obtiene la secuencia de control futura $U(t-t_{sc}|t-t_{sc})$ y la secuencia de salida del sistema futuro $Y(t-t_{sc}|t-t_{sc})$, donde

$$U(t-t_{sc}|t-t_{sc}) = [u(t-t_{sc}|t-t_{sc}), u(t-t_{sc}+1|t-t_{sc}), ..., u(t-t_{sc}+N-1|t-t_{sc})]^T \quad (2.48)$$

$$Y(t-t_{sc}|t-t_{sc}) = [y(t-t_{sc}|t-t_{sc}), y(t-t_{sc}+1|t-t_{sc}), ..., y(t-t_{sc}+N-1|t-t_{sc})]^T. \quad (2.49)$$

Luego de haberse generado 2.48 y 2.49, son empaquetados juntos y enviados a la sección de la planta.

2.4.2. Compensador de retardo de red

La tarea del compensador es la de elegir la señal apropiada para la secuencia futura de control predecido, considerando dos escenarios.

1. **El paquete de control se recibe durante el ciclo de control**

Si se lo recibe durante el ciclo de control en el tiempo t del lado de la planta, el compensador debe elegir la señal de control apropiada de la secuencia entrante. La selección se basa en la

salida previa del sistema $y(t-1), y(t-2), ...y(t-n)$ que están almacenadas en el búfer de la planta.

Para encontrar la mejor señal de control para el actuador se compara la secuencia de predicción de salida $y(t-t_d+k-1|t-t_d), y(t-t_d+k-2|t-t_d), ...y(t-t_d+k-m|t-t_d)$ con la salida de la planta almacenada actualmente $y(t-1), y(t-2), ...y(t-m)$, donde $k = m, m+1, ..., N$ y m es la profundidad de comparación, seleccionando y aplicando la salida de control correspondiente $u(t-t_d+k|t-t_d)$.

El detalle del algoritmo de ajuste se describe a continuación. Si la profundidad de comparación es m, entonces el coeficiente de peso se define como $\alpha 1, \alpha 2, ..., \alpha m$, donde αk representa la importancia relativa de la salida previa de $k-pasos$ a la salida del controlador actual. La fórmula de comparación es

$$\begin{aligned} J_k = {} & \alpha_1 |y(t-t_d+k-1|t-t_d) - y(t-1)| + \alpha_2 |y(t-t_d+k-2|t-t_d) - y(t-2)| \\ & + ...\alpha_m |y(t-t_d+k-m|t-t_d) - y(t-m)|. \end{aligned} \tag{2.50}$$

Cada $J_1, J_2, ..., J_{N-m}$ se calcula, respectivamente, y se compara entre sí. El J_k más pequeño es seleccionado. De forma correspondiente, el valor k de la secuencia de control $U(t-t_d|t-t_d)$ se elige como la señal del actuador.

2. **El paquete de control no se recibe durante el ciclo de control**

Debido al rendimiento aleatorio de la red, es posible que no se reciba un paquete de control durante el ciclo de control. En esta situación, se debe comparar la secuencia previa $U(t-t_d-1|t-t_d-1)$ y $Y(t-t_d-1|t-t_d-1)$. La adaptación del enfoque descrito previamente debería dar como resultado el mismo resultado que si el paquete de control se recibe durante el ciclo de control. La fórmula de comparación es

$$\begin{aligned} J_k = {} & \alpha_1 |y(t-t_d+k-2|t-t_d-1) - y(t-1)| + \alpha_2 |y(t-t_d+k-3|t-t_d-1) - y(t-2)| \\ & + ...\alpha_m |y(t-t_d+k-m-1|t-t_d-1) - y(t-m)|. \end{aligned} \tag{2.51}$$

Los valores $J_1, J_2, ..., J_{N-m}$ se comparan, y se elige el valor J_k más pequeño y se selecciona valor correspondiente en la secuencia de control $U(t-t_d+k-1|t-t_d)$.

2.4.3. Búfer del actuador

Luego de seleccionarse la señal de control u, esta es almacenada con la señal de salida y. La longitud del búfer se determina por la descripción del modelo y el método de control seleccionado, por ejemplo las siguientes secuencias están en cola en el búfer

$$\begin{bmatrix} y(t) \\ y(t-1) \\ \vdots \\ y(t-n) \end{bmatrix}, \begin{bmatrix} u(t-1) \\ u(t-2) \\ \vdots \\ u(t-m-1) \end{bmatrix}. \tag{2.52}$$

Las secuencias almacenadas en el búfer son requeridas para la predicción de las secuencias futuras y además de la selección de la señal adecuada de estas secuencias. Este proceso se repite cuando se inicia un nuevo ciclo de control.

2.4.4. Identificador de parámetros on-line

La precisión del modelo es importante para el rendimiento de los sistemas NPC incluso con este nuevo algoritmo de selección. Al no ser este modelo preciso, la calidad del control se degradada e incluso puede inestabilizar al sistema de control. Dado que los sistemas de la planta son invariables y ligeramente no lineales y tienen parámetros que son variables, dependiendo de las condiciones de operación, entonces el modelo que representa la planta debe seguir estos cambios. Por lo tanto, se adopta un estimador de parámetros de mínimos cuadrados recursivos en el esquema de control [18] . La planta se describe como

$$(1+a_1z^{-1}+\ldots+a_nz^{-n})y(t+d) = (b_0+B_1z^{-1}+\ldots+b_mz^{-m})u(t). \tag{2.53}$$

El algoritmo se puede escribir como

$$\begin{aligned}
\hat{\theta}(t) &= \hat{\theta}(t-1) + L(t)[y(t) - \varphi^T(t)\hat{\theta}(t-1)] \\
K(t) &= \frac{P(t-1)\varphi(t)}{\lambda + \varphi^T(t)P(t-1)\varphi(t)} \\
P(t) &= [P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda + \varphi^T(t)P(t-1)\varphi(t)}]/\lambda
\end{aligned} \tag{2.54}$$

donde el valor inicial del vector estimado $\hat{\theta} = [-a_1, -a_2, \ldots, -a_n, b_0, b_1, \ldots, b_n]^T$, el vector regresión es

$$\varphi(t) = [y(t-1), y(t-2), ..., y(t-n), u(t-d), u(t-d-1), ..., u(t-d-m)]^T \tag{2.55}$$

y λ es el factor de olvido. El vector de regresión $\varphi(t)$ y $y(t)$ se obtiene del paquete enviado desde el lado de la planta. Se almacenan en la memoria intermedia del actuador. $\varepsilon(t)$ es la diferencia entre la salida real y la predicción de un paso $\varphi^T(t)\hat{\theta}(t-1)$. Cuando $\varepsilon(t)$ es grande, indica que el modelo actual no es exacto. En este caso, el vector de parámetros $\hat{\theta}(t)$ realizará los cambios correspondientes para ajustar los parámetros del modelo.

Capítulo 3

Metodología

En este capítulo se realiza una revisión condensada de los lazos de control en red sincronizados en los instantes de actuación y posteriormente se establecen las directrices de implementación del mismo.

3.1. Antecedentes

3.1.1. Arquitectura de una NCS

Un lazo de control simple basado en una NCS tiene una arquitectura como se muestra en la Figura 3.1, donde el sensor, controlador y actuador intercambian datos a travéz de mensajes de red. El enviar un mensaje sobre una red toma un tiempo, por lo tanto en una NCS los retardos de tiempo son introducidos en cada lazo de control. En este tipo de arquitectura se identifican tres tipos de retardos de tiempo: retardo entre el sensor y controlador, τ_{sc}, retardo en el controlador, τ_c, y el retardo entre el controlador y el actuador, τ_{ca}. Dentro de cada período de muestreo h los tres retardos de tiempo dan como resultado un retardo acumulado representado por

$$\tau = \tau_{sc} + \tau_c + \tau_{ca}, \tag{3.1}$$

donde τ_{sc} y τ_{ca} depende de la red de comunicación y τ_c depende del algoritmo de control y el nodo de procesamiento.

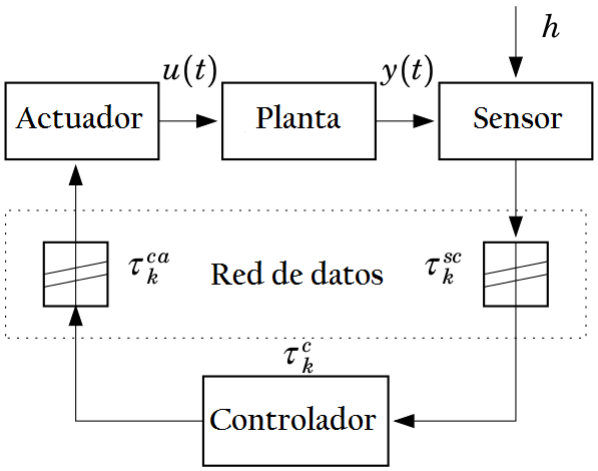


Figura 3.1: Arquitectura común de una NCS

3.1.2. Modelo de control fundamental con retardo de tiempo

El enfoque común para diseñar sistemas de control en red tiene dos pasos fundamentales. El primero es obtener el modelo discreto de la planta y el segundo es diseñar una ley de control de tiempo discreto para el modelo de la planta en tiempo discreto. Un enfoque tradicional sugiere que en el lazo de control se mantenga un muestréo y actuación periódica.

Considérese la descripción matemática tradicional de una planta dada por el modelo de espacio de estados de un sistema de tiempo discreto lineal invariante en el tiempo con un período de muestreo h [14]

$$x_k + 1 = \Phi(h)x_k + \Gamma(h)u_k \tag{3.2}$$

$$y_k = Cx_k,$$

donde x_k es el estado de la planta, u_k y y_k son las entradas y salidas de la planta, la matriz

$C \in \mathbb{R}^{p\times n}$ es la matriz de salida y las matrices $\Phi(t)$ y $\Gamma(t)$ se obtienen mediante

$$\Phi(t) = e^{At},\ \ \Gamma(t) = \int_0^{\tau} e^{As} B\, ds, \tag{3.3}$$

con $t = h$, donde $A \in \mathbb{R}^{n\times n}$ y $B \in \mathbb{R}^{n\times m}$ son las matrices de dinámica del sistema y las matrices de entrada en la forma de tiempo continuo

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t) \tag{3.4}$$

$$y(t) = Cx(t).$$

Las variables m, n y p denotan las dimensiones de los estados, entradas y salidas respectivamente.

Para la operación de un lazo de control estándar de 3.2, el comando de control u_k se calcula por

$$u_k = Lx_k \ \text{ con } \ L \in \mathbb{R}^{1\times n}, \tag{3.5}$$

donde L es la ganancia de realimentación de estado obtenida usando métodos de diseño de control estándar de las matrices $\Phi(h)$ y $\Gamma(h)$. La aplicación de 3.5 a la planta dispone calcular el comando de control con *tiempo cero*, lo cual es físicamente imposible incluso para un lazo de control basado en procesador, debido a que ejecutar el algoritmo de control toma cierto tiempo. El modelo 3.2 puede ser modificado para hacer frente a un retraso de tiempo que modela una latencia de Entrada/Salida que aparece debido al cálculo del algoritmo de control o debido a la inserción de una red en un lazo de control, obteniendo un modelo estándar que incorpora un delay de tiempo τ, con $\tau \leq h$, es [19]

$$x_{k+1} = \Phi(h)x_k + \Phi(h-\tau)\Gamma(\tau)u_{k-1} + \Gamma(h-\tau)u_k. \tag{3.6}$$

El modelo 3.6 se utiliza como modelo de control simple fundamental para el diseño y análisis de NCS. Este modelo asume una referencia de tiempo dado por los instantes de muestreo

con un retardo de tiempo fijo desde el muestreo hasta la actuación, es decir, el sincronismo esta dado por los instantes de muestreo. La aplicación de este modelo en un lazo de control en red es usualmente infringido, debido a que los retardos de tiempo insertan impredecibilidad de tiempo desde el muestreo hacia la actuación.

3.1.3. Modelo de control con sincronización en los instantes de actuación

Para hacer frente a los problemas que generan los retardos de tiempo en un lazo de control en red, se ha modificado el modelo de la sección 3.1.2, permitiendo de esta forma eliminar el retardo de tiempo de Entrada/Salida, y basar su funcionamiento en solo un punto de sincronización (instante de actuación) permitiendo de esta manera asimilar muestreos con tiempos irregulares [19].

Sincronizando la operación de cada lazo de control en los instantes de actuación, se obtiene que el tiempo transcurrido entre cada instante de actuación consecutivo t_{k-1} y t_k, es el período de muestreo h. Dentro de este intervalo de tiempo el estado de la planta es muestreado, $x_{s,k} \in (t_{k-1}, t_k)$, y la marca de tiempo a la cual se tomó la muestra, $t_{s,k}$. La diferencia entre este tiempo y el tiempo del siguiente instante de actuación

$$\tau_k = t_k - t_{s,k} \tag{3.7}$$

es usado para estimar el estado en el instante de actuación como

$$\hat{x}_k = \Phi(\tau_k) x_{s,k} + \Gamma(\tau_k) u_{k-1}. \tag{3.8}$$

Haciendo uso de 3.8, se calcula la acción de control como

$$u_k = L\hat{x}_k \ \text{ con } \ L \in \mathbb{R}^{1\times n} \tag{3.9}$$

donde L es la ganancia del controlador original como en 3.5. La acción de control u_k se mantiene constante entre cada instante de actuación. Además, la acción de muestreo no se requiere que

sea periódica, debido a que τ_k en 3.7 puede variar en cada operación del lazo de control.

3.2. Directrices de implementación

3.2.1. Algoritmo de control

El algoritmo de control basado en 3.7 - 3.9 se fundamenta en mediciones de tiempo absolutas y mensajes de control como lo ilustra la Figura 3.2 , donde las letras A, S, y C representan la ejecución del código en el actuador, sensor y controlador respectivamente. Y los números 1, 2 y 3 representan los mensajes requeridos para cada operación de lazo de control. Cada operación

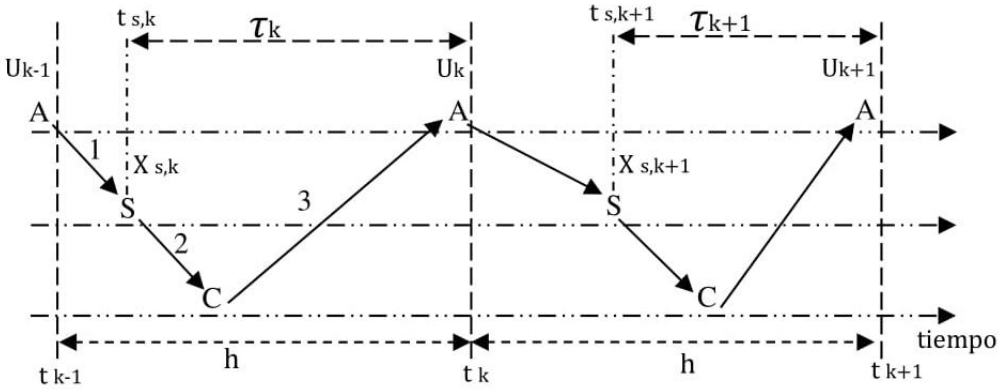


Figura 3.2: Operación del algoritmo de control

de lazo de control se inicia por el nodo actuador luego de aplicar la señal de control u_{k-1}, inicia el siguiente instante de actuación t_k, cuyo valor es enviado al sensor (mensaje 1). El sensor al recibir el mensaje, muestrea la planta $x_{s,k}$ y obtiene la marca de tiempo absoluto $t_{s,k}$. Este último juntamente con t_k es usado para calcular τ_k para luego enviar al controlador $x_{s,k}$ y τ_k (mensaje 2). El controlador estima el estado de la planta que se aplicara en el instante t_k, 3.8, y calcula la acción de control u_k, 3.9. La acción de control u_k es enviada al actuador (mensaje 3), el cual lo aplicará a la planta en el tiempo de actuación sincronizado. El algoritmo 4, presenta los pseudocódigos necesarios para ejecutarse el lazo de control en red utilizando el enfoque actual.

Algoritmo 4.1: Nodo sensor, disparado por mensaje
begin
mientras (no haya mensaje del actuador) hacer (nada);
t_k := leer el mensaje;
$(x_{s,k}, t_{s,k})$:= lee el estado junto con la marca de tiempo;
$\tau_k := t_k - t_{s,k}$;
envía el mensaje al controlador $(x_{s,k}, \tau_k)$;
end

Algoritmo 4.2: Nodo controlador, disparado por mensaje
begin
mientras (no haya mensaje del sensor) hacer (nada);
$(x_{s,k}, \tau_k)$:= leer el mensaje;
$\hat{x}_k := \Phi(\tau_k)x_{s,k} + \Gamma(\tau_k)u_{k-1}$;
$u_k := L\hat{x}_k$;
$u_{k-1} := u_k$;
envía el mensaje al actuador (u_k);
end

Algoritmo 4.3: Nodo actuador, sincronizado por h
begin
mientras (no haya mensaje del controlador) hacer (nada);
u_k := leer el mensaje;
esperar a la interrupción de actuación;
aplicar a la planta u_k;
$t_k := t_k + h$;
envía el mensaje al sensor t_k;
end

Algoritmo 4: Pseudocódigo del sensor, controlador y actuador

3.2.2. Sincronización de reloj

Como se ha mencionado previamente, el modelo de control con sincronización en los instantes de actuación se basa su funcionamiento en mediciones de tiempo absolutas y mensajes, siendo escencial para el correcto funcionamiento. Por lo tanto la sincronización de reloj y la programación de la red son de vital importancia.

Para llevar acabo este tipo de sincronización en el lazo de control en red, se asumirá que los nodos sensor, controlador y actuador están basados en una red CAN. Además que estén sincronizados usando el protocolo IEEE 1588 PTP (Presicion Time Protocol). Este protocolo asegura que los eventos y marcas de tiempo en los nodos usen la misma base de tiempo. En [20] se detalla la aplicación de este protocolo en microcontroladores de bajo costo, los cuales típicamente funcionan a bajas frecuencias de reloj con una resolución de tiempo pobre, especialmente en el rango de los milisegundos. Teniendo en cuenta esta limitación, la precisión de sincronización para nuestra implementación es de 1 ms. Esta precisión, aunque dista mucho de la precisión de un microsegundo, que puede alcanzar este protocolo, en muchos casos es suficiente para los propósitos de control.

El protocolo detalla que para llevar a cabo el proceso de sincronización cada reloj de los nodos esclavo se deben sincronizar con el reloj del nodo maestro a través de mensajes de sincronización. El proceso de sincronización se lleva a cabo en dos fases.

1. **Corrección de la diferencia de tiempo entre el maestro y esclavos.** Como se detalla en la Figura 3.3 la medición de compensación se lleva a cabo por medio de mensajes de sincronización (TM1) el cual contiene el tiempo de transmisión. El esclavo al recibir este mensaje mide el tiempo de recepción, luego el maestro envía un segundo mensaje (mensaje de seguimiento) con el tiempo exacto de transmisión de TM1, con estos parametros el esclavo calcula la compensación para aplicar al reloj. Este proceso se lleva a cabo en intervalos de 2 segundos, además, si en la red de comunicación no se produce ningún delay de transmisión de mensaje, los relojes ya se habrán sincronizado.

2. **Medición de la latencia entre el esclavo y el maestro.** Como se muestra en la Figura 3.4

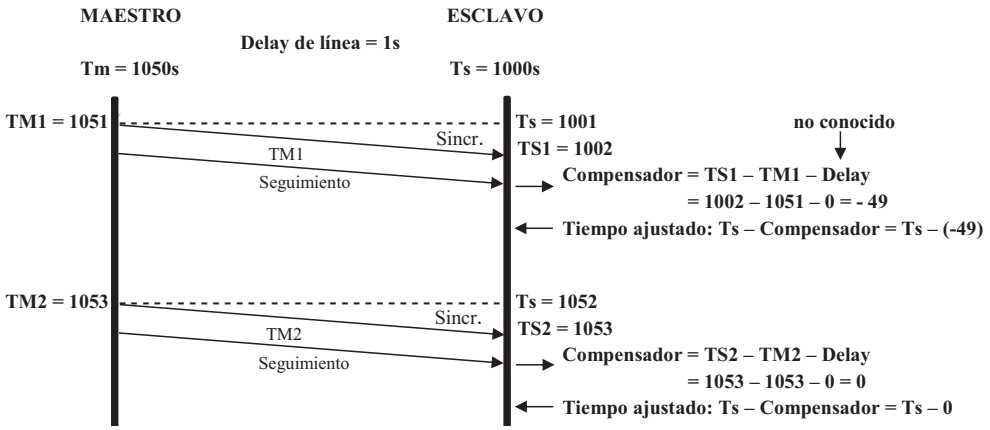


Figura 3.3: Corrección de diferencia de tiempo

el esclavo envía un mensaje de solicitud de delay (TS3) hacia el maestro, el cual contiene el tiempo exacto de transmisión. El maestro tras recibir el mensaje determinará el tiempo de recepción de TS3 y lo enviará de regreso en un mensaje de respuesta del retardo de tiempo (TM3). Con la ayuda de estos dos parametros, el esclavo puede calcular el retardo de tiempo entre el esclavo y el maestro. La medición de este parametro se lo realiza en intervalos que pueden ser desde 4 hasta 60 segundos.

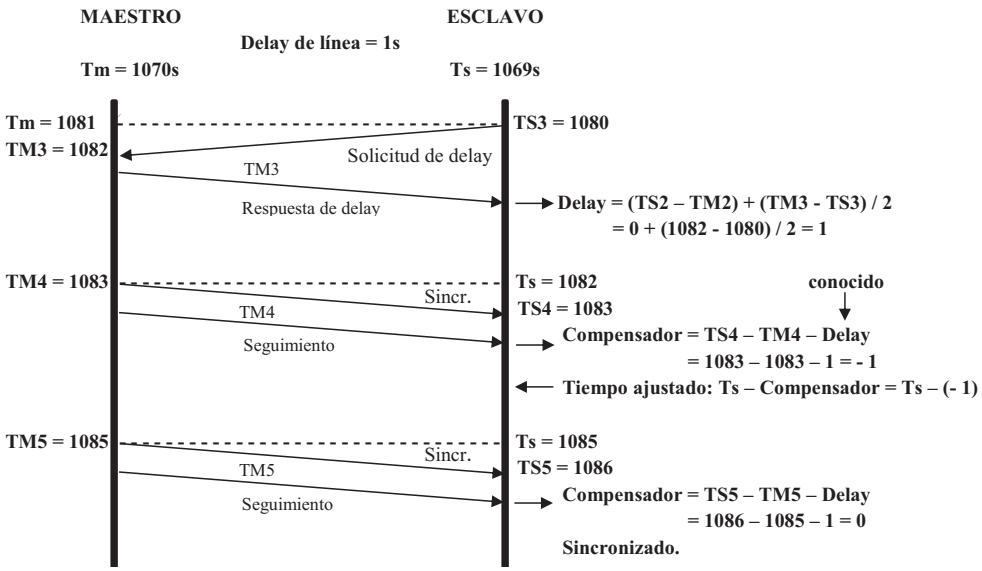


Figura 3.4: Medición de delay

3.2.3. Protocolo CAN

El protocolo CAN especifica que el mensaje con el identificador más bajo tiene la mayor prioridad, por lo cual los conflictos de acceso al bus es resuelto por arbitraje de los identificadores. Tomando en cuenta esta regla se determina a los identificadores de mensaje de la siguiente forma:

- Los mensajes de sincronización de reloj tienen la prioridad más alta.
- Todos los mensajes enviados por el nodo actuador tienen el siguiente nivel de prioridad, debido a que estos mensajes disparan la acción de operación de cada lazo de control.
- Los mensajes enviados por el nodo sensor tienen el siguiente nivel de prioridad.
- Por último, los mensajes enviados por el nodo controlador tienen el último nivel de prioridad.

Esta regla de especificación de los identificadores de mensaje, es similar a la presentada en [21].

Capítulo 4

Implementación y pruebas

En este capítulo, la teoría desarrollada hasta ahora es validada a través de una simulación en el software Matlab y posteriormente aplicada en un ejemplo numérico, específicamente sobre un circuito electrónico doble integrador. Además, se analizan los resultados obtenidos 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. Para lo cual se hace uso de la plataforma Arduino UNO y la shield CAN-BUS DEV-10039. Cada shield CAN-BUS va apilada sobre la plataforma Arduino, permitiendo la comunicación mediante el bus CAN con el resto de nodos en la red.

En la Figura 4.1, se presenta el diagrama de conecciones del sistema para llevar a cabo la implementación del algoritmo 4. Nótese a) la red CAN-BUS, la cual es conformada por tres Arduino UNO y tres shield CAN-BUS, que comprenden los nodos sensor, controlador y actuador, necesarios para implementar el algoritmo 4. Además de b) la planta doble integradora y c) el dispositivo myRIO, el cual realiza la supervisión de la planta para comprobar el funcionamiento de la misma.

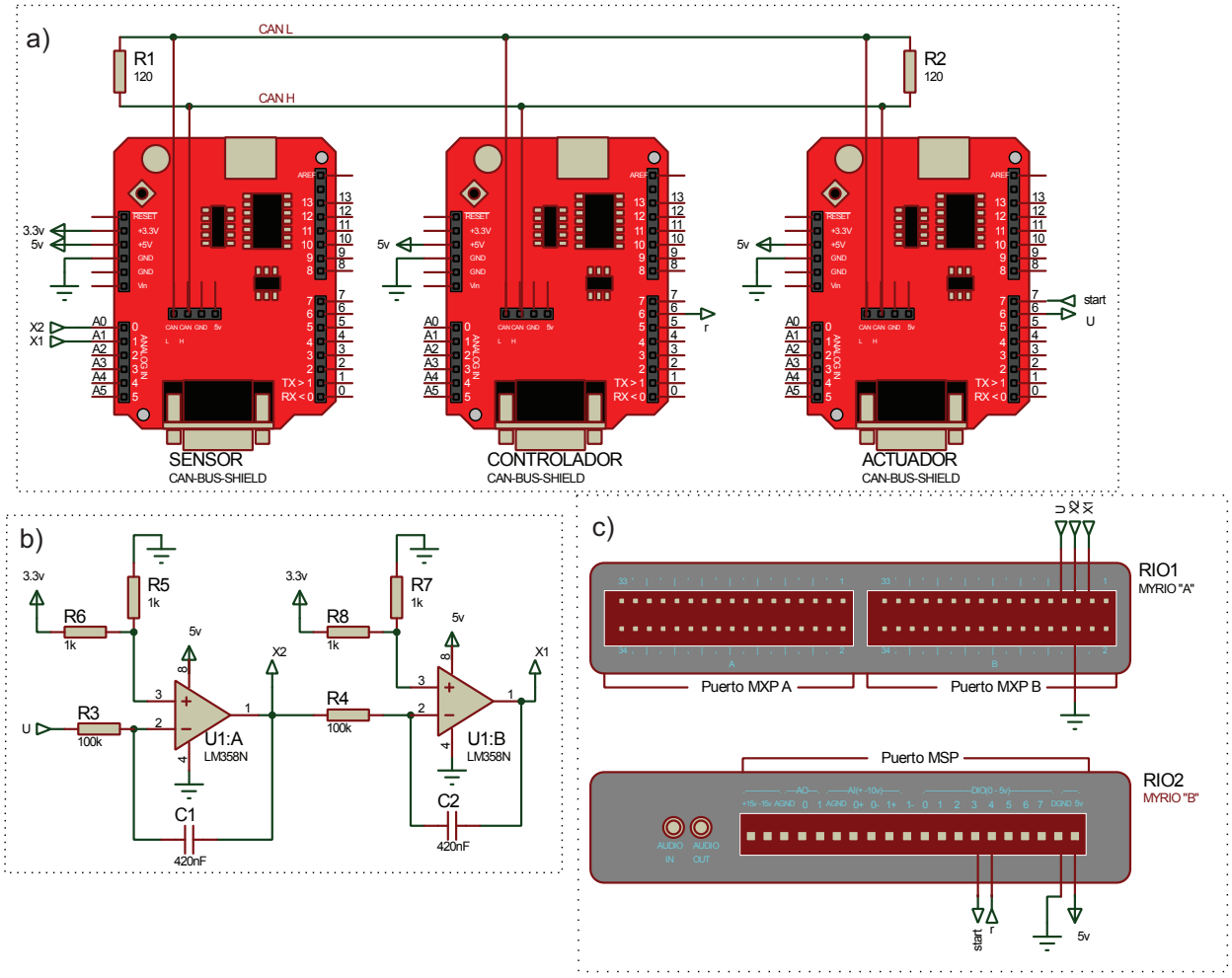


Figura 4.1: Diagrama de conección del sistema a implementarse, a) red CAN, b) planta y c) myRIO.

4.1.1. Planta

Como planta experimental se utiliza un circuito electrónico de doble integración (críticamente estable). La representación en el espacio de estados es

$$A = \begin{bmatrix} 0 & -23{,}81 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ -23{,}81 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{4.1}$$

Como se detalla en el algoritmo 4, el valor τ_k es usado para la estimación de estados $\hat{x}$ en el instante de actuación de la planta τ_k, el cual tiende a variar en cada lazo de control. Es decir, que la planta 4.1 se debe discretizar con los diferentes τ_k posibles, para lo cual se usaran funciones polinomiales para inferir los valores e imitar su comportamiento en el microprocesador. Se llamará al intervalo de los τ_k posibles $T \in \mathbb{R}^{1\times s}$ dentro de un intervalo cerrado $[\tau_{k\ min}, \tau_{k\ max}]$ como

$$T = \{\tau_{k\ min}, \tau_{k\ min} + \tau_g, \tau_{k\ min} + 2\tau_g, ...\tau_{k\ max}\}. \tag{4.2}$$

donde τ_g es la granularidad de τ_k definida como la unidad de menor incremento para los valores de estimación de estado y s el largo de T.

El resultado obtenido de la discretización de las matrices A y B, y definiendo los valores del conjunto T como $\tau_k\ min = 1ms$, $\tau_k\ max = 20ms$ y $\tau_g = 1$ para la discretización del sistema se obtiene

$$\Phi(\tau_k) = \begin{bmatrix} \Phi_{11}(\tau_k) & \Phi_{12}(\tau_k) \\ \Phi_{21}(\tau_k) & \Phi_{22}(\tau_k) \end{bmatrix} \tag{4.3}$$

$$\Gamma(\tau_k) = \begin{bmatrix} \Gamma_{11}(\tau_k) \\ \Gamma_{21}(\tau_k) \end{bmatrix}, \tag{4.4}$$

donde el elemento $\Phi_{12}(\tau_k)$ varia su valor, mientras que los elementos restantes de $\Phi(\tau_k)$ permanecen constantes, además, en la matriz $\Gamma(\tau_k)$ ambos elementos varian su valor, pero el elemento $\Gamma_{21}(\tau_k)$ es igual al elemento $\Phi_{12}(\tau_k)$. En la figura 4.2, la línea continua representa la curva ajustada del elemento $\Phi_{12}(\tau_k)$ y $\Gamma_{21}(\tau_k)$, mientras que los valores $\Phi_{12}(\tau_k)$ y $\Gamma_{21}(\tau_k)$ obtenidos con

T se representa mediante círculos.

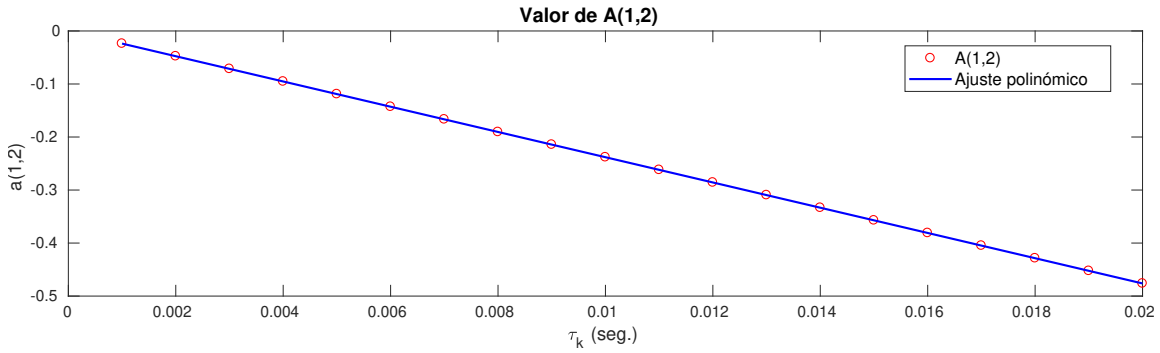


Figura 4.2: Ajuste polinómico del elemento $\Phi_{12}(\tau_k)$ y $\Gamma_{21}(\tau_k)$

En la figura 4.3, la línea continua representa la curva ajustada del elemento $\Gamma_{11}(\tau_k)$, y mediante círculos se representan los valores $\Gamma_{11}(\tau_k)$ obtenidos con T.

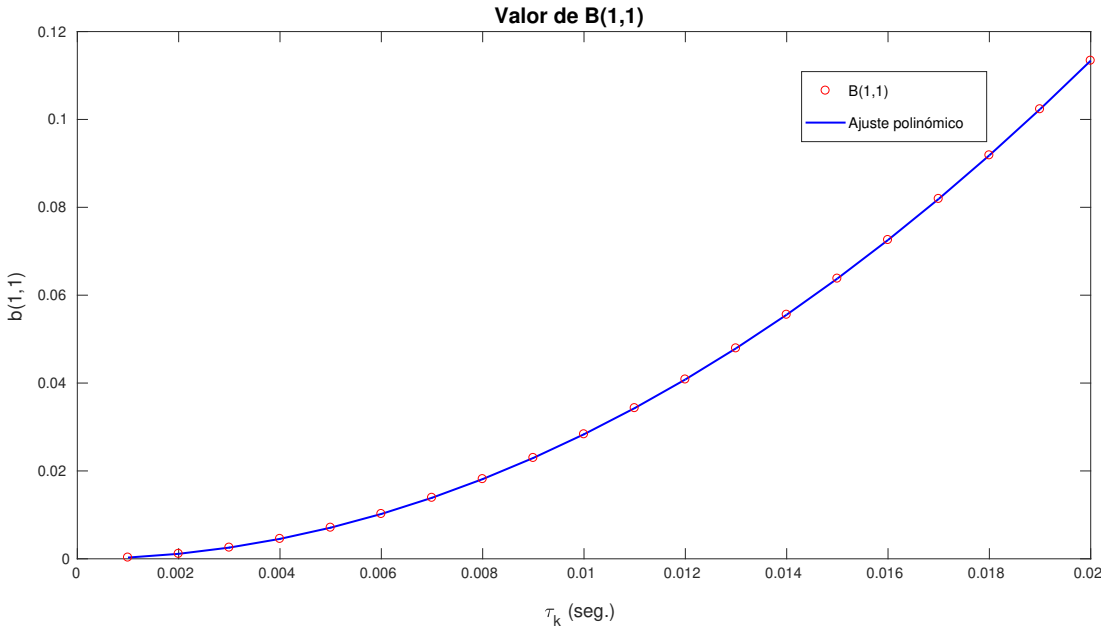


Figura 4.3: Ajuste polinómico del elemento $\Gamma_{11}(\tau_k)$

A continuación se resumen los resultados numéricos:

- El elemento $\Phi_{12}(\tau_k)$ y $\Gamma_{21}(\tau_k)$ están formados del siguiente polinomio:

$\Phi_{12}(\tau_k) = -23{,}8095\tau_k$.

- El elemento $\Gamma_{11}(\tau_k)$ se forma del siguiente polinomio:
  $\Gamma_{11}(\tau_k) = 283{,}4461(\tau_k)^2$.

Como resultado de la discretización de las matrices A y B, y remplazando los elementos variables por su respectivo polinomio tenemos

$$\Phi(\tau_k) = \begin{bmatrix} 1 & -23{,}8095\tau_k \\ 0 & 1 \end{bmatrix}, \quad \Gamma(\tau_k) = \begin{bmatrix} 283{,}4461(\tau_k)^2 \\ -23{,}8095\tau_k \end{bmatrix}. \tag{4.5}$$

4.1.2. Controlador

Como no es posible realizar una realimentacion exacta del estado, pues este no se conoce, la ley de control $u_k = Lx_k$ se reemplazara por una realimentación del estado estimado y teniendo en cuenta la ley de realimentación se obtiene que

$$u_k = -K(\tau_k)\hat{x}, \tag{4.6}$$

donde $K(\tau_k)$ es calculado en cada ejecución del controlador considerando como tiempo de muestreo los valores τ_k de 4.2.

Con la ayuda de el comando *lqrd* en Matlab se determina la matriz $K(\tau_k)$ del vector de control óptimo, el cual hace el uso de las matrices 4.1 y se establece una matriz de peso Q = [2 0; 0 0.5] y un parámetro de optimización R=0.01, y T especifica los tiempos de muestreo del regulador discreto.

Por lo tanto, se obtiene un total de s matrices de ganancias de control del tipo $K(\tau_k) \in \mathbb{R}^{m\times n}$ ya que son evaluadas para cada uno de los posibles s valores de τ_k dentro del conjunto T. Estas matrices tienen la forma

$$K\tau_k = lqrd(A,B,Q,R,T(\tau_k)) = \begin{bmatrix} K_{11}(\tau_k) & \cdots & K_{1n}(\tau_k) \\ \vdots & \ddots & \vdots \\ K_{m1}(\tau_k) & \cdots & K_{mn}(\tau_k) \end{bmatrix} \tag{4.7}$$

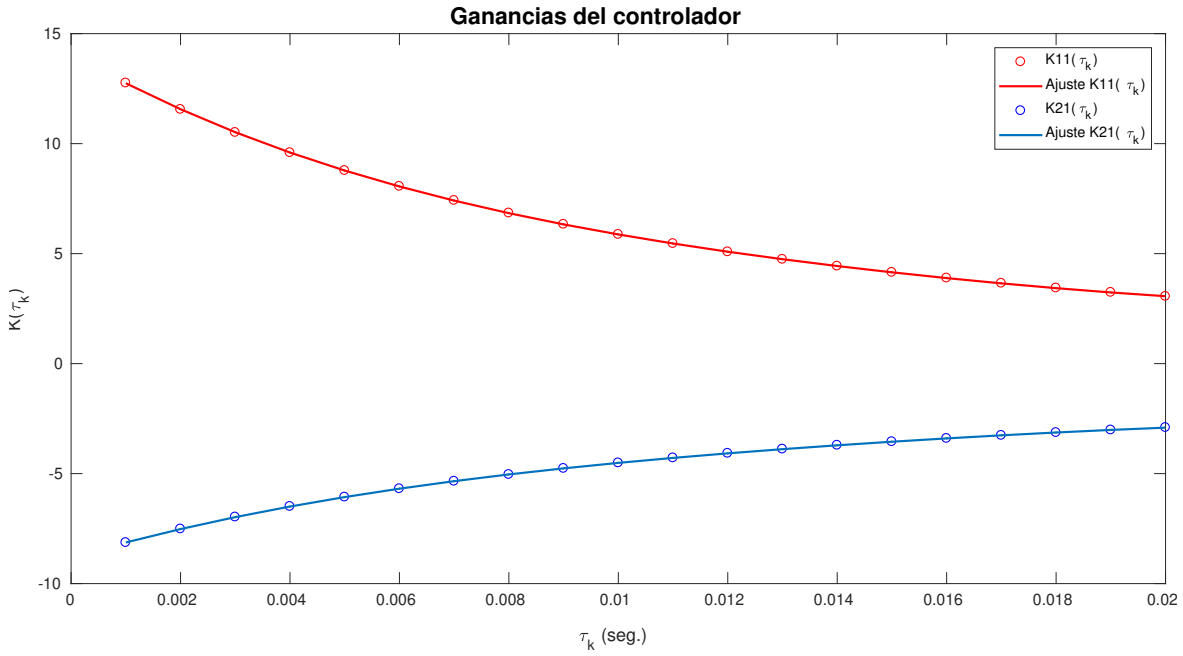


Figura 4.4: Ajuste polinómico de las ganancias $K_{11}(\tau_k)$ y $K_{21}(\tau_k)$

teniendo un total de $m \times n$ polinomios, cada uno siguiendo la forma

$$K_{ij}(\tau_k) = \alpha_1^{(ij)}\tau_k^v + \alpha_2^{(ij)}\tau_k^{v-1} + ...\alpha_v^{(ij)}\tau_k + \alpha_{v+1}^{(ij)}, \tag{4.8}$$

donde el subíndice (ij) indica la pertenencia de los coeficientes α al polinomio $K_{ij}(\tau_k)$; las filas i y las columnas j denotan la posición de los polinomios en la matriz de ganancias. En la Figura 4.4 las curvas ajustadas (líneas continuas) describen aproximadamente el comportamiento de las ganancias y las ganancias del controlador evaluadas para el conjunto T se muestran mediante círculos. Como resultado tenemos que la matriz de ganancias del controlador está formada de dos polinomios

$$K(\tau_k) = [K_{11}(\tau_k), K_{12}(\tau_k)], \tag{4.9}$$

donde el elemento $K_{11}(\tau_k)$ se forma del siguiente polinomio:

$K_{11}(\tau_k) = 44399448{,}08\tau_k^4 - 2949673{,}35\tau_k^3 + 84402{,}9\tau_k^2 - 1414{,}04\tau_k + 14{,}0780$

y el elemento $K_{12}(\tau_k)$ se forma del siguiente polinomio:

$$K_{12}(\tau_k) = -22103894{,}7\tau_k^4 + 1465116{,}84\tau_k^3 - 41985{,}38\tau_k^2 + 725{,}74\tau_k - 8{,}8157.$$

4.2. Simulación

La simulación se llevo a cabo con el simulador *TrueTime*, como se muestra en la Figura 4.5; los tres nodos (sensor, controlador y actuador) se implementaron usando un kernel de TrueTime, los cuales son conectados a un bloque de red CAN de TrueTime configurado a una velocidad de transmisión de 125 Kbps.

En cada uno de los nodos de la red se implementan los pseudocódigos presentados en el algoritmo 4. Además, en el nodo actuador se crea la única tarea periódica, la cual se activa cada 12ms marcando el inicio de cada instante de actuación t_k, mientras que el resto de nodos se disparan por mensajes. En el nodo controlador se realiza la discretización de la planta con τ_k y se implementa un regulador lineal cuadrático (LQR) para determinar la matriz de ganancias K. En cuanto a la sincronización de reloj en la simulación no se realiza, ya que todos los nodos trabajan con el reloj interno del computador y la sincronización en los instantes de actuación esta dada por la tarea periódica creada en el actuador.

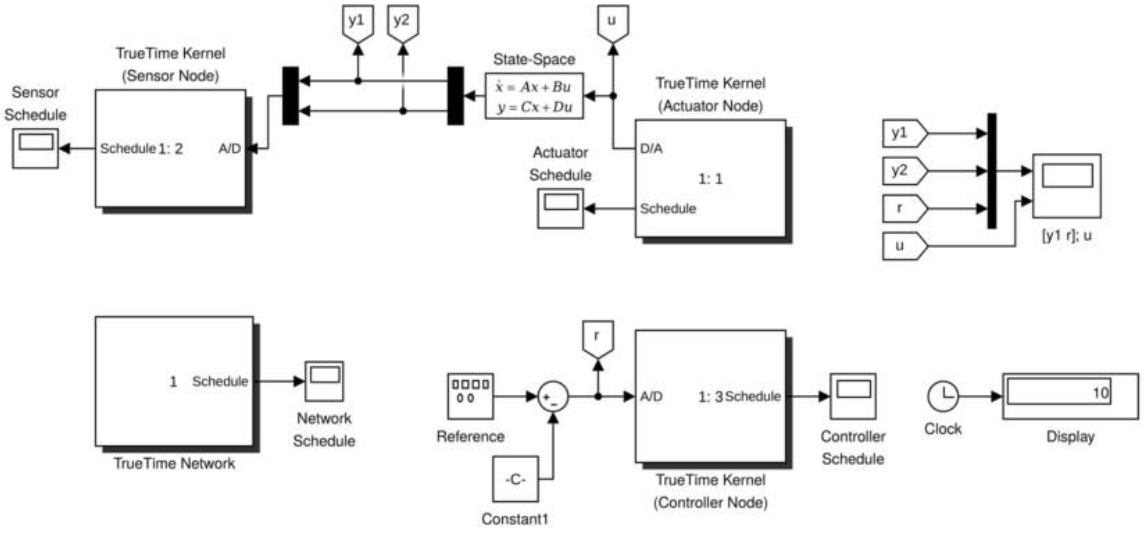


Figura 4.5: Simulación en *TrueTime*

4.3. Implementación sobre un microprocesador

La Figura 4.6 comprende los tres nodos necesarios para llevar a cabo la implementación: a) el nodo sensor, b) el nodo actuador y c) el nodo controlador. Cada nodo de la red ejecuta internamente el algoritmo 4. Además, la shield CAN-BUS se configura a una velocidad de transmisión de 125 Kbps. Para conocer más acerca del ambiente de la shield CAN-BUS se recomienda revisar [22]. Nótese en la Figura 4.6: d) el sistema doble integrador (planta), al que se realiza la acción de control. Como detalla el algoritmo 4, en el nodo actuador se configura una interrupción cada h que es igual a 12ms, en la que se genera el siguiente instante de actuación t_k que es enviado al nodo sensor. Además de ejecutar el nuevo instante de actuación, también aplica la señal u_k a la planta mediante modulación de ancho de pulso (PWM). Posteriormente, el nodo sensor tras recibir t_k muestrea la planta, para luego calcular el valor τ_k y enviar estos valores al controlador. En el nodo controlador, en lugar de discretizar la planta y de calcular las ganancias de K con τ_k, se usan los polinomios presentados en 4.5 y 4.9 con los que se estima el estado de la planta para el instante t_k y calcula la señal de control para enviarla finalmente al nodo actuador.

En la Figura 4.7 se presenta todo el sistema necesario para realizar la implementación, nótese el dispositivo myRIO y adicional una laptop, la cual corre el software LabVIEW que permite presentar las gráficas de la supervición de la planta.

La sincronización de reloj en el sistema se lleva a cabo usando el protocolo IEEE 1588 PTP, usando como reloj maestro al nodo actuador, donde se configura una interrupción cada dos segundos en la cual se envía los mensajes de sincronización al resto de nodos. Cabe mencionar que la acción de sincronización es muy independiente de la acción de lazo de control.

En la Figura 4.8 se presenta los datos de la sincronización de reloj de los diferentes nodos a) actuador, b) sensor y c) controlador, donde la primera columna de las figuras detalla la hora, la segunda columna el tiempo al que se recibe el mensaje del puerto serial y la tercera columna los datos del puerto serial; en esta última columna de las tres figuras se observa que los relojes

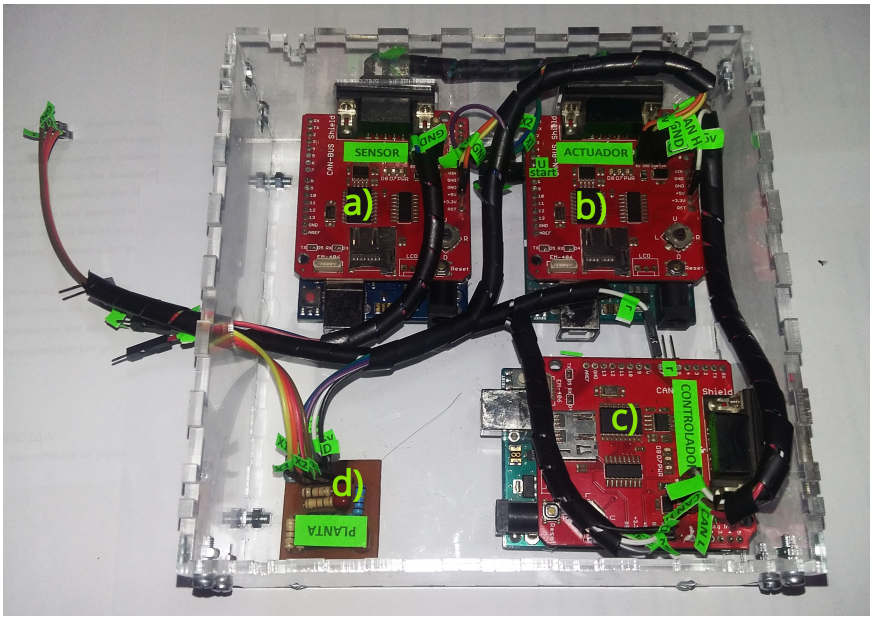


Figura 4.6: Implementación real de los algoritmos, a) sensor, b) actuador, c) controlador y d) planta.

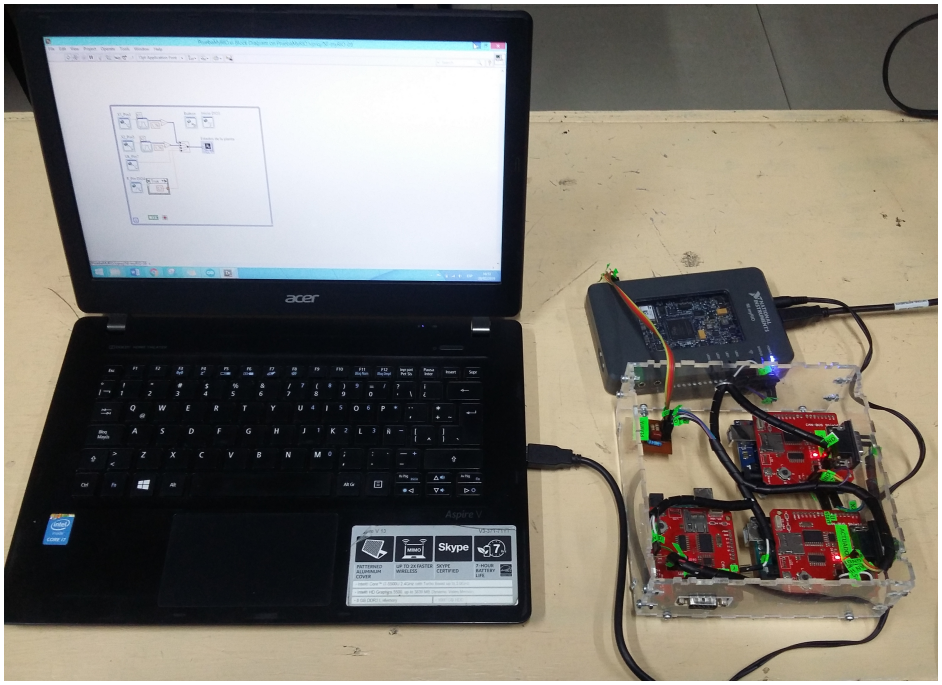


Figura 4.7: Sistema para realizar las pruebas de implementación.

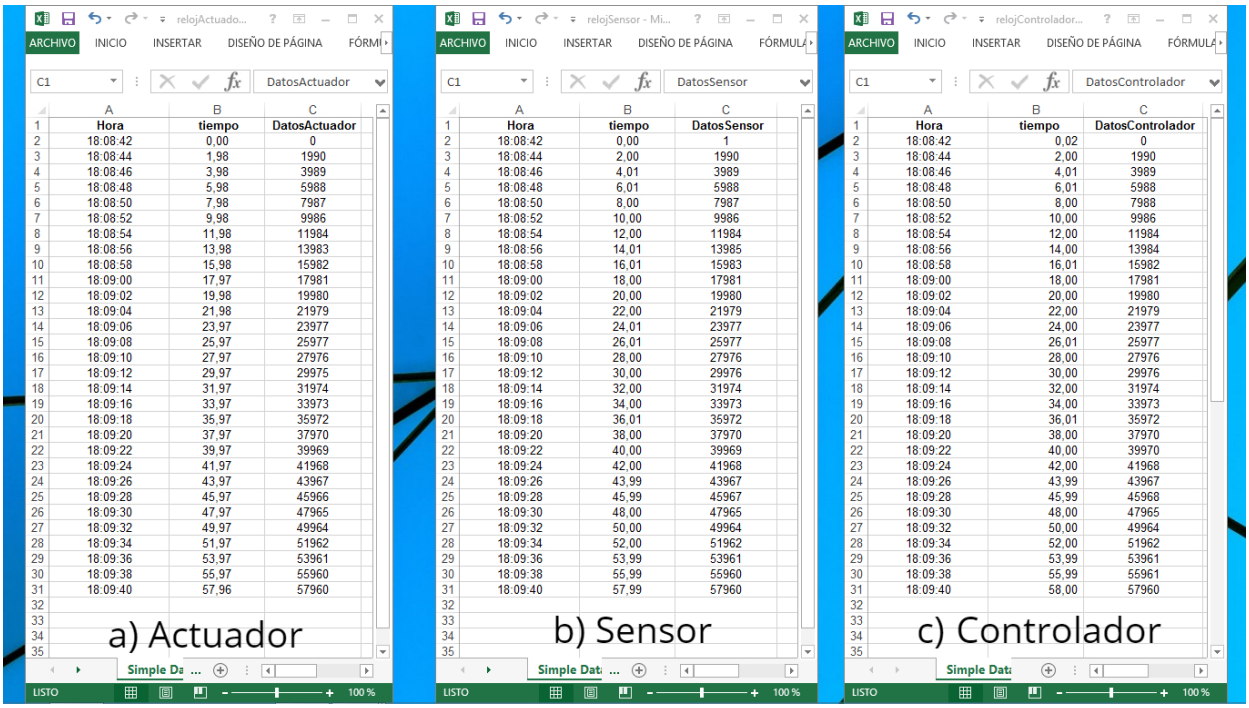


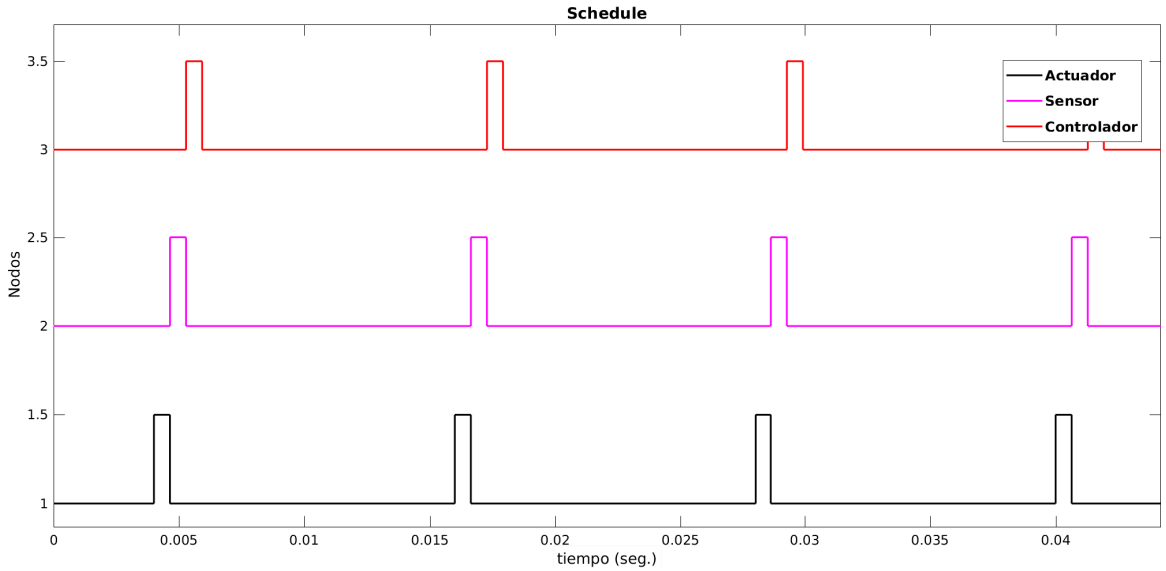
Figura 4.8: Resultados de la sincronización de reloj: nodo a) actuador, b) sensor y c) controlador.

del sensor y controlador estan sincronizados con el actuador, teniendo un error de hasta 1ms, tal como lo detalla [20]. Los datos de los diferentes nodos se los obtuvo con el software plxDAQ, para mas información sobre este ambiente referirse a [23].

4.4. Análisis de resultados

La Figura 4.9 muestra la activación de los nodos tanto en: a) la simulación, como en b) la planta real; en la implementación real el nodo actuador se ejecuta cada 12 ms aproximadamente ya que en ocasiones oscila entre valores muy pequeños, esto debido a incertidumbre. Tras haberse iniciado el lazo de control en el actuador, se procede con el nodo sensor, el cual tiende a varíar su ejecución en cada lazo como se ha predicho, lo cual permite tener un muestreo asíncrono de la planta, es decir que varia su valor τ_k en cada lazo de control como se observa en la Figura 4.9.b).

a) Simulación



b) Implementación real

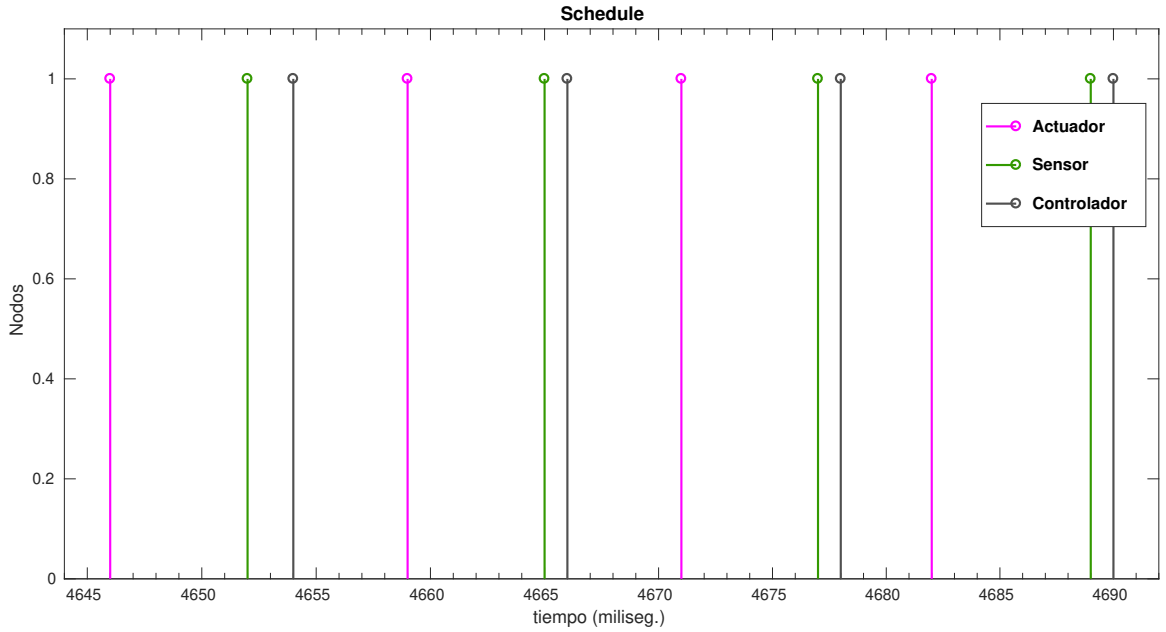


Figura 4.9: Evolución de la activación de los nodos: a) simulación (arriba), b) implementación real (abajo)

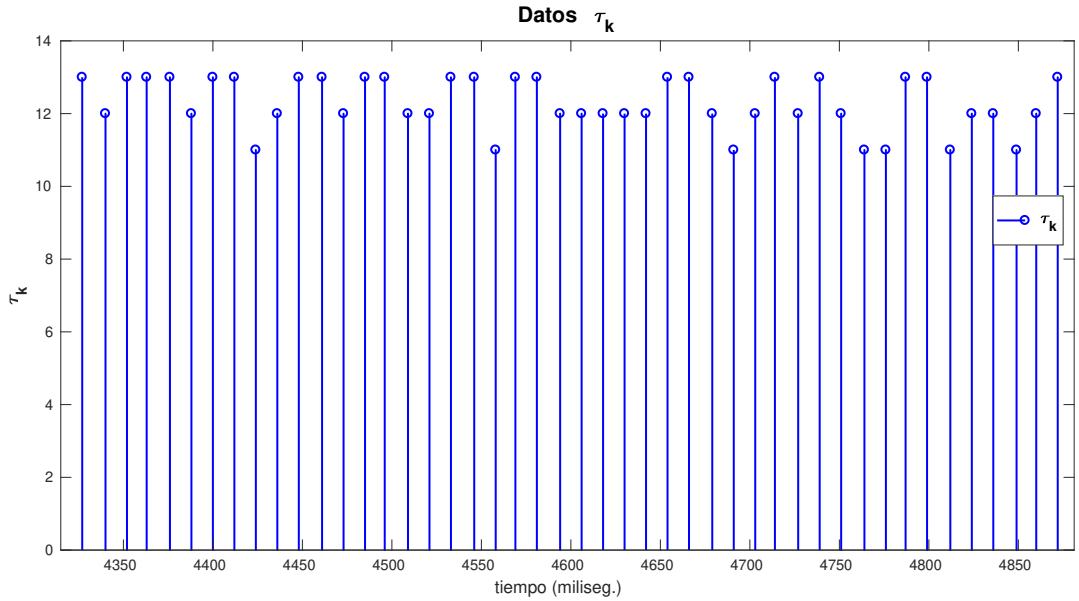
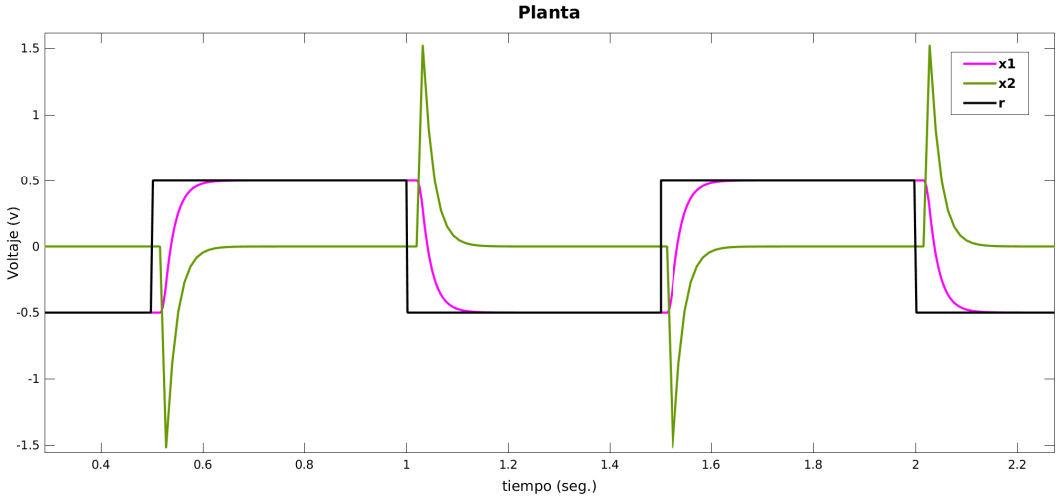


Figura 4.10: Valores de τ_k en la planta real

Al diseñarse la planta con un periodo de actuación de 12 ms, el valor de τ_k debe mantenerse en este valor; como se nota en la figura 4.10 el valor de τ_k oscila alrededor de este valor, determinando que los valores se encuentran dentro del intérvalo $T = [1ms : 20ms]$, dando a notar que el tiempo de ejecución del nodo sensor es irregular.

Por otro lado en la Figura 4.11, se tiene el comportamiento de la planta al aplicarse el algoritmo de control tanto en a) la simulación como en b) la planta real. En la planta real se aprecia que el estado $x1$ tiende a seguir a la referencia r, pero no se mantiene en un estado estable por lo cual se origina un oscilamiento. Este comportamiento es debido a que los estados medidos de la planta tienen interferencia y a la incertidumbre en la sincronización de reloj.

a) Simulación



b) Implementación real

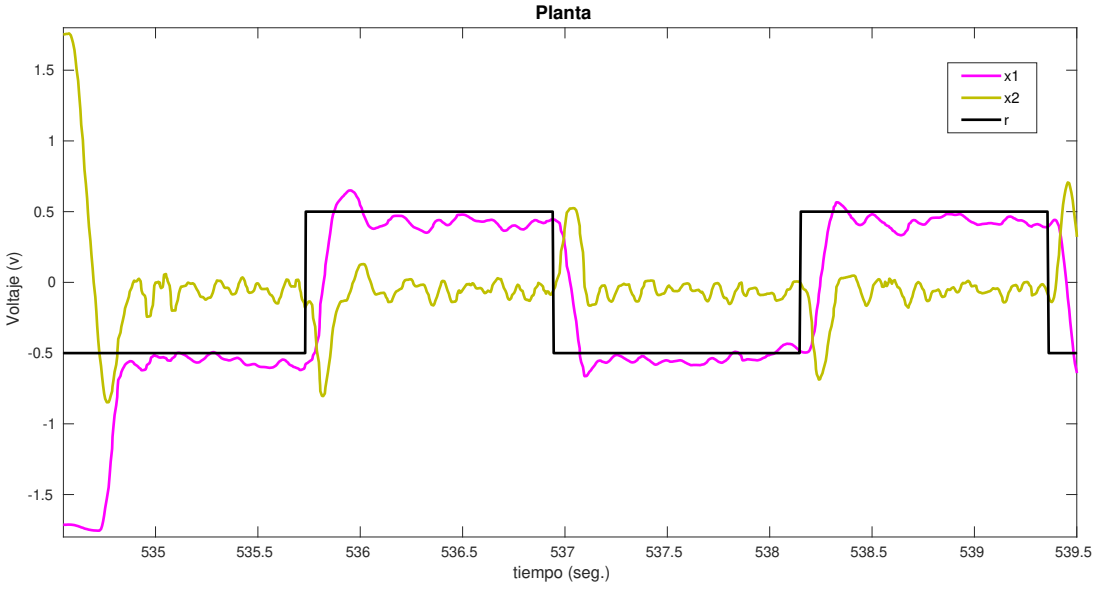


Figura 4.11: Evolución de los estados de la planta aplicando el algoritmo de control: a) simulación (arriba), b) implementación real (abajo)

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

- Se demostro que la aplicación de este algoritmo de control puede controlar una planta basando su funcionamiento en solo un punto de sincronización, en este caso en el instante de actuación. Para ello, se implemento el algoritmo sobre microcontroladores de bajo costo y que manejan una velocidad de reloj baja.
- Se remplazo el uso del problema LQRD y de discretización de la planta por polinomios que asemejan su funcionamiento en el nodo controlador evitando asi el cálculo fuera de linea de estos parámetros.
- Se corroboró que la teoría del algoritmo de control llega a cumplirse al realizarse su aplicación en una red y planta real.

5.2. Trabajo futuro

Por medio de simulaciones y de la aplicación práctica sobre un microprocesador real verifican la efectividad del algoritmo de control sobre muestreos con tiempo irregulares, lo cual abre un campo de investigación para mejorar el desempeño de la planta, en relación a implementar

técnicas de observación para eliminar los errores que presenta la planta.

Bibliografía

[1] Y. Tipsuwan and M. Y. Chow, Control methodologies in networked control systems, Control Engineering Practice, Vol. 11, pp. 1099-1111, 2003.

[2] B. Wittenmark, J. Nilsson and M. Törngren, "Timing Problems in Real-Time Control Systems," 1995 American Control Conference, 1995.

[3] J. Baillieul and P.J. Antsaklis, Control and Communication Challenges in Networked Real-Time Systems,, in Proceedings of the IEEE, January, 2007, 95:1, pp. 9-28.

[4] J. Hespanha, P. Naghshtabrizi, Y. Xu, A Survey of Recent Results in Networked Control Systems, Proc. of IEEE Special Issue on Technology of Networked Control Systems, 95(1):138162, Jan. 2007.

[5] IEEE 1588^{TM} -2002. Standard for A Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.

[6] ISO 11898-1: Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling, 2003

[7] P. Ogren, E. Fiorelli, N. E. Leonard, «Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment,» IEEE Trans. Automat. Contr., vol. 49, n^{o} 8, pp. 1292-1302, 2004.

[8] C. Meng, T. Wang, W. Chou, S. Luan,Y. Zhang, Z. Tian, «Remote surgery case: Robot-assisted teleneurosurgery,» IEEE Int. Conf. Robot. and Auto., vol. 1, pp. 819-823, 2004.

[9] Pete Seiler, Raja Sengupta, «An H∞ Approach to Networked Control,» IEEE Trans. On Aut. Control, vol. 50, nº 3, pp. 356-364, 2003.

[10] P. Martí, J.M. Fuertes and G. Fohler, «An Integrated Approach to Realtime Distributed Control Systems Over Fieldbuses.» 8th IEEE International Conference on Emerging Technologies and Factory Automation, Antibes Juan-les-pins, France, October 2001.

[11] C. Lozoya, M. Velasco, and P. Martí, "The One-Shot Task Model for Robust Real-Time Embedded Control Systems," IEEE Transactions on Industrial Informatics, vol.4, no.3, pp.164-174, Aug. 2008.

[12] P. Martí, A. Camacho, M. Velasco, P. Marés, J. Fuertes, «Synchronizing Sampling and Actuation in the Absence of Global Time in Networked Control Systems,» Emerging Technologies and Factory Automation (ETFA), IEEE Conference, Bilbao, Spain, Sept. 2010.

[13] P. Martí, M. Velasco, J. Yépez, X. Martn, «Lowering Traffic without Sacrificing Performance in Networked Control Systems,» Emerging Technologies and Factory Automation (ETFA),IEEE 16th Conference, Toulouse, France, Sept. 2011.

[14] K.J. Åström and B. Wittenmark, Computer-Controlled Systems, third ed, Prentice-Hall, 1997

[15] W. Hu, G. Liu, and D. Rees, "Event-driven networked predictive control," IEEE Transactions on Industrial Electronics, vol. 54, n.3, 1603-1613, 2007.

[16] W. S. Hu, G. P. Liu, and D. Rees, "Design and implementation of networked predictive control systems based on round trip time delay measurement," in Proc. Amer. Control Conf., Minneapolis, MN, Jun. 2006,pp. 674-679.

[17] W. S. Hu, G. P. Liu, and D. Rees, "Design and implementation of networked predictive control systems based on round trip time delay measurement," in Proc. Amer. Control Conf., Minneapolis, MN, Jun. 2006, pp. 674–679.

[18] L. Ljung, System Identification: Theory for the User. Englewood Cliffs, NJ: Prentice-Hall, 1987, pp. 305–311.

[19] P. Martí, M. Velasco, ”Toward flexible scheduling of real-time control tasks: Reviewing basic control models”, Proc. 10th Int. Conf. Hybrid Systems Computation and Control, 2007.

[20] P. Martí, M. Velasco, C. Lozoya, and J.M. Fuertes, “Clock Synchronization for Networked Control Systems Using Low-Cost Microcontrollers”, Research report ESAII-RR-08-02, Automatic Control Dept., Technical University of Catalonia, April 2008.

[21] M. Velasco, P. Martí, R. Castañé, J. Guardia and J. M. Fuertes, “A CAN Application Profile for Control Optimization in Networked Control Systems”, 32th Anual Conference of the IEE Industrial Electronics Society, Nov. 2006.

[22] SparkFun Electronics, 2018, [Online], Available: https://www.sparkfun.com/products/retired/10039. [Accedido: Dic-2018]

[23] Parallax Inc., 2018, [Online], Available: https://www.parallax.com/downloads/plx-daq. [Accedido: Dic-2018]

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 y parámetros de simulación (runThisFileFirst.m)

Programa 1: Diseño del controlador y parámetros para la simulación

```
%%CONTROLLER DESIGN

clear all, close all, clc

global A B C D
h = 12e-3;        % sampling period

A = [0 -23.8095; 0 0];   % matrix A
B = [0; -23.8095];       % matrix B
C = [1 0];               % matrix C
D = [0];

sys = ss(A, B, C, D)     % Space states
sys_d = c2d(sys, h)

%%Optimal design
Q = [2 0; 0 0.5];
R = 0.01;
[K,S,e] = lqrd(A,B,Q,R,h);

%%SIMULATION IN FRONT OF GIVEN CONDITIONS

tTop = 10;          % maximum time for simulation
%tick = 0.0001;     % granularity

x = [1; -1];        % given initial conditions
```

```
history_x = [];                    % buffer initialization
%history = [history sys_d.c*x];
history_x = [history_x x];
history_u = [];
history_u = [history_u 0];

for i = 0 : h : tTop
    u = -K*x;                      % applies control
    x = sys_d.a*x+sys_d.b*u;
    y = sys_d.c*x;
    history_x = [history_x x];     % updates buffer
    %history_y = [history y];      % updates buffer
    history_u = [history_u u];
end

time = 0:h:tTop+h;

figure
plot(time, history_x(1,:), time, history_x(2,:), time, history_u(:));
xlabel('Time [s]');                % plot figures
ylabel('x, u');
legend('x[1]','x[2]','u');
```

.B. Firmware

.B.1. Código implementado sobre microcontrolador (nodo1Working.ino)

Programa 2: Código implementado sobre el nodo Sensor

```
#include "mcp_can.h"
#include <SPI.h>

const int SPI_CS_PIN = 10; // CS pin arduino
unsigned char stmp[5] = {0, 0, 0, 0, 0}; // vector to send data over CAN
unsigned char len = 0; // length of data
unsigned char buf[5] = { 0, 0, 0, 0, 0}; // buffer to save data of CAN
```

```
int x1; // states
byte *ptr_x1 = (byte*)&x1;
int x2;
byte *ptr_x2 = (byte*)&x2;

unsigned long tauk = 0;
unsigned long timeStamp = 0;
unsigned long tk = 0;
byte *ptr_tk = (byte*)&tk;

unsigned long TS1 = 0; // 'offset' of clock synchronization
unsigned long T_S1 = 0;
unsigned long TM1 = 0;
byte *ptr_TM1 = (byte*)&TM1;
unsigned long sysTime = 0;
long offset = 0;

char e[10]; // vectors to send data over serial port
char e1[10];
char e2[10];
char e3[10];
int marca = 1; // stamp of node activation
long reset_timer = 0; // reset timer of plx-DAQ

MCP_CAN CAN(SPI_CS_PIN); // set CS pin

void datos() // read states of the plant
{
  x1 = analogRead(A0);
  x2 = analogRead(A1);

  timeStamp = millis() - offset - 1;
  tauk = tk - timeStamp;

  for (int i = 1; i >= 0; i--)
    stmp[i] = ptr_x1[i];

  for (int i = 1; i >= 0; i--)
    stmp[2 + i] = ptr_x2[i];

  stmp[4] = tauk;
  CAN.sendMsgBuf(0x02, 0, 5, stmp); // sends data to the controller 0x02
```

```
}

void enviaSchedule() // sends schedule data over serial port
{
  Serial.print("DATA,TIME,");
  dtostrf(sysTime, 5 , 3 , e1); // changes data to char
  dtostrf(marca, 5 , 3 , e);
  Serial.print(e1);Serial.print(",");
  Serial.println(e);Serial.println();
}

void enviaSinc() // sends synchronization data over serial port
{
  if (reset_timer == 1)
    Serial.println("RESETTIMER"); // reset timer of plx-DAQ

  Serial.print("DATA,TIME,TIMER,");
  sysTime = millis() - offset - 1;
  dtostrf(sysTime, 5 , 3 , e2); // changes data to char
  Serial.print(e2);Serial.println();
}

void enviaTauk() // sends tauk data over serial port
{
  Serial.print("DATA,TIME,");
  sysTime = millis() - offset - 1;
  dtostrf(sysTime, 5 , 3 , e2); // changes data to char
  dtostrf(tauk, 5 , 3 , e3);
  Serial.print(e2);Serial.print(",");
  Serial.print(e3);Serial.println();
}

void setup()
{
  Serial.begin(38400);

  while (CAN_OK != CAN.begin(CAN_125KBPS)) // init can bus : baudrate = 125k
  {
    Serial.println("CAN BUS Shield init fail");
    Serial.println(" Init CAN BUS Shield again");
  }
  Serial.println("CAN BUS Shield init ok! 'Sensor'");
  Serial.println("CLEARDATA"); // clear previous data of the plx-DAQ
```

```
  Serial.println("LABEL, Hora, tiempo, Datos_S"); // names the columns' labels of the plx-DAQ
}

void loop()
{
  if (CAN_MSGAVAIL == CAN.checkReceive()) // checks incoming data
  {
    T_S1 = millis(); // message arrival time
    CAN.readMsgBuf(&len, buf); // save data in the buffer

    if (CAN.getCanId() == 0x01) // start to read and send the states
    {
      //sysTime = millis()-offset-1; // time stamp to send in the enviaSchedule loop to the
          plx-DAQ
      for (int i = 3; i >= 0; i--)
        ptr_tk[i] = buf[i];
      datos();
      //enviaTauk(); // send tauk data to the plx-DAQ
      //enviaSchedule(); // send schedule data to the plx-DAQ
    }

    if (CAN.getCanId() == 0x00) // clock synchronization of the first message 0x00
    {
      if (buf[4] == 0) {
        for (int i = 3; i >= 0; i--)
          ptr_TM1[i] = buf[i];
        TS1 = T_S1;
      }

      if (buf[4] == 1) // clock synchronization of the second message 0x00
      {
        for (int i = 3; i >= 0; i--)
          ptr_TM1[i] = buf[i];
        offset = TS1 - TM1;
        //reset_timer += 1; // timer's stamp for the plx-DAQ
        //enviaSinc();  // send sincronization data to plx-DAQ
      }
    }
  }
}
```

.B.2. Código implementado sobre microcontrolador (nodo2Working.ino)

Programa 3: Código implementado sobre el nodo Controlador

```
#include <SPI.h>
#include "mcp_can.h"

const int SPI_CS_PIN = 10; // CS pin arduino
unsigned char len = 0; // length of data
unsigned char buf[5] = {0, 0, 0, 0, 0}; // buffer to save data of CAN
unsigned char stmp[5] = {0, 0, 0, 0, 0}; // vector to send data over CAN

float K1 = 0.0; // gains
float K2 = 0.0;
float tauk = 0.0;
const float v_max = 3.45;
int pin_r = 6; // reference output pin

#define refPos  0.5 // positive an negative values for the reference
#define refNeg  -0.5

int topCount = 100; // period to change the reference (actuationPeriod = 0.012)
unsigned int count = 0; // T_ref = 2 * topCount*actuationPeriod

unsigned long sysTime = 0;

float r = refNeg; // reference
unsigned char *ptr_r = (unsigned char*)&r;

int x1 = 0; // state x1 at the output of the second integrator
unsigned char *ptr_x1 = (unsigned char*)&x1;

int x2 = 0; // state x2 at the output of the first integrator
unsigned char *ptr_x2 = (unsigned char*)&x2;

float x11 = 0.0; // state x1
unsigned char *ptr_x11 = (unsigned char*)&x11;

float x22 = 0.0; // state x2
unsigned char *ptr_x22 = (unsigned char*)&x22;
```

```
int uk = 0; // control
float uk1 = 0.0; // uk-1
unsigned char *ptr_uk1 = (unsigned char*)&uk1;

float x_hat_11 = 0.0;
float x_hat_21 = 0.0;

float B_11 = 0.03259; // matrix B
float B_21 = -0.2553;

float A_11 = 1.0; // matrix A
float A_12 = -0.2553;
float A_21 = 0.0;
float A_22 = 1.0;

unsigned long TS1 = 0; // 'offset' of clock synchronization
unsigned long T_S1 = 0;
unsigned long TM1 = 0;
byte *ptr_TM1 = (byte*)&TM1;
long offset = 0;

char e[10]; // vectors to send data over serial port
char e1[10];
char e2[10];
int marca = 1; // stamp of node activation
long reset_timer = 0; // reset timer of plx-DAQ

MCP_CAN CAN(SPI_CS_PIN); // set CS pin

void calculateControl()
{
  for (int i = 1; i >= 0; i--)
    ptr_x1[i] = buf[i];

  for (int i = 1; i >= 0; i--)
    ptr_x2[i] = buf[2 + i];

  tauk = (float)buf[4] / 1000;

  x11 = ((float)x1 * 5 / 1024) - v_max / 2 ;
  x22 = ((float)x2 * 5 / 1024) - v_max / 2 ;

  A_12 = -23.8095 * tauk; // polynomials for matrix elements
```

```
  B_11 = 283.4461 * (tauk * tauk);
  B_21 = A_12;

  // polynomials for controller gains
  K1 = 44399448.08 * (tauk * tauk * tauk * tauk) - 2949673.35 * (tauk * tauk * tauk) + 84402.9
       * (tauk * tauk) - 1414.04 * tauk + 14.0780;
  K2 = -22103894.7 * (tauk * tauk * tauk * tauk) + 1465116.84 * (tauk * tauk * tauk) -
       41985.38 * (tauk * tauk) + 725.74 * tauk - 8.8157;

  x_hat_11 = ((A_11 * x11) + (A_12 * x22)) + (B_11 * uk1);
  x_hat_21 = ((A_21 * x11) + (A_22 * x22)) + (B_21 * uk1);

  uk1 = (-K1 * (x_hat_11 - r)) - (K2 * x_hat_21);

  if (uk1 > 1.65)
      uk1 = 1.65;

  if (uk1 < -1.65)
      uk1 = -1.65;

  uk = uk1;
  uk = (uk / v_max) * 168 + 84;
  uk = (int)uk;

  stmp[0] = uk;
  CAN.sendMsgBuf(0x03, 0, 1, stmp); // sends control action to the actuator 0x03
}

void ref()
{
  count++;

  if (count >= topCount) // change reference
  {
    digitalWrite(pin_r, digitalRead(pin_r) ^ 1);
    count = 0;

    if (r == refPos)
    {
      r = refNeg;
    }
    else if (r == refNeg)
    {
```

```
      r = refPos;
    }
  }
}

void enviaSchedule() // sends schedule data over serial port
{
  Serial.print("DATA,TIME,");
  dtostrf(sysTime, 5 , 3 , e1); // changes data to char
  dtostrf(marca, 5 , 3 , e);
  Serial.print(e1);Serial.print(",");
  Serial.println(e);Serial.println();
}

void enviaSinc() // sends synchronization data over serial port
{
  if (reset_timer == 1)
    Serial.println("RESETTIMER"); // reset timer of plx-DAQ

  Serial.print("DATA,TIME,TIMER,");
  sysTime = millis() - offset - 1;
  dtostrf(sysTime, 5 , 3 , e2); // changes data to char
  Serial.print(e2);Serial.println();
}

void setup()
{
  Serial.begin(38400);
  pinMode(pin_r, OUTPUT); // digital output of reference

  while (CAN_OK != CAN.begin(CAN_125KBPS)) // init can bus : baudrate = 125k
  {
    Serial.println("CAN BUS Shield init fail");
    Serial.println(" Init CAN BUS Shield again");
    delay(100);
  }
  Serial.println("CAN BUS Shield init ok!'Controlador'");
  Serial.println("CLEARDATA"); // clear previous data of the plx-DAQ
  Serial.println("LABEL, Hora, tiempo, Datos_C"); // names the columns' labels of the plx-DAQ
}

void loop()
{
```

```
  if (CAN_MSGAVAIL == CAN.checkReceive()) // checks incoming data
  {
    T_S1 = millis(); // message arrival time
    CAN.readMsgBuf(&len, buf); // save data in the buffer

    if (CAN.getCanId() == 0x02) // start to calculate control
    {
      //sysTime = millis()-offset-1; // time stamp to send in the enviaSchedule loop to the
          plx-DAQ
      calculateControl();
      ref();
      //enviaSchedule(); // send schedule data to plx-DAQ
    }

    if (CAN.getCanId() == 0x00) // clock synchronization of the first message 0x00
    {
      if (buf[4] == 0) {
        for (int i = 3; i >= 0; i--)
          ptr_TM1[i] = buf[i];
        TS1 = T_S1;
      }

      if (buf[4] == 1) // clock synchronization of the second message 0x00
      {
        for (int i = 3; i >= 0; i--)
          ptr_TM1[i] = buf[i];
        offset = TS1 - TM1;
        //reset_timer += 1; // timer's stamp for the plx-DAQ
        //enviaSinc();  // send sincronization data to plx-DAQ
      }
    }
  }
}
```

.B.3. Código implementado sobre microcontrolador (nodo3Working.ino)

Programa 4: Código implementado sobre el nodo Actuador

```
#include <SPI.h>
#include "mcp_can.h"
#include <avr/io.h>
#include <avr/interrupt.h>

const int SPI_CS_PIN = 10; // CS pin arduino
const int pwm1 = 6; // pwm output
int h = 12; // actuation period
long tiempo_anterior = -h;

int uk = 0; // control action
unsigned char len = 0; // length of data
unsigned char buf[4] = {0, 0, 0, 0}; // buffer to save data of CAN
unsigned char stmp0[6] = {0, 0, 0, 0, 0, 0}; // vectors to send data over CAN
unsigned char stmp1[4] = {0, 0, 0, 0};

unsigned long tk = 0; // actuation instant
byte *ptr_tk = (byte*)&tk;

unsigned long message_time = 0; // time to synchronization
byte *ptr_message = (byte*)&message_time;

int start = 0; // system start stamp
int inicio = 7; // start button
int marca = 1; // stamp of node activation
char e[10]; // vectors to send data over serial port
char e1[2];
char e2[10];
long reset_timer = 0; // reset timer of plx-DAQ

MCP_CAN CAN(SPI_CS_PIN); // set CS pin

void enviaSchedule() // sends schedule data over serial port
{
  Serial.print("DATA,TIME,");
  dtostrf(message_time, 5 , 3 , e1); // changes data to char
  dtostrf(marca, 5 , 3 , e);
  Serial.print(e1); Serial.print(",");
  Serial.println(e); Serial.println();
}

void enviaSinc() // sends synchronization data over serial port
{
```

```
  if (reset_timer == 1)
    Serial.println("RESETTIMER"); // reset timer of plx-DAQ

  Serial.print("DATA,TIME,TIMER,");
  dtostrf(millis(), 5 , 3 , e2); // changes data to char
  Serial.print(e2); Serial.println();
}

void setup()
{
  cli(); // disable interrupts
  Serial.begin(38400);

  while (CAN_OK != CAN.begin(CAN_125KBPS)) // init can bus : baudrate = 125k
  {
    Serial.println("CAN BUS Shield init fail");
    Serial.println(" Init CAN BUS Shield again");
  }
  Serial.println("CAN BUS Shield init ok!'Actuador'");
  Serial.println("CLEARDATA"); // clear previous data of the plx-DAQ
  Serial.println("LABEL, Hora, tiempo, Datos_A"); // names the columns' labels of the plx-DAQ

  // int1 set to 2 sec
  // TCNT = 2^(res.timer)-((period [sec] * Tosc)/ preescale)
  // TCNT = 65535 - ((2 * 16 Mhz)/1024)
  TCCR1A = 0;
  TCCR1B = 0;
  TIMSK1 = (0 << OCIE1B) | (0 << OCIE1A) | (1 << TOIE1);
  TCCR1B |= (1 << CS10);
  TCCR1B |= (0 << CS11);
  TCCR1B |= (1 << CS12);
  TCNT1 = 34285;
  OCR1A = 0x00;
  OCR1B = 0x00;

  // int2 set to 12ms
  // TCNT = 2^(res.timer)-((period [sec] * Tosc)/ preescale)
  // TCNT = 255 - ((0.012 * 16 Mhz)/1024)
  TCCR2A = 0;
  TCCR2B = 0;
  TIMSK2 = (0 << OCIE2B) | (0 << OCIE2A) | (1 << TOIE2);
  TCCR2B |= (1 << CS20);
  TCCR2B |= (1 << CS21);
```

```
  TCCR2B |= (1 << CS22);
  TCNT2 = 67;
  OCR2A = 0x00;
  OCR2B = 0x00;
  sei(); // enable interruptions
}

ISR(TIMER1_OVF_vect) // starts to synchronization
{
    TCNT1 = 34285; // timer value
    message_time = millis(); // time to synchronize
    for (int i = 3; i >= 0; i--)
      stmp0[i] = ptr_message[i];
    stmp0[4] = 0;
    CAN.sendMsgBuf(0x00, 0, 5, stmp0); // sends the first synchronization mesagge  0x00

    message_time = millis(); // exact time that message was sent
    for (int i = 3; i >= 0; i--)
      stmp0[i] = ptr_message[i];
    stmp0[4] = 1;
    CAN.sendMsgBuf(0x00, 0, 6, stmp0); // sends the second synchronization mesagge  0x00

    //reset_timer += 1; // timer's stamp for the plx-DAQ
    //enviaSinc(); // send sincronization data to plx-DAQ

    if (start > 0) {
      start += 1;
  }
}

ISR(TIMER2_OVF_vect)
{
  TCNT2 = 67; // timer value
  if (start > 0)
  {
    analogWrite(pwml, uk); // control action to the plant
    message_time = millis();
    tk += message_time - tiempo_anterior; // tk = tk + h
    tiempo_anterior = message_time;

    for (int i = 3; i >= 0; i--)
      stmp1[i] = ptr_tk[i];
```

```
    CAN.sendMsgBuf(0x01, 0, 4, stmp1); // sends tk message to the sensor 0x01
    //enviaSchedule(); // send schedule data to the plx-DAQ
 }
}

void loop()
{
  if (digitalRead(inicio) == HIGH) // system start
      start = 1;
  delay(1);

  if (start >= 10)
  {
    start = 0;
  }

  if (CAN_MSGAVAIL == CAN.checkReceive()) // check incoming data
  {
    CAN.readMsgBuf(&len, buf); // save data in the buffer
    if (CAN.getCanId() == 0x03) {
     uk = buf[0];
    }
  }
}
```