Control de trayectoria tolerante a fallas para sistemas discretos con aplicación a un robot agrícola.

Alumno
Nicolás Soncini
soncininicolas@gmail.com

Director
Ernesto Kofman
ekofman@gmail.com

Abril 24, 2021

A mi familia y amigos, sin quienes nada de esto podría haber sido posible.
A los docentes que me mostraron el camino en todas las etapas.
Al DCC, por ser un segundo hogar para mis estudios.

"La gente inteligente no se aburre, siempre encuentra algo para hacer."

- Anónimo

Resumen

En esta tesis de grado se desarrolla una nueva técnica de técnica de control tolerante a fallas para sistemas de tiempo discreto, se demuestran sus principales propiedades teóricas y se aplica sobre un sistema complejo consistente en el modelo de un robot agrícola.

La técnica de control desarrollada, basada en una técnica existente para sistemas de tiempo continuo, utiliza el concepto novedoso de conjuntos invariantes probabilísticos. Los mismos se utilizan para detectar la presencia de diversas fallas en el sistema, en función de la pertenencia de ciertas variables a dichos conjuntos.

La aplicación de esta técnica a un caso realista, en tanto, se realizó en el control de trayectorias de un modelo de robot agrícola. Si bien no se probó el sistema sobre el robot real, sí se lo hizo sobre simulaciones en tiempo real del mismo, incluyendo variaciones de parámetros y retardos que demuestran la robustez del esquema propuesto.

Índice

CAPÍTULO 1

Introducción

Un sistema de control automático es un mecanismo utilizado para actuar sobre cierto sistema (denominado "planta") con el objetivo de que este último se comporte de manera deseada. En los esquemas de control denominados "a lazo cerrado", esto se realiza mediante la medición de ciertas magnitudes, denominadas variables de "salida", y la manipulación de otras señales, denominadas variables de "entrada".

Los sistemas de control son generalmente diseñados a partir de un modelo matemático del sistema que desean controlar, pero es usual que este modelo sea sólo una aproximación del sistema real, o que en el comportamiento del día a día el mismo se vea afectado por diversas situaciones predecibles o impredecibles que lo alejan del comportamiento modelado. Es por esto que los mismos deben tener en cuenta la posibilidad de diversas fuentes de divergencia entre el sistema para el cual fueron diseñados (a partir del modelo) y el sistema real que manipulan.

Dentro de la teoría de control hay diversas ramas que tratan con las problemáticas que surgen entre los sistemas y sus modelos. Por un lado, la teoría de control robusto, que aborda el problema de garantizar que un sistema funcione adecuadamente aún bajo condiciones de incertidumbre en el modelo. Asimismo, la teoría de control estocástico trata con los problemas que surgen cuando los sistemas y sus mediciones estan afectados por señales desconocidas denominadas "ruido". Finalmente, el control tolerante a fallas busca garantizar el correcto funcionamiento de los sistemas de control cuando ocurren cambios repentinos e inesperados en el comportamiento de la planta, denominados "fallas".

Esta tesina, desde el punto de vista teórico, estará enfocada en el control tolerante a fallas, incorporando herramientas de control estocástico y control robusto.

1.1 Motivación

En nuestro trabajo nos interesa abordar las problemáticas antes mencionadas por medio de un sistema integrado que detecte fallas, reconfigure el sistema y sea tolerante a cierto ruido presente en la operación de los sistemas. Luego se focaliza en implementarlo como sistema de control de un robot agricultor, llamado robot desmalezador, presente en la institución anfitriona, el cual deseamos que opere nominalmente tanto bajo las incertezas del espacio físico en el cual se encuentra, como ante fallas de los motores que lo propulsan.

Para esto utilizaremos como base la metodología propuesta por Pizzi et al. [1] (que explicaremos en detalle en la sección 2.4). En dicha metodología se utiliza un esquema de detección de fallas basado en determinar la pertenencia de ciertas variables a determinados conjuntos probabilísticos y, ante la detección de fallas, se provee un mecanismo de reconfiguración del sistema de control.

Dado que dicha metodología está planteada para sistemas de tiempo continuo y que la implementación real implica usar esquemas muestreados, aquí extenderemos dicha metodología para sistemas de tiempo discreto.Esta extensión, demostrando las propiedades teóricas de estabilidad y capacidad de detección correcta de las fallas en sistemas de tiempo discreto, es la principal motivación de la Tesina.

Otra motivación importante es la implementación de la metodología desarrollada en un problema complejo en condiciones mucho más realistas que las estudiadas originalmente en [1]. Particularmente, buscaremos aplicar la metodología en un problema de control de trayectoria del robot desmalezador desarrollado en la institución anfitriona, implementando la misma sobre a una simulación en tiempo real del vehículo.

1.2 Organización de la Tesina

La tesina se divide en 5 capítulos, el primero de los cuales es el que se encuentra leyendo.

En el capítulo 2 se presenta el marco teórico y los conceptos previos que llevarán al entendimiento de la tesina. Se da una introducción a la teoría de control estocástica y tolerante a fallas, y finalmente un resumen de los resultados que pensamos adaptar.

En el capítulo 3 se presenta la teoría del sistema de control, con los teoremas y demostraciones que la respaldan. Se presentan el esquema de control, el modelo de planta a usar y los resultados teóricos obtenidos.

En el capítulo 4 mostramos una aplicación del sistema de control a una simulación de un robot agricultor. Presentamos el modelo del robot, los pasos a seguir y el diseño de

componentes del esquema, además de los resultados obtenidos, comparables a los de la tesina original.

En el capítulo 5 se muestra la implementación del sistema de control en lenguaje C, para su futuro uso en el control de un robot agricultor, junto a consideraciones y resultados de simulación del mismo con un modelo modélica, debido a la imposibilidad de acceder al robot real por la situación que transcurren los autores.

Finalmente el capítulo 6 presenta las conclusiones y el trabajo futuro.

CAPÍTULO 2

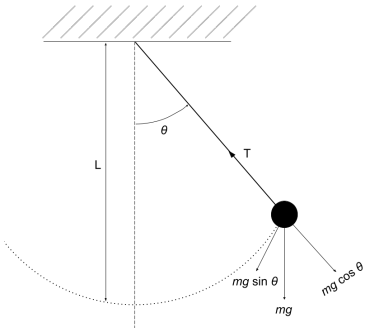
Conceptos Previos

En este capítulo se introducen los conceptos previos utilizados que serán necesarios para el entendimiento del resto de la tesina.

2.1 Sistemas dinámicos

Un sistema dinámico es una formalización matemática de una función que rige la dependencia en el tiempo, de un punto en un espacio de valores dado. Al punto del espacio en un momento dado se lo suele llamar "estado" del sistema, y está representado por una tupla o vector formado por las componentes del espacio en que se encuentra. El comportamiento del sistema está dado por una o mas reglas de "evolución" que describen los futuros estados en los cuales éste se encontrará, derivados a partir del estado actual.

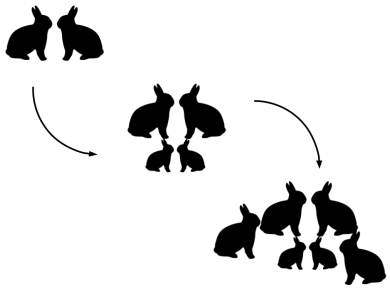
Los sistemas dinámicos son ideales para modelar las reglas que rigen la evolución en el tiempo de un sistema físico. Un ejemplo es el péndulo simple, cuya representación se muestra en la *Fig.* 2.1 y cuya función de evolución está dada por la ecuación diferencial de la misma figura, la cual modela los cambios en el ángulo que forma éste con su eje en tiempo continuo.



$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin\theta = 0$$

Figure 2.1: Péndulo simple, donde g es aceleración dada por la gravedad, l es el largo del péndulo, y θ es el desplazamiento angular.

Existen sistemas dinámicos cuya dinámica puede representarse en tiempo discreto. En ese caso la función de evolución suele estar dada por una ecuación en diferencias, como es el caso del sistema de la *Fig.* 2.2, que modela el crecimiento poblacional de una especie con intervalos regulares de reproducción.



$$p_{n+1} - p_n = kp_n + \beta$$

Figure 2.2: Población promedio en el instante n (p_n) regida por constante de reproducción k y constante de inmigración/emigración β.

Los sistemas dinámicos de tiempo discreto nos permiten simular el modelo en forma más exacta en sistemas de cómputo, que son inherentemente discretos en su forma de computar.

En nuestro trabajo nos basaremos en la teoría propuesta para sistemas lineales invariantes al tiempo (LTI). La propiedad de linealidad implica que la relación entre entradas y salidas del sistema es lineal, es decir:

- Si la entrada $x_1(t)$ produce la salida $y_1(t)$
- y la entrada $x_2(t)$ produce la salida $y_2(t)$
- entonces la entrada $x_1(t) + x_2(t)$ produce la salida $y_1(t) + y_2(t)$
- y la entrada $\alpha x_1(t)$ produce la salida $\alpha y_1(t)$,

por su parte la propiedad de invarianza en el tiempo implica que el sistema produce la misma salida ante la misma entrada, independientemente del instante en que se le aplique:

- Si la entrada $x(t)$ produce la salida $y(t)$
- entonces la entrada $x(t + T)$ produce la salida $y(t + T)$.

La mayoría de los sistemas físicos no se comportan en la realidad en forma LTI, ya que suelen sufrir de problemas como:

1. Saturación de señales: una entrada ilimitadamente grande al sistema no produce una salida de la misma magnitud, puesto que los sistemas no suelen poder reaccionar mas allá de cierto límite físico.
2. Envejecimiento: el uso de los mecanismos desgasta ciertos componentes, haciendo que en el tiempo sus comportamientos varíen.

3. No-linealidad: los comportamientos del sistema son inherentemente no-lineales.

Sin embargo son los sistemas que más se estudian, ya que son aquellos en que la matemática "es mas sencilla" y por consecuente mejores resultados pueden ser obtenidos[1].
Respecto a los sistemas no-lineales existen técnicas para "linealizarlos", es decir aproximarlos con un modelo lineal. Se toma un punto en el espacio de estados del sistema, llamado punto de operación, y un punto en el espacio de entradas al sistema, sobre los cuales se aproxima el sistema de forma lineal. De esta forma podemos asegurarnos que mientras el sistema se mantenga en cercanía a estos valores habremos hallado una buena aproximación de su comportamiento, en forma de ecuación lineal.

2.2 Control de Sistemas Dinámicos

La teoría de control se encuentra en la confluencia entre la matemática y la ingeniería en el estudio de los sistemas dinámicos. El objetivo básico de la teoría de control es manipular las entradas de un sistema dinámico para lograr que el mismo se comporte de una forma específica y deseada.
Un sistema de control moderno detecta y mide la operación del sistema, las compara con algún valor deseado, calcula acciones correctivas basadas en un modelo de la respuesta del sistema a entradas externas, y acciona el sistema para efectuar el cambio calculado[2]. Este tipo de sistema es conocido como control por retroalimentación, ya que sus salidas, luego de ser captadas por sensores, son redireccionadas en cierta forma como entradas al mismo sistema.
La problemática clave que se aborda el diseñar una lógica de control es la de asegurar que la dinámica del sistema controlado sea estable, es decir, que perturbaciones acotadas den lugar a errores acotados, y que tengan cierto comportamiento deseado adicional (buena atenuación de perturbaciones, rápida respuesta ante cambios de objetivo, etc)[2].
Para un desarrollo mas amplio del tema se recomienda al lector referirse a [2].

2.2.1 Modelado y Control de Sistemas Dinámicos

Los modelos de sistemas suelen describirse por $\mathrm{d}x/\,\mathrm{d}t = f(x,t)$ si se trata de modelos en tiempo continuo, o $x(t+1) = f(x,t)$ si se trata de modelos en tiempo discreto, donde x es una variable que representa el estado del sistema, t representa el tiempo y f es una función que rige la evolución del estado del sistema en el tiempo.

[1] Como dijo Richard Feynman en una de sus clases en Caltech: "Los sistemas lineales son importantes porque podemos resolverlos".

Si queremos modelar las posibles perturbaciones que el sistema puede sufrir podemos representarlas como una señal desconocida $w(t)$. En muchas aplicaciones esta señal se suele asumir como ruido blanco gaussiano, lo cual obliga a reescribir, para el caso de tiempo continuo, las ecuaciones diferenciales en la forma de ecuaciones diferenciales estocásticas: $\mathrm{d}x = f(x,t)\,\mathrm{d}t + g(x)\,\mathrm{d}w$. En el caso de tiempo discreto se reescriben las ecuaciones en diferencias en forma de ecuaciones en diferencia con un término estocástico: $x(t+1) = f(x,t) + g(x)w(t)$.
Ahora bien, si queremos manipular a nuestro sistema debemos modelar sus entradas, esto lo hacemos mediante una variable u que afecta directamente a nuestra función de evolución. Con esto obtenemos una ecuación de la forma $\mathrm{d}x = f(x,t,u(t))\,\mathrm{d}t + g(x)\,\mathrm{d}w$ para tiempo continuo, o $x(t+1) = f(x,t,u(t)) + g(x)w(t)$ para tiempo discreto.
El objetivo de control es, generalmente, encontrar una ley para $u(t)$ en función de $x(t)$ tal que este último se comporte de cierta manera (tienda a un valor, siga una determinada referencia, etc.).
En el caso de sistemas lineales, la función de evolución del sistema se representa mediante matrices de adecuado tamaño, con lo cual todo lo antes dicho queda en la forma $\mathrm{d}x = Ax(t)\,\mathrm{d}t + Bu(t)\,\mathrm{d}t + G\,\mathrm{d}w$ para sistemas de tiempo continuo o $x(t+1) = Ax(t) + Bu(t) + Gw(t)$ en el caso de tiempo discreto.

2.2.2 Conjuntos Invariantes y Cotas Finales

Los conjuntos surgen dentro de la teoría de control en forma natural cuando se desean analizar propiedades de restricción, incertezas y especificación de los diseños, como se plantea en [3]. Según este mismo autor, son los mas apropiados para estudiar y delimitar el accionar del ruido sobre los sistemas de control de retroalimentación.
Como sabemos, la estabilidad es una de las propiedades más importantes que un sistema puede poseer. Garantiza que las trayectorias, es decir los estados que se suceden en el tiempo, se acercan asintóticamente a los puntos de equilibrio. Ahora bien, ante la presencia de ruido no evanescente, es decir, ruido que no desaparece a medida que el sistema evoluciona, estos puntos de equilibrio dejan de ser tales y las trayectorias solo pueden acercarse y permanecer en un entorno de los mismos. Este entorno y sus propiedades pueden caracterizarse mediante un conjunto, llamado "cota final". Además, dentro de ellos, suele ser importante encontrar aquellos que a su vez son "invariantes", es decir, aquellos en los cuales las trayectorias del sistema no "escapan" debido a las perturbaciones.
El concepto entonces de punto de equilibrio (punto en el espacio desde donde las trayectorias no se mueven más y al que, bajo condiciones de estabilidad las trayectorias convergen) se ve reemplazado en presencia de perturbaciones por dos conjuntos:

la cota final y los invariantes.

2.2.3 Cotas Finales Probabilísticas

Llamamos cota final de un sistema a una región o conjunto acotado dentro del espacio de estados del mismo, en el cual su evolución queda confinada al mismo a partir de un cierto instante de tiempo. El principal problema es que en presencia de ruido blanco gaussiano no podemos obtener dicha cota normalmente, ya que por definición la señal no es acotada, y eso provoca que las trayectorias del sistema afectadas por el mismo nunca queden confinadas a una región acotada. Con cierta probabilidad, cualquier señal puede escapar de cualquier región en cierto momento. Es por esto que los sistemas estocásticos afectados por ruido blanco gausiano, en general, no tienen cota final.
Para subsanar este problema, en Kofman et al. [4] se propuso el concepto de "cota final probabilística", en la cual se establece una probabilidad de permanencia. La definición, para sistemas de tiempo discreto, es la que sigue:

Definición 1. *dado* $0 < p \leq 1$ *y* $S \subset \mathbb{R}^n$ *decimos que* S *es una cota final probabilística (PUB por sus siglas en inglés) con probabilidad* p *para el sistema* $x(t+1) = Ax(t) + w(t)$ *si para cada estado inicial* $x(t_0) = x_0 \in \mathbb{R}^n$ *existe* $T = T(x_0) \in \mathbb{N}$ *tal que la probabilidad* $Pr[x(t) \in S] \geq p$ *para cada* $t \geq t_0 + T$.

Resultados adicionales sobre las mismas pueden encontrarse en Kofman et al. [5].

2.3 Control tolerante a fallas

Como se describe en Blanke et al. [6], llamamos falla de un sistema a los cambios inesperados en comportamiento del mismo que resultan en que el sistema ya no satisfaga su objetivo de funcionamiento original. Las fallas pueden darse por eventos internos del sistema, cambios en las condiciones del ambiente de operación, errores en las señales de control o errores en el diseño del sistema bajo ciertas condiciones, entre otros.
Los sistemas de control clásicos suelen estar diseñados para la planta libre de fallas, de tal forma que el control a lazo cerrado cumpla ciertas especificaciones de desempeño. Para prevenir posibles problemas que puede causar el sistema ante la presencia de fallas las mismas deben de ser detectadas y evitadas, o tenidas en cuenta, lo antes posible; dichas medidas deben de ser llevadas a cabo por el propio sistema de control de la planta, el cual debe ser capaz de reaccionar ante la existencia de fallas, alterando su comportamiento original. A este acoplamiento entre control de fallas y control de la planta lo llamamos control tolerante a fallas.

La arquitectura general de control tolerante a fallas se presenta como una serie de sistemas o bloques acoplados al esquema de control original. Los bloques de "diagnóstico" y "rediseño de control" llevan a cabo los dos pasos fundamentales en la forma que muestra la *Fig.* 2.3.

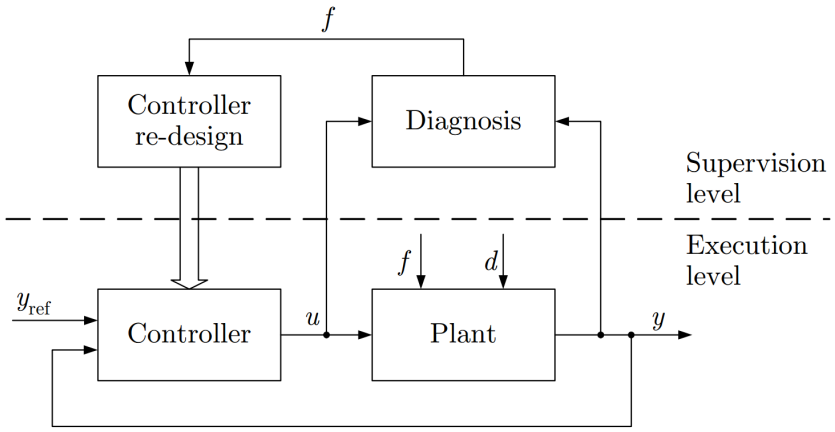


Figure 2.3: Arquitectura de control tolerante a fallas según se muestra en [6]. La conexión entre el bloque de rediseño de control y el bloque de control se presenta diferenciada ya que puede consistir en el cambio total de la configuración de este último.

El bloque de diagnóstico recibe las entradas y salidas medibles del sistema y testea su consistencia con el modelo de planta preexistente. Su resultado es una caracterización de la falla con suficiente precisión para poder rediseñar o reconfigurar el sistema de control.
Por su parte, el bloque de rediseño o reconfiguración utiliza la información provista de la falla en transcurso para ajustar el bloque de control y permitir la estabilidad del sistema de todas formas.
Como se menciona en [6], hay 3 estadíos que se suceden (posiblemente múltiples veces) en el tiempo en un sistema con control tolerante a fallas:

1. Previo a la ocurrencia de una falla el sistema es controlado utilizando el control nominal, los objetivos de control se satisfacen.

2. Entre la ocurrencia de una falla y el cambio de estrategia de control el sistema es controlado usando un controlador nominal, y los objetivos de control en general no se satisfacen.

3. Luego de la detección y reconfiguración el sistema vuelve a ser controlado por un nuevo sistema de control, los objetivos de control vuelven a satisfacerse.

Esta enumeración es completamente optimista, ya que nada garantiza que durante el segundo estadío el sistema no se vuelva inestable y deje de poder recuperarse su operación nominal. En nuestro trabajo probaremos que con el sistema de control toler-

ante a fallas que desarrollamos, bajo ciertas hipótesis, se puede garantizar la vuelta a operación nominal luego de la reconfiguración.
Para un trabajo más profundo del tema se refiere al lector a Blanke et al. [6].

2.4 Control tolerante a fallas basado en cotas finales probabilísticas para sistemas continuos

En Pizzi et al. [1] se propone un esquema de control tolerante a fallas basado en cotas finales probabilísticas para sistemas lineales y continuos bajo perturbaciones de tipo gaussianas. Describiremos a continuación el esquema propuesto y los resultados teóricos más importantes ya que nos basaremos fuertemente en ellos para desarrollar nuestro propio marco teórico para tiempo discreto.

2.4.1 Esquema

En primer lugar, [1] propone un esquema de control tolerante a fallas como se muestra en la *Fig.* 2.4.

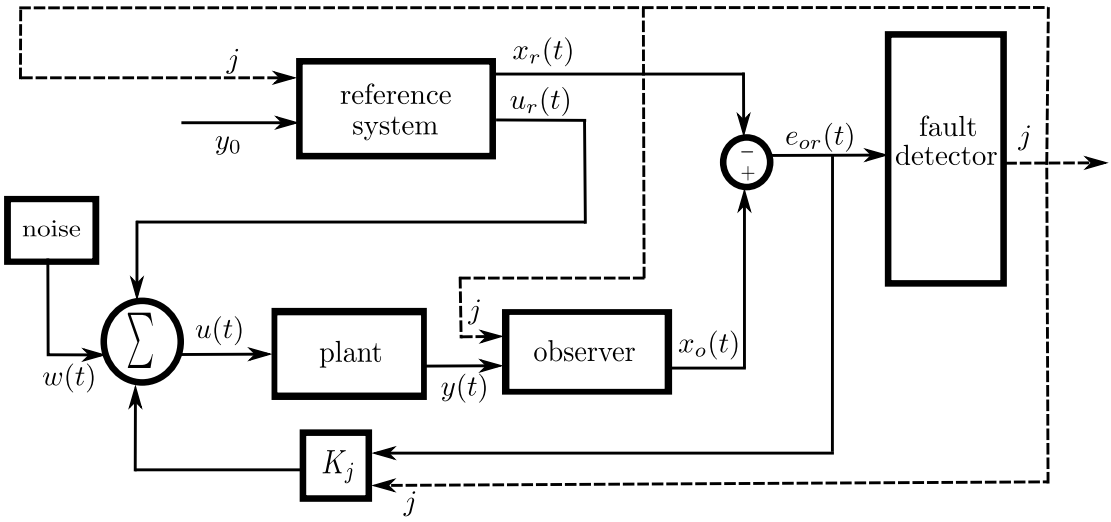


Figure 2.4: Esquema propuesto en [1].

Bajo este esquema asigna una estrategia de control que asegure el seguimiento de la dinámica de la planta bajo todas las fallas consideradas, un observador que se reconfigura dependiendo de la falla en transcurso, y un sistema de detección e identificación de fallas basado en cotas finales probabilísticas.
El modelo de planta consiste en un sistema LTI de la forma

$$dx(t) = Ax(t)\,\mathrm{d}t + BPu(t)\,\mathrm{d}t + F\,\mathrm{d}w(t)$$
$$y(t) = Cx(t),$$

donde x es el estado del sistema, u es la entrada, w es un proceso de Wiener (cuya derivada consiste en el llamado ruido blanco), y es la salida de la planta e A, B, F y C

son matrices de tamaños adecuados. Utiliza además una matriz P para representar las posibles fallas de actuadores del sistema, de la forma

$$\begin{aligned} P &\triangleq \text{diag}\{P_1, P_2, ..., P_m\}, \quad 0 \leq P_k \leq 1 \\ 0 < P_k < 1 &\iff \text{falla parcial en actuador } k \\ 0 = P_k &\iff \text{falla total en actuador } k \\ 1 = P_k &\iff \text{ninguna falla en actuador } k \end{aligned}$$

donde en ausencia de fallas la matriz P es la matriz de identidad. Considera un número finito de fallas posibles del sistema, representadas por $P = P^i$ (para $i = 0, ..., q$) que define como

$$\begin{aligned} P^0 &= I \\ P^i &= \text{diag}\{P_1^i, P_2^i, ..., P_m^i\}. \end{aligned}$$

El sistema de referencia esta dado por las ecuaciones libres de perturbaciones:

$$\begin{aligned} \mathrm{d}x_r(t) &= Ax_r(t)\,\mathrm{d}t + BP^j u_r(t)\,\mathrm{d}t \\ y_r(t) &= Cx_r(t), \end{aligned}$$

donde la entrada

$$u_r(t) = \overline{u}_r^j + \Delta u_r(t)$$

se calcula tal que $y_r(t)$ sigue exponencialmente una salida de referencia y_0, y j es la falla en transcurso, obtenida del sistema de detección de fallas. La entrada de referencia $u_r(t)$ se compone de una parte constante $\overline{u}_r^j$ y una parte variable $\Delta u_r(t)$.
El observador estima el estado de la planta para luego detectar fallas en la planta. Pizzi et al. [1] propone un observador que se adapta a la falla diagnosticada, la cual llamándola j caracteriza a la dinámica del observador por:

$$\mathrm{d}x_o(t+1) = Ax_o(t)\,\mathrm{d}t + BP^j u(t)\,\mathrm{d}t + L(y_p(t) - Cx_o(t))\,\mathrm{d}t,$$

para j = 0, . . . , q.
La entrada de control se computa en base al estado observado y las señales de referencia, y toma la forma:

$$u(t) = K_j(x_o(t) - x_r(t)) + u_r(t),$$

donde K_j representa la matriz de ganancia de retroalimentación diseñada para la j-ésima situación de falla (j = 0, . . . , q).

2.4.2 Resultados

El funcionamiento básico de éste esquema consiste en asegurar que existen dichos conjuntos PUB para las distintas fallas, esto es, asegurar que existe un PUB para el residuo

de la forma

$$e(t) \triangleq \begin{bmatrix} e_{or}(t) \\ e_{po}(t) \end{bmatrix},$$

donde e_{or} es el error entre el observador y la referencia, y e_{po} es el error entre la planta y el observador. Dichos PUB son calculados en la forma

$$S^{i,j} = \{e : |e_k - \bar{e}_k^{-i,j}| \leq b_k^{i,j} + \epsilon\}$$

donde $\bar{e}$ es una constante calculada a partir de la entrada constante a la planta. Los índices de S refieren a los conjuntos calculados para la situación en la que el sistema está configurado para la situación de falla j y transcurriendo la situación de falla i, donde i y j pueden ser iguales.

Luego de calculados los conjuntos, en [1] se demuestra que se puede detectar la ocurrencia de una falla mediante el filtrado simple del movimiento del residual entre dichos conjuntos, con una probabilidad arbitraria p de detección fallida. Esto se logra mostrando que dada la probabilidad p (mayor a una cierta constante) existe un tiempo $\hat{T}$ a partir del cual

$$\Pr\left[d^{i,j}(t) < d^{k,j}(t)\right] < \hat{\delta}; \quad \forall t > \hat{T} \tag{2.4.1}$$

donde $d^{i,j}$ es un filtro paso bajo sobre la función indicatriz (de pertenencia) del conjunto $S^{i,j}$, j es la configuración de falla actual del sistema e i es la falla en ocurrencia. Esta prueba asegura que luego de cierto tiempo de ocurrida la falla, la probabilidad de que el valor del filtro que detecta la falla en ocurrencia sea menor a la de uno de los filtros restantes es arbitrariamente pequeña (menor a un $\hat{\delta}$ arbitrario).

Finalmente en [1] se demuestra que la reconfiguración de la planta ante fallas mantiene las trayectorias del residuo convergentes en dichos PUB, siempre y cuando exista un cierto intervalo mínimo de tiempo entre la ocurrencia de consecutivas fallas. Esto es, dada una cota inferior T de los intervalos de reconfiguración del sistema y una probabilidad arbitraria $0 < p < 1$, puede encontrarse una cota final probabilística con probabilidad p para el residuo.

CAPÍTULO 3

Control tolerante a fallas basado en conjuntos probabilísticos para sistemas de tiempo discreto

Este capítulo presenta la principal contribución de la tesina, consistente en un esquema de control tolerante a fallas para una planta LTI de tiempo discreto afectada por perturbaciones gaussianas basado en cotas finales probabilísticas. A tal fin se deriva la dinámica de la planta discreta y se computan los PUB para cada una de las fallas en consideración. Luego se muestra que es posible detectar dichas fallas basados en la trayectorias del sistema y se describe un esquema de reconfiguración basado en la misma. Para ello se establecen los requisitos necesarios para preservar ciertas propiedades de lazo cerrado luego de que ocurren dichas reconfiguraciones.

3.1 Esquema de control

Siguiendo las ideas de Pizzi et al. [1] presentamos el esquema de control que asegura un buen seguimiento de la dinámica de la planta bajo todos los casos de falla de actuadores considerados.

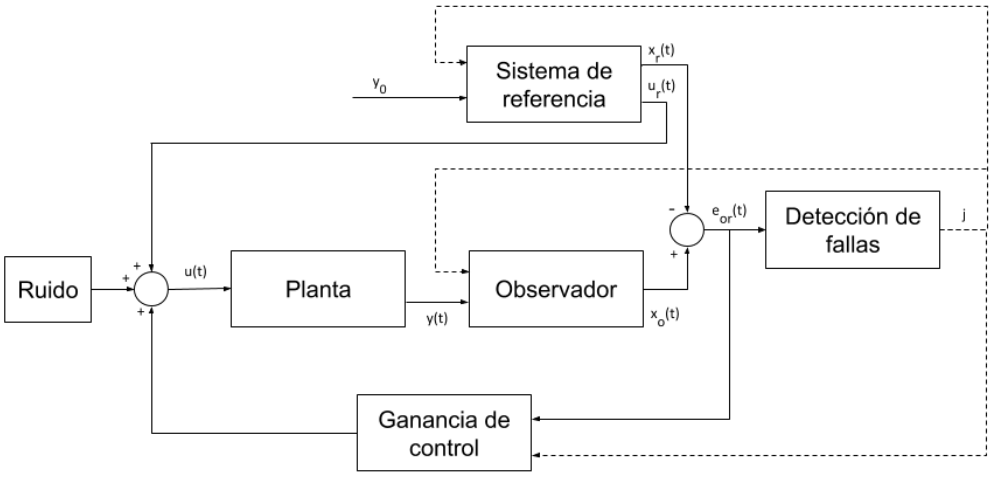


Figure 3.1: Esquema de control propuesto

El esquema se compone de la planta o sistema que se desea controlar, un observador que computa el estado observado a partir de las salidas del sistema, un sistema de referencia que modela el funcionamiento de la planta libre de ruido, una ganancia de control que provee control a partir del error entre el estado observado y el estado de referencia, y un sistema de detección de fallas que computa la falla predicha y la transmite a otros componentes del esquema.

3.2 Modelo de planta

La planta que controlaremos se modeliza con un sistema LTI (Lineal e Invariante en el Tiempo) perturbado por ruido blanco

$$\begin{aligned} x_p(t+1) &= Ax_p(t) + BPu(t) + Fw(t) \\ y_p(t) &= Cx_p(t) \end{aligned} \tag{3.2.1}$$

donde $x_p(t) \in \mathbb{R}^n$ es el vector de estados del sistema, $u(t) \in \mathbb{R}^m$ es el vector de entradas, $w(t)$ es un vector de ruido gaussiano con media cero y matriz de autocovarianza Σ_w, $y_p(t) \in \mathbb{R}^s$ es el vector de salidas, y A, B, C y F son matrices constantes de dimensiones apropiadas que representan la matriz de estados, matriz de entrada, matriz de salida y matriz de ruido respectivamente.
Al igual que en Pizzi et al. [1] la matriz P es utilizada para modelar los casos de falla de actuadores del sistema, donde

$$P \triangleq \text{diag}\{P_1, P_2, ..., P_m\}, \quad 0 \leq P_k \leq 1$$

$$0 < P_k < 1 \iff \text{falla parcial en actuador } k$$

$$0 = P_k \iff \text{falla total en actuador } k$$

$$1 = P_k \iff \text{ninguna falla en actuador } k$$

donde en ausencia de fallas la matriz P es la matriz de identidad. En nuestro caso consideraremos un número finito de fallas posibles del sistema, representadas por $P = P^i$ (para $i = 0, ..., q$) que definimos como

$$P^0 = I$$
$$P^i = \text{diag}\{P_1^i, P_2^i, ..., P_m^i\}.$$

De aquí en más asumiremos que el sistema (3.2.1) es estabilizable para todos los valores posibles de P considerados. Además denotaremos la falla que está afectando al sistema con un supra índice ($P = P^j$), con lo cual obtenemos la siguiente notación para la dinámica del sistema

$$\begin{aligned} x_p(t+1) &= Ax_p(t) + BP^j u(t) + Fw(t) \\ y_p(t) &= Cx_p(t) \end{aligned} \tag{3.2.2}$$

3.2.1 Planta de referencia

Utilizaremos un sistema de referencia para computar la entrada de referencia $u_r(t)$, y la trayectoria de los estados del sistema, $x_r(t)$. Bajo la j-ésima situación de falla estas trayectorias de referencia satisfacen el modelo libre de perturbaciones dado por las ecuaciones

$$\begin{aligned} x_r(t+1) &= Ax_r(t) + BP^j u_r(t) \\ y_r(t) &= Cx_r(t) \end{aligned} \tag{3.2.3}$$

donde la entrada

$$u_r(t) = \bar{u}_r^j + \Delta u_r(t) \tag{3.2.4}$$

se computa bajo una ley de control tal que $y_r(t)$ sigue exponencialmente la salida de referencia y_0 (esto es $\lim_{t\to\infty}[y_r(t) - y_0(t)] = 0$). Este último es el valor que finalmente querremos que describa nuestra salida de la planta y_p.
La entrada de referencia $u_r(t)$ está compuesta por una parte constante $\bar{u}_r^j$ y una parte variable $\Delta u_r(t)$, donde $\bar{u}_r^j$ representa la entrada en la situación del sistema en equilibrio, y $\Delta u_r(t)$ las variaciones en torno a la primera. Bajo condiciones de estabilidad, la segunda componente tiende a cero a medida que las trayectorias del sistema se aproximan al equilibrio.

3.2.2 Observador de planta

A fin de obtener un estimado del estado de la planta, que será utilizado para detectar y aislar las fallas de los actuadores del sistema, proponemos un observador que se adapte a diagnosticar la situación de falla caracterizada por

$$x_o(t+1) = Ax_o(t) + BP^j u(t) + L(y_p(t) - Cx_o(t)) \tag{3.2.5}$$

para j = 0, . . . , q. Las propiedades y restricciones que hacen a la matriz L se verán más adelante a la hora de analizar la dinámica de lazo cerrado.

3.2.3 Control de lazo cerrado

La entrada de control propuesta para la planta (3.2.2) está basado en el estado observado (3.2.5) y las señales de referencia (3.2.3) y (3.2.4), y toma la forma

$$u(t) = K_j(x_o(t) - x_r(t)) + u_r(t) \tag{3.2.6}$$

donde K_j representa la matriz de ganancia de retroalimentación diseñada para la j-ésima situación de falla (j = 0, . . . , q).

3.2.4 Detección y reconfiguración

Para la detección de fallas utilizamos una metodología similar a la de [1]. En la misma se calcula al residuo como la resta entre el estado calculado por el observador y el estado calculado por el sistema de referencia, y luego se verifica a qué conjunto PUB (que explicaremos más adelante) pertenece el mismo. Finalmente se realiza como método de "filtrado" una media móvil de las señales producidas por cada función indicatriz de los conjuntos PUB y se detecta como falla en transcurso aquella cuya media móvil sea mayor (esto es, el conjunto en el cual el residuo ha permanecido más tiempo en las últimas T muestras, donde T es el tamaño de ventana de la media móvil).

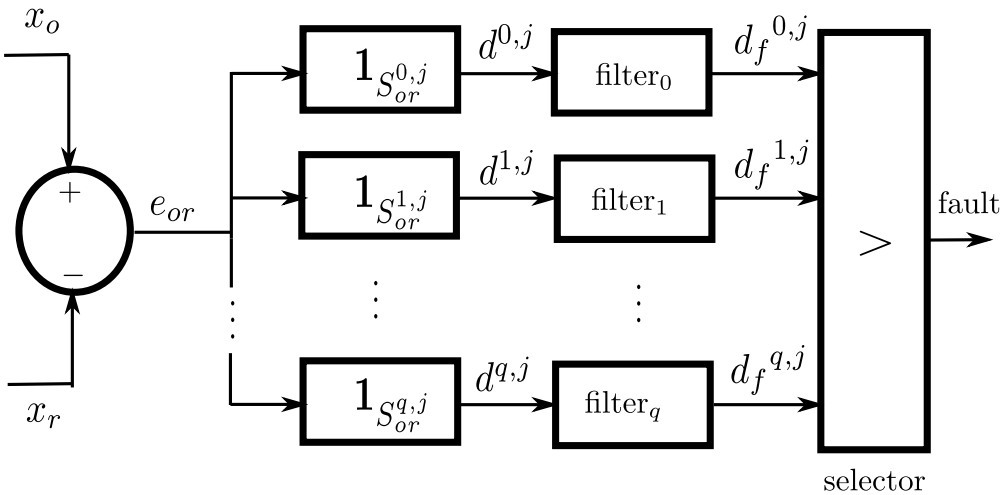


Figure 3.2: Funcionamiento de la detección de fallas en forma gráfica. $S^{i,j}$ conjuntos sobre los que se analiza la pertenecia del residuo (que describiremos más adelante), $d^{i,j}$ es el resultado de la función indicatriz del residuo sobre dichos conjuntos y $d_f^{i,j}$ es la salida de la media móvil. Imagen tomada de Pizzi et al. [1]

Por su parte, la reconfiguración del sistema se realiza ante una señal emitida por el bloque de detección de fallas que representa la falla en transcurso detectada por el

mismo. Esta señal permitirá que los sistemas de referencia, observador y ganancia de control se reconfiguren para controlar al sistema ante la nueva situación de falla.

3.3 Resultados

En esta sección se derivan las dinámicas de lazo cerrado de la planta, luego se presentan resultados para el calculo de cotas finales probabilísticas para la misma, y finalmente se determina un esquema de reconfiguración del sistema de control ante presencia de fallas.

3.3.1 Dinámica de lazo cerrado

De aquí en más consideraremos que el sistema está configurado para la j-ésima situación de falla y atravesando la i-ésima (con $i, j \in \{0, ..., m\}$ las situaciones de falla consideradas).
Definimos el error de seguimiento de la estimación de estado, que será utilizado como valor residual, como

$$e_{or}(t) \triangleq x_o(t) - x_r(t), \tag{3.3.1}$$

la ley de control (3.2.6) como

$$u(t) = K_j e_{or}(t) + u_r(t), \tag{3.3.2}$$

y el error de estimación de estado como

$$e_{po}(t) \triangleq x_p(t) - x_o(t). \tag{3.3.3}$$

Considerando el sistema de lazo cerrado con control (3.3.2), y utilizando las ecuaciones (3.2.1), (3.2.3) y (3.2.5), la dinámica del error de seguimiento de la estimación de estado $e_{or}(t)$ puede escribirse como

$$\begin{aligned} e_{or}(t+1) &= x_o(t+1) - x_r(t+1) \\ &= (A + BP^j K_j) e_{or}(t) + LCe_{po}(t) \end{aligned} \tag{3.3.4}$$

En forma similar, utilizando (3.2.1) y (3.2.5), la dinámica del error de estado $e_{po}(t)$ pueden reescribirse como

$$\begin{aligned} e_{po}(t+1) &= x_p(t+1) - x_o(t+1) \\ &= (A - LC) e_{po}(t) + B(P^i - Pj) K_j e_{or}(t) \\ &+ B(P^i - P^j) u_r(t) + Fw(t) \end{aligned} \tag{3.3.5}$$

De las ecuaciones (3.3.4) y (3.3.5), y definiendo

$$e(t) \triangleq \begin{bmatrix} e_{or}(t) \\ e_{po}(t) \end{bmatrix} \tag{3.3.6}$$

se obtiene el siguiente sistema

$$\begin{aligned} e(t+1) = &\begin{bmatrix} A + BP^jK_j & LC \\ B(P^i - Pj)K_j & A - LC \end{bmatrix} e(t) \\ &+ \begin{bmatrix} 0 \\ B(P^i - P^j) \end{bmatrix} u_r(t) + \begin{bmatrix} 0 \\ F \end{bmatrix} w(t) \end{aligned} \tag{3.3.7}$$

que modela la evolución del error de la planta.

Observación 1. *Para asegurar la estabilidad del sistema de lazo cerrado (3.3.7) las matrices de ganancia de retroalimentación K_j y la matriz del observador L en (3.2.5) tendrán que ser tales que*

$$A_\ell^{i,j} = \begin{bmatrix} A + BP^jK_j & LC \\ B(P^i - Pj)K_j & A - LC \end{bmatrix} \tag{3.3.8}$$

para $i, j \in \{0, ..., q\}$, sean estables (todo autovalor de las mismas se encuentre estrictamente dentro del círculo unitario complejo).

3.3.2 Calculo de cotas finales probabilísticas

Si asumimos que el sistema está configurado para la j-ésima situación de falla y atravesando la i-ésima (con $i, j \in \{0, ..., m\}$) podemos reescribir la ecuación (3.3.7) como

$$e(t+1) = A_\ell^{i,j} e(t) + B_\ell^{i,j} u_r(t) + Gw(t), \tag{3.3.9}$$

donde $A_\ell^{i,j}$ es como se define en (3.3.8), y donde

$$B_\ell^{i,j} \triangleq \begin{bmatrix} 0 \\ B(P^i - P^j) \end{bmatrix}, \; G \triangleq \begin{bmatrix} 0 \\ F \end{bmatrix}. \tag{3.3.10}$$

Entonces, utilizando el termino de referencia de entrada constante de (3.2.4) podemos definir

$$\bar{e}^{i,j} = \begin{bmatrix} \bar{e}_{or}^{i,j} \\ \bar{e}_{po}^{i,j} \end{bmatrix} \triangleq (I - A_\ell^{i,j})^{-1} B_\ell^{i,j} \bar{u}_r^j, \tag{3.3.11}$$

y considerando el siguiente cambio de coordenadas

$$\tilde{e}^{i,j}(t) = e(t) - \bar{e}^{i,j},$$

el sistema en (3.3.9) puede expresarse como

$$\begin{aligned} \tilde{e}^{i,j}(t+1) &= e(t+1) - \bar{e}^{i,j} \\ &= A_\ell^{i,j} e(t) + B_\ell^{i,j} u_r(t) + Gw(t) - \bar{e}^{i,j} \\ &= A_\ell^{i,j} e(t) + B_\ell^{i,j} \bar{u}_r^j + B_\ell^{i,j} \Delta u_r(t) + Gw(t) - \bar{e}^{i,j} \\ &= A_\ell^{i,j} e(t) + (I - A_\ell^{i,j}) \bar{e}^{i,j} + B_\ell^{i,j} \Delta u_r(t) + Gw(t) - \bar{e}^{i,j} \\ &= A_\ell^{i,j} e(t) + A_\ell^{i,j} \bar{e}^{i,j} + B_\ell^{i,j} \Delta u_r(t) + Gw(t) \\ &= A_\ell^{i,j} \tilde{e}^{i,j}(t) + B_\ell^{i,j} \Delta u_r(t) + Gw(t) \end{aligned} \tag{3.3.12}$$

Dado que en el cómputo de PUBs sólo nos interesa el comportamiento cuando $t \to \infty$, podemos ignorar el termino B_ℓ^{ij} del mismo, ya que $\lim_{t\to\infty} \Delta u_r(t) = 0$.
Como mencionamos en *Obs.* 1, la matriz $A_\ell^{i,j}$ en (3.3.12) es estable, y dado que $w(t)$ es ruido blanco gaussiano de media cero y matriz de auto-covarianza Σ_w, podemos utilizar el Teorema 14 de Kofman et al. [5] para computar la cota final, para $0 < p < 1$, como

$$\tilde{S}^{i,j} = \{\tilde{e}^{i,j} \in \mathbb{R}^{2n} : |\tilde{e}_k^{i,j}| \leq b_k^{i,j} + \epsilon;\ k = 1, \ldots, 2n\} \tag{3.3.13}$$

donde

$$b_k^{i,j} \triangleq \sqrt{2[\Sigma_x^{i,j}]_{k,k}}\text{erf}^{-1}(1 - \tilde{p}_k); \quad k = 1, \ldots, 2n$$

con $\tilde{p}_k \in (0, 1)$ tal que

$$\sum_{k=1}^{2n} \tilde{p}_k = 1 - p,$$

donde $\Sigma_x^{i,j}$ es la solución de la ecuación discreta de Lyapunov

$$A_\ell^{i,j}\Sigma_x^{i,j}(A_\ell^{i,j})^T + \Sigma_x^{i,j} = -G\Sigma_w G^T,$$

y erf es la función de error: $\text{erf}(z) \triangleq \frac{2}{\sqrt{\pi}} \int_0^z e^{-\zeta^2} d\zeta$.
Volviendo a las coordenadas originales ($e(t)$), el conjunto $\tilde{S}^{i,j}$ se transforma en

$$S^{i,j} = \{e \in \mathbb{R}^{2n} : |e_k - \bar{e}_k^{i,j}| \leq b_k^{i,j} + \epsilon;\ \ k = 1, \ldots, 2n\}, \tag{3.3.14}$$

que representa una cota final probabilística para el sistema (3.3.7) configurado para la j-ésima falla y transcurriendo la i-ésima falla.

3.3.3 Detección de fallas

Ya calculados los PUB correspondientes a las fallas, la idea de la detección de fallas es comprobar si el residuo e se ha asentado dentro de uno de estos conjuntos S. Sin embargo el componente e_{po} del mismo no puede ser medido, ya que desconocemos el estado interno de la planta si no es por sus salidas o la aproximación realizada por el observador. Por lo tanto debemos quedarnos sólo con la porción que podemos conocer, esto es, e_{or}.
Es por esto que a partir de ahora nos interesará trabajar con los siguientes conjuntos:

$$S_{or}^{i,j} = \{e_{or} \in \mathbb{R}^n : |e_{or} - \bar{e}_{or}^{i,j}| \preceq b_{or}^{i,j} + \bar{\epsilon}\},$$

donde $\bar{\epsilon} \triangleq [\epsilon, ..., \epsilon]$, $b_{or}^{i,j} \triangleq [b_1^{i,j}, ..., b_n^{i,j}]^T$, y el símbolo $\preceq$ representa la desigualdad componente a componente de los vectores correspondientes.
Entonces ahora la detección de fallas consiste en verificar si el error e_{or} se encuentra dentro de uno de los conjuntos S_{or}.

Lema 1. *Suponiendo al sistema en la configuración para la falla $j \in \{0,\dots,q\}$, y asumiendo que el sistema transcurre la i-ésima falla desde el instante de tiempo t_f^i. Entonces, dado $p \in (0,1)$, existe un $T > 0$ tal que la probabilidad $\Pr[e_{or}(t) \in S_{or}^{i,j}] \geq p, \forall t \geq t_f^i + T$.*

Proof. La prueba puede ser encontrada en Pizzi et al. [1], Lemma 1. □

En el contexto actual, el residuo del sistema puede fluctuar entre diferentes conjuntos PUB sin realmente estar ocurriendo dicha falla, o encontrarse fuera de cualquier conjunto aún en operación nominal. Esto se debe al ruido que experimenta el sistema; los conjuntos calculados nos aseguran con una probabilidad arbitrariamente grande pero nunca igual a uno.
El teorema que sigue nos plantea una forma de calcular cuándo el residuo se ha asentado en un PUB (acorde a las probabilidades asignadas), y así determinar la ocurrencia de una falla.

Teorema 1. *Suponemos al sistema bajo la configuración j, y transcurriendo la i-ésima situación de falla desde el instante de tiempo t_f^i. Asumiendo que los conjuntos PUB son disjuntos ($S_{or}^{i,j} \cap S_{or}^{k,j} = \emptyset$) para $i \neq k$, y definiendo*

$$d^{k,j}(t) \triangleq \mathbf{1}_{S_{or}^{k,j}}(e_{or}(t)); \quad k = 0,\dots,q, \tag{3.3.15}$$

donde $\mathbf{1}_S$ es la función indicatriz del conjunto S. Definiendo también, dado $N > 0$, la media móvil

$$\bar{d}_N^{k,j}(t) = \frac{1}{N} \sum_{\tau=t-N}^{t} d^{k,j}(\tau); \quad k = 0,\dots,q. \tag{3.3.16}$$

Entonces, si la probabilidad del PUB $p > \frac{1}{2}$ y el parámetro de la media $N > 0$ es lo suficientemente grande, entonces dado $\delta > 0$, existe un T tal que

$$\Pr\left[\bar{d}_N^{i,j}(t) < \bar{d}_N^{k,j}(t)\right] < \delta; \quad \forall t > T. \tag{3.3.17}$$

Proof. Dado[1] $t_0 \gg t_f^i$, para cada $\tau \geq t_0$, $e_{or}(\tau)$ es un proceso gaussiano uniformemente ergódico y estacionario. Sea $C^{i,j} \triangleq \mathbb{R}^n \setminus S_{or}^{i,j}$ el complemento de $S_{or}^{i,j}$, y tomando $t_1 \geq t_0$ resulta

$$\frac{1}{N} \sum_{\tau=t_1}^{t_1+N} \mathbf{1}_{C^{i,j}}(e_{or}(\tau)) \xrightarrow[N\to\infty]{p} \mathrm{E}[\mathbf{1}_{C^{i,j}}(e_{or}(t_1))]$$

uniforme en t_1. En particular, dados $\frac{\delta}{2} > 0$ y $\epsilon > 0$, existe un $N_0 > 0$ independiente de t_1 tal que para todo $N > N_0$ sea

$$\Pr\left[\frac{1}{N} \sum_{\tau=t_1}^{t_1+N} [1 - d^{i,j}(\tau)] \geq \mathrm{E}[\mathbf{1}_{C^{i,j}}(e_{or}(t_1))] + \epsilon\right] < \frac{\delta}{2},$$

[1] En particular un t_0 tal que $t_0 - t_f^i$ sea mayor que el tiempo en que $e_{or}(t)$ evoluciona dentro del conjunto $S_{or}^{i,j}$ con probabilidad mayor o igual a p, de forma de cumplir con 1

donde fue utilizado el hecho de que $\mathbf{1}_{C^{i,j}}(e_{or}(\tau)) = 1 - d^{i,j}(\tau)$.
Como $E[\mathbf{1}_{C^{i,j}}(e_{or}(t_1)] = \Pr[e_{or}(t_1) \notin S_{or}^{i,j}] = 1 - \Pr[e_{or}(t_1) \in S_{or}^{i,j}] \leq 1 - p$, entonces, para cada $N > N_0$

$$\Pr\left[\frac{1}{N}\sum_{\tau=t_1}^{t_1+N}[1 - d^{i,j}(\tau)] \geq 1 - p + \epsilon\right] < \frac{\delta}{2}.$$

Entonces con un poco de álgebra simple obtenemos

$$\begin{aligned}
&\Pr\left[\frac{1}{N}\sum_{\tau=t_1}^{t_1+N}[1 - d^{i,j}(\tau)] \geq 1 - p + \epsilon\right] \\
&= \Pr\left[\frac{1}{N}\sum_{\tau=t_1}^{t_1+N} 1 - \frac{1}{N}\sum_{\tau=t_1}^{t_1+N} d^{i,j}(\tau) \geq 1 - p + \epsilon\right] \\
&= \Pr\left[1 - \check{d}_N^{i,j}(t_1 + N) \geq 1 - p + \epsilon\right] \\
&= \Pr\left[\check{d}_N^{i,j}(t_1 + N) < p - \epsilon\right],
\end{aligned}$$

lo cual resulta en

$$\Pr\left[\check{d}_N^{i,j}(t_1 + N) < p - \epsilon\right] < \frac{\delta}{2}. \tag{3.3.18}$$

Ahora bien, tomando en consideración que $S_{or}^{i,j} \cap S_{or}^{k,j} = \emptyset \overset{i\neq k}{\Rightarrow} d^{k,j}(\tau) \leq 1 - d^{i,j}(\tau) \Rightarrow \check{d}^{i,j}(\tau) \leq 1 - \check{d}^{k,j}(\tau)$, y utilizando (3.3.18) podemos derivar que

$$\Pr\left[\check{d}_N^{k,j}(t_1 + N) > 1 - p + \epsilon\right] < \frac{\delta}{2}. \tag{3.3.19}$$

De las ecuaciones (3.3.18) y (3.3.19) y propiedades probabilísticas[2] obtenemos que

$$\Pr\left[\check{d}_N^{i,j}(t_1 + N) - \check{d}^{k,j}(t_1 + N) < 2p - 2\epsilon - 1\right] < \delta, \tag{3.3.20}$$

cualesquiera sean $t_1 \geq t_0$ y $N > N_0$. Por lo tanto, dado que tomamos $t_1 \geq t_0$, podemos asegurar que

$$\Pr\left[\check{d}_N^{i,j}(t_0 + N) - \check{d}^{k,j}(t_0 + N) < 2p - 2\epsilon - 1\right] < \delta. \tag{3.3.21}$$

Y si asumimos que p fue elegido de tal forma que $p > \frac{1}{2} + \epsilon$, obtenemos $2p - 2\epsilon - 1 > 0$. Entonces, utilizando las mismas propiedades probabilísticas que antes

$$\Pr\left[\check{d}_N^{i,j}(t_0 + N) < \check{d}^{k,j}(t_0 + N)\right] < \delta; \quad \forall N > N_0, \tag{3.3.22}$$

que concluye nuestra prueba, tomando $T = t_0 + N_0$. □

El *Teo.* 1 nos muestra que si el sistema está configurado para la j-ésima falla y en tiempo t_f^i la i-ésima falla ocurre, eligiendo un N lo suficientemente grande, la media móvil simple del i-ésimo conjunto tendrá una probabilidad arbitrariamente pequeña de hacerse más pequeña que cualquier otra señal. Esto es, podemos detectar la i-ésima falla con una probabilidad arbitrariamente pequeña de detección errada.
Una vez detectado el conjunto al cual la falla pertenece, tendremos que reconfigurar el sistema como se detalla en la siguiente sección.

[2]Dadas dos variables aleatorias $x \in \mathbb{R}, y \in \mathbb{R}$, y dos números reales a y b, los eventos $\{(x,y) : x < a\}$, $\{(x,y) : y > b\}$ y $\{(x,y) : x + b < y + a\}$ satisfacen $\{x + b < y + a\} \subset \{x < a\} \cup \{y > b\}$. Entonces $\Pr[x + b < y + a] \leq \Pr[x < a] + \Pr[y > b]$.

3.3.4 Esquema de reconfiguración

Cuando ocurre un cambio en la situación de falla que está transcurriendo nuestra planta debemos actuar para corregir o preservar el correcto funcionamiento del sistema. En estos casos procedemos a reconfigurar los sistemas de control, observador y referencia a la falla actual.
Ahora bien, luego de la reconfiguración, no podremos detectar otra falla hasta transcurrido T tiempo, esto es lo que plantea el *Teo.* 1.

Observación 2. *Observemos que en la prueba anterior demostramos que podemos detectar la falla en ocurrencia luego de transcurrido T tiempo desde la falla anterior. Para esto pedimos que las matrices $A_\ell^{i,j}$ sean estables, pero nada decimos de la estabilidad del sistema luego de una serie de fallas y reconfiguraciones arbitrarias.*

El teorema que sigue muestra que la trayectoria del sistema en la situación que plantea la Obs. 2 converge (en probabilidad) a una región o conjunto estable, siempre y cuando exista suficiente tiempo entre fallas de forma que el sistema converja al conjunto acotado correspondiente.

Lema 2. *Dada una matriz A con sus autovalores estrictamente dentro del círculo unitario, se cumple que $\|A^k\|$ (cualquiera sea la norma matricial) está acotada superiormente, para todo k. Más aún, existe $T > 0$ tal que se cumple que $\|A^k\| < 1$ para todo $k > T$.*

Proof. Si definimos el radio espectral de A como $\rho(A)$, entonces por hipótesis tenemos $\rho(A) < 1$. La fórmula de Gelfand nos dice que $\rho(A) = \lim_{k\to\infty} \|A^k\|^{1/k}$.
Tomando $\gamma \in [\rho(A), 1)$, y por la desigualdad que surge de la definición de límite infinito, existe $T > 0$ tal que $\forall k = T+1, T+2, \ldots$ se cumple que $\|A^k\| < \gamma^k < 1$. Hemos demostrado entonces la segunda proposición.
Ahora bien, si tomamos $M = \max\{\|A^k\|/\gamma^k \;;\; k = 0, \ldots, k_0\}$ tenemos $\|A^k\| < M\gamma^k$, que es válido para todo k, concluyendo así nuestra prueba. □

Teorema 2. *Consideremos el sistema descrito en* (3.3.9). *Asumamos que los autovalores de las matrices $A_\ell^{i,j}$ recaen todos dentro del círculo unitario, y que la señal de entrada de referencia $\bar{u}_r(t)$ es acotada. Asumamos también que los intervalos de conmutación son acotados por debajo por un $T > 0$ suficientemente grande, tal que $\|A_\ell^{i,j(t)}\| < 1$ para todo $t > T$ y todos i y j. Entonces, dada una probabilidad arbitraria $0 < p < 1$, se puede encontrar un PUB acotado S con probabilidad p para el sistema dado.*

Proof. Como en Pizzi et al. [1] comenzamos formando una sucesión t_k a partir de una sucesión de conmutación arbitraria $\tau_1, \tau_2, \ldots, \tau_j, \tau_{j+1}, \ldots$ que satisfaga $\tau_1 \geq t_0 + T$, de la siguiente forma

$$t_{k+1} = \begin{cases} \tau_j & \text{si } \tau_j \text{ es tal que } t_k + T \leq \tau_j \leq t_k + 2T \\ t_k + T & \text{en otro caso} \end{cases}$$

Donde $\{\tau_k\}$ esta contenido en $\{t_k\}$, y los puntos adicionales son añadidos para satisfacer

$$T \leq t_{k+1} - t_k < 2T \tag{3.3.23}$$

Sea $t \in (t_k, t_{k+1}]$. Como no hay instancias de conmutación en dicho intervalo, la solución de (3.3.9) es

$$e(t) = A_\ell^{i,j^{(t-t_k)}} e(t_k) + \sum_{\tau=t_k}^{t-1} \left(A_\ell^{i,j^{(t-\tau-1)}} B_\ell^{i,j} u_r(\tau) \right) + \sum_{\tau=t_k}^{t-1} \left(A_\ell^{i,j^{(t-\tau-1)}} G w(\tau) \right) \tag{3.3.24}$$

Aplicando la función de esperanza matemática a ambos lados de la ecuación obtenemos

$$\begin{aligned} E[e(t)] &= E\left[A_\ell^{i,j^{(t-t_k)}} e(t_k) + \sum_{\tau=t_k}^{t-1} \left(A_\ell^{i,j^{(t-\tau-1)}} B_\ell^{i,j} u_r(\tau) \right) \right. \\ &\quad \left. + \sum_{\tau=t_k}^{t-1} \left(A_\ell^{i,j^{(t-\tau-1)}} G w(\tau) \right) \right] \\ &= A_\ell^{i,j^{(t-t_k)}} E[e(t_k)] + \sum_{\tau=t_k}^{t-1} \left(A_\ell^{i,j^{(t-\tau-1)}} B_\ell^{i,j} u_r(\tau) \right) \end{aligned}$$

donde el término de ruido es cero ya que $w(t)$ tiene esperanza cero.
Si tomamos norma 2 a ambos lados obtenemos

$$\begin{aligned} \|E[e(t)]\| &= \| A_\ell^{i,j^{(t-t_k)}} E[e(t_k)] + \sum_{\tau=t_k}^{t-1} \left(A_\ell^{i,j^{(t-\tau-1)}} B_\ell^{i,j} u_r(\tau) \right) \| \\ &\leq \| A_\ell^{i,j^{(t-t_k)}} \| \| E[e(t_k)] \| + \sum_{\tau=t_k}^{t-1} \left(\| A_\ell^{i,j^{(t-\tau-1)}} B_\ell^{i,j} \| \| u_r(\tau) \| \right) \end{aligned} \tag{3.3.25}$$

Definamos

$$\gamma \triangleq \max_{i,j} \sup_{\tau > T} \| A_\ell^{i,j^{(\tau)}} \|; \quad v \triangleq \max_{i,j} \sup_{\tau > 0} \| A_\ell^{i,j^{(\tau)}} \| \tag{3.3.26}$$

cuya existencia ha sido demostrada en *Lema* 2. Nótese que cuando $\tau > T$ obtenemos $\| A_\ell^{i,j^{(\tau)}} \| < 1$ y entonces $\gamma < 1$. Además, por causa del teorema antes nombrado, v está acotada por alguna constante.
Definamos también

$$\eta \triangleq \max_{i,j} \sup_{t > 0} \sum_{\tau=0}^{t} \left(\| A_\ell^{i,j^{(\tau)}} B_\ell^{i,j} \| \bar{u}_r \right) \tag{3.3.27}$$

donde $\bar{u}_r$ es una cota superior para la señal acotada $\|u_r(\tau)\|$ para todo $\tau \geq t_0$. Además η es acotada por *Lema* 2 y porque la norma 2 es submultiplicativa.
Usando las ecuaciones (3.3.26)-(3.3.27) en la inecuación (3.3.25) con $t = t_{k+1}$ obtenemos

$$\|E[e(t_{k+1})]\| = \gamma \|E[e(t_k)]\| + \eta \tag{3.3.28}$$

Por lo tanto la secuencia $\|E[e(t_k)]\|$ está acotada superiormente por una sucesión monótona que converge a $\frac{\eta}{1-\gamma}$. Luego tenemos

$$\lim_{k \to \infty} \|E[e(t_{k+1})]\| \leq \frac{\eta}{1-\gamma} \tag{3.3.29}$$

y entonces, dado $\epsilon > 0$, existe K_ϵ tal que

$$\|E[e(t_k)]\| \leq \frac{\eta}{1-\gamma} + \epsilon, \quad \forall k \geq K_\epsilon \tag{3.3.30}$$

Usando ahora las ecuaciones (3.3.27)-(3.3.30) en la inecuación (3.3.25), obtenemos

$$\begin{aligned} \|E[e(t)]\| &\leq \|A_\ell^{i,j^{(t-t_k)}}\| \left(\frac{\eta}{1-\gamma} + \epsilon\right) + \eta \\ &\leq v\left(\frac{\eta}{1-\gamma} + \epsilon\right) + \eta \quad \triangleq \mu_\epsilon \end{aligned} \tag{3.3.31}$$

para todo $t > t_k$ con $k \geq K_\epsilon$.
Nótese que $t_{k+1} - t_k < 2T$ en (3.3.23) implica que $t_k < t_0 + 2kT$. Entonces si definimos

$$T_\epsilon \triangleq t_0 + 2K_\epsilon T$$

la condición $t \geq T_\epsilon$ implica que $t > t_k$ con $k = K_\epsilon$, y $\|E[e(t)]\| < \mu_\epsilon$ para todo $t \geq T_\epsilon$.
La matriz de covarianza de $e(t)$ en (3.3.24) está definida como $\Sigma_e(t) = E[(e(t) - E[e(t)])(e(t) - E[e(t)])^T]$.
Como el término $\sum_{\tau=t_k}^{t-1}(A_\ell^{i,j^{(t-\tau-1)}} B_\ell^{i,j} u_r(\tau))$ de (3.3.24) es determinista no contribuye a la matriz de covarianza de $e(t)$. Entonces podemos computarlo como $\Sigma_e(t) = \Sigma_z(t)$, donde

$$z(t) = A_\ell^{i,j^{(t-t_k)}} z(t_k) + \sum_{\tau=t_k}^{t-1} \left(A_\ell^{i,j^{(t-\tau-1)}} G w(\tau)\right)$$

Véase que $z(t_k) = e(t_k)$. Esta ecuación es la solución de una ecuación en diferencias estocástica lineal cuya varianza, como muestra Åström [7], está dada por

$$\Sigma_z(t) = A_\ell^{i,j^{(t-t_k)}} \Sigma_z(t_k) [A_\ell^{i,j^{(t-t_k)}}]^T + \sum_{\tau=t_k}^{t-1} A_\ell^{i,j^{(t-\tau-1)}} G \Sigma_w G^T [A_\ell^{i,j^{(t-\tau-1)}}]^T$$

que volviendo a $e(t)$ quedaría

$$\Sigma_e(t) = A_\ell^{i,j^{(t-t_k)}} \Sigma_e(t_k) [A_\ell^{i,j^{(t-t_k)}}]^T + \sum_{\tau=t_k}^{t-1} A_\ell^{i,j^{(t-\tau-1)}} G \Sigma_w G^T [A_\ell^{i,j^{(t-\tau-1)}}]^T$$

Tomando la norma 2 en ambos lados de la última ecuación

$$\begin{aligned} \|\Sigma_e(t)\| &\leq \|A_\ell^{i,j^{(t-t_k)}} \Sigma_e(t_k) [A_\ell^{i,j^{(t-t_k)}}]^T\| \\ &\quad + \sum_{\tau=t_k}^{t-1} \|A_\ell^{i,j^{(t-\tau-1)}} G \Sigma_w G^T [A_\ell^{i,j^{(t-\tau-1)}}]^T\| \\ &\leq \|A_\ell^{i,j^{(t-t_k)}}\|^2 \|\Sigma_e(t_k)\| + \delta \end{aligned} \tag{3.3.32}$$

donde

$$\delta \triangleq \max_{i,j} \sup_{t>0} \sum_{\tau=t_k}^{t-1} \|A_\ell^{i,j^{(\tau)}} G \Sigma_w G^T [A_\ell^{i,j^{(\tau)}}]^T\|$$

Utilizando (3.3.32) con $t = t_{k+1}$ obtenemos

$$\|\Sigma_e(t_k+1)\| \leq \|A_\ell^{i,j^{(t_{k+1}-t_k)}}\|^2 \|\Sigma_e(t_k)\| + \delta \leq \gamma^2 \|\Sigma_e(t_k)\| + \delta \tag{3.3.33}$$

Si consideramos al estado inicial $e(t_0)$ determinista tendremos que la covarianza del estado inicial es cero ($\Sigma_e(t_0) = 0$). Esto, junto a la ecuación anterior, nos permite acotar a $\|\Sigma_e(t_k)\|$ por una secuencia monótona que converge al valor $\frac{\delta}{1-\gamma^2}$, luego

$$\|\Sigma_e(t_k)\| \leq \frac{\delta}{1-\gamma^2}$$

para todo $k \geq 0$. Luego, usando (3.3.32) nos queda

$$\|\Sigma_e(t)\| \leq \|A_\ell^{i,j^{(t-t_k)}}\|^2 \|\Sigma_e(t_k)\| + \delta \leq v^2 \frac{\delta}{1-\gamma^2} + \delta \triangleq \sigma^2 \tag{3.3.34}$$

que implica que la matriz de covarianza de $e(t)$ está acotada en norma por σ^2 para todo $t \geq t_0$.

Sea $e_i(t)$ la i-ésima componente de $e(t)$, y recordando que $\|E[e(t)]\| \leq \mu_e$ (para $t \geq T_\epsilon$), $\|\Sigma_e(t)\| \leq \sigma^2$, y que $e(t)$ es $2n$ dimensional ($n \in \mathbb{N}$), tenemos que

$$|E[e_i(t)]| \leq \mu_\epsilon, \quad \forall t \geq T_\epsilon$$

y

$$[\Sigma_e(t)]_{i,i} \leq \|\Sigma_e(t)\|_\infty \leq \|\Sigma_e(t)\|\sqrt{2n} \leq \sigma^2\sqrt{2n} \tag{3.3.35}$$

Luego, para $t \geq T_\epsilon$, y para un dado $\tilde{p}_i$ tal que $0 < \tilde{p}_i < 1$, obtenemos

$$\begin{aligned}
&\Pr\left[|e_i(t)| \geq \mu_\epsilon + \sigma\sqrt[4]{\frac{2n}{\tilde{p}_i^2}}\right] \\
&\leq \Pr\left[|e_i(t)| \geq |E[e_i(t)]| + \sqrt{[\Sigma_e(t)]_{i,i}}\sqrt{\frac{1}{\tilde{p}_i}}\right] \\
&\leq \Pr\left[|e_i(t) - E[e_i(t)]| \geq \sqrt{\frac{[\Sigma_e(t)]_{i,i}}{\tilde{p}_i}}\right] \\
&\leq \tilde{p}_i
\end{aligned}$$

donde el último paso corresponde a la desigualdad de Chebyshev.

Tomando $0 < \tilde{p}_i < 1$ con $i = 1, \ldots, 2n$ tales que $\sum_{i=1}^{2n} \tilde{p}_i = 1 - p$, definimos el conjunto

$$S = \left\{ e \in \mathbb{R}^{2n} : |e_i| < \mu_\epsilon + \sigma\sqrt[4]{\frac{2n}{\tilde{p}_i^2}};\ \ i = 1, \ldots, 2n \right\} \tag{3.3.36}$$

y podemos demostrar que

$$\begin{aligned}
\Pr[e(t) \in S] &= 1 - \Pr[e(t) \notin S] \\
&\geq 1 - \sum_{i=1}^{2n} \Pr\left[|e_i(t)| \geq \mu_\epsilon + \sigma\sqrt[4]{\frac{2n}{\tilde{p}_i^2}}\right] \\
&\geq 1 - \sum_{i=1}^{2n} \tilde{p}_i = p
\end{aligned}$$

para $t \geq T_\epsilon$, probando así que el conjunto acotado S es un PUB con probabilidad p para el sistema (3.3.9).

□

Este último teorema prueba que las trayectorias del sistema convergen en probabilidad a un PUB, siempre que haya una cota en los intervalos de conmutación que satisfaga la hipótesis del teorema anterior.

CAPÍTULO 4

Aplicación

Este capítulo presenta la construcción del sistema de control tolerante a fallas basado en lo descrito anteriormente para un robot agrícola denominado Robot Desmalezador. En la primera sección se describe el modelado y dinámica del Robot Desmalezador. Luego de ello se muestra la construcción y simulación del sistema de control creados en la herramienta Simulink®de MATLAB®. Y finalmente se muestran resultados con un control discreto, comparables a los presentados para tiempo continuo en Pizzi et al. [1].

4.1 Robot Desmalezador

El modelo del robot desmalezador consiste en un vehículo de 4 ruedas, cada una con rotación y motor independientes entre sí, que se desplaza en un plano. Cada una de las 4 ruedas se modela teniendo en cuenta fricción contra el piso y desplazamiento.
La dinámica del robot se modela por medio de 9 variables de estado, que son la posición en el eje y, la rotación θ, las velocidades en ambos ejes vx y vy, la velocidad rotacional ω, y las velocidades rotacionales de cada rueda wlr, wrr, wlf, wrf:

$$\begin{bmatrix} y & \theta & vx & vy & \omega & wlr & wrr & wlf & wrf \end{bmatrix}^T.$$

Se ignora la posición en el eje x ya que nuestro objetivo de control será mover el vehículo a velocidad constante sobre el eje x, por lo cual se pueden ignorar además los ángulos de cada una de las ruedas.
El robot es controlado mediante 5 entradas, que son los 4 torques que ejercerá el motor sobre cada una de las ruedas, junto al ángulo de dirección:

$$\begin{bmatrix} Tlr & Trr & Tlf & Trf & \psi \end{bmatrix}^T.$$

Las ecuaciones (en formato Modelica®) del modelo se presentan en el Apéndice A, donde se incluye además una descripción de ciertos parámetros que se coinciden con los nombres de la *Fig.* A.1 allí presente.

4.2 Linealización y discretización del modelo

Para encontrar un sistema discreto que modele al robot en cuestión procedimos utilizando la función provista por MATLAB®llamada `dlinmod` que genera un modelo de espacio de estados lineal y discreto al rededor de un punto de operación. Para nuestros casos de uso discretizaremos alrededor de un punto del espacio de estados y un punto del espacio de entradas. La validez de dicho modelo lineal se mantendrá siempre y cuando el estado no se aleje mucho del punto de equilibrio, de acuerdo a la versión de tiempo discreto del teorema de Hartman-Grobman [8]. Si bien a priori es difícil estimar cuánto puede alejarse el estado del punto de linealización sin incorporar much error, veremos en las simulaciones que el esquema funciona correctamente incluso ante variaciones de parámetros en el sistema no lineal.
El punto en el espacio de estados, sobre el que se discretiza, corresponde a la velocidad en x de 2m/s, una velocidad rotacional de cada rueda de 2m/s, y el resto de los valores en cero:

$$\begin{bmatrix} y & \theta & vx & vy & \omega & wlr & wrr & wlf & wrf \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 2 & 2 & 2 & 2 \end{bmatrix}^T.$$

Estos valores se coinciden la operación nominal del robot, aún si difieren levemente del objetivo de control que propondremos mas adelante.
Por su parte, el punto en el espacio de entradas alrededor del cual se discretiza, corresponde a un torque de 10N/m para cada rueda y un ángulo de dirección de 0 grados:

$$\begin{bmatrix} Tlr & Trr & Tlf & Trf & \psi \end{bmatrix}^T = \begin{bmatrix} 10 & 10 & 10 & 10 & 0 \end{bmatrix}^T.$$

Lo que se obtienen son las matrices A, B y C que aproximan el sistema en forma discreta y lineal bajo un modelo de la forma:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) \\ y(t+1) &= Cx(t), \end{aligned},$$

las cuales toman los valores:

$$\mathrm{A} \approx \begin{bmatrix} 1.000 & 0.006 & 0 & 0.091 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.000 & 0 & 0 & 0.061 & -0.001 & 0.001 & -0.001 & 0.001 \\ 0 & 0 & 0.833 & 0 & 0 & 0.012 & 0.012 & 0.012 & 0.012 \\ 0 & 0.109 & 0 & 0.819 & 0.004 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.349 & -0.020 & 0.020 & -0.020 & 0.020 \\ 0 & 0 & 0.491 & 0 & -0.195 & 0.815 & 0 & 0.008 & 0 \\ 0 & 0 & 0.491 & 0 & 0.195 & 0 & 0.815 & 0 & 0.008 \\ 0 & 0 & 0.491 & 0 & -0.195 & 0.008 & 0 & 0.815 & 0 \\ 0 & 0 & 0.491 & 0 & 0.195 & 0 & 0.008 & 0 & 0.815 \end{bmatrix},$$

$$B \approx \begin{bmatrix} 0 & 0 & 0 & 0 & 0.003 \\ 0 & 0 & 0 & 0 & 0.009 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.055 \\ 0 & 0 & 0 & 0 & 0.147 \\ 0.018 & 0 & 0 & 0 & -0.029 \\ 0 & 0.018 & 0 & 0 & 0.029 \\ 0 & 0 & 0.018 & 0 & -0.029 \\ 0 & 0 & 0 & 0.018 & 0.029 \end{bmatrix}, C \approx \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

4.3 Diseño de componentes del esquema

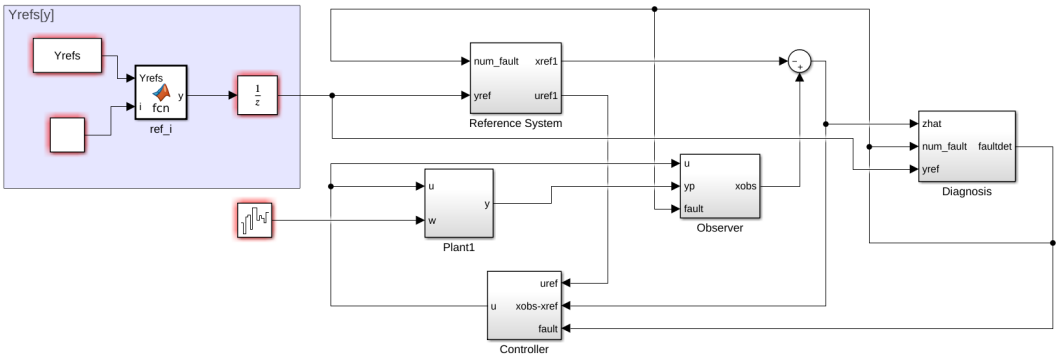


Figure 4.1: Esquema completo de Simulink®. Los parámetros (matrices y constantes) son pre-calculados en un espacio de trabajo de MATLAB®y utilizados por éste esquema al simular.

4.3.1 Sistema de referencia

Necesitamos implementar un sistema de referencia que cumpla con las premisas de que la entrada $u_r(t)$ sea tal que $y_r(t)$ siga exponencialmente la referencia y_0, como mencionamos en la Sección 3.2.1.

Para esto utilizamos el comando `lqi` de MATLAB®, que calcula la ley de control por retroalimentación óptima para el seguimiento de dicha referencia, utilizando un regulador lineal cuadrático con integral. El resultado es una matriz Kr_i para cada $i = 0, ..., nfaults$, la cual tiene la forma $Kr_i = [K_{pi} \; K_I]^T$, que satisface que la ley de control[1] $u_r(t+1) = -Kr \left[x(t), x_I(t)\right]^T$ permite que $y_r(t)$ siga exponencialmente la referencia y_0 para cada escenario de falla.

Luego utilizamos el sistema aumentado con $\xi(t) = \sum_{\tau=0}^{t} y_r(t) - y_0$, al cual podemos

[1] donde x_I es el output del integrador, calculado a partir de la fórmula de Euler: $x_I(t+1) = x_I(t) + Ts(y_0 - y_r(t))$

escribir en la forma

$$\begin{bmatrix} x(t+1) \\ \xi(t+1) \\ u(t+1) \end{bmatrix} = \begin{bmatrix} A-I & \varnothing & BP^i \\ C & \varnothing & \varnothing \\ K_P & K_I & \varnothing \end{bmatrix} \begin{bmatrix} x(t) \\ \xi(t) \\ u(t) \end{bmatrix} \tag{4.3.1}$$

con ley de control $u_i = -K_{P^i}x - K_I\xi$.

4.3.2 Observador

Como estimador del estado de la planta utilizamos un observador de Luenberger de matriz L, la cual satisface que $A - LC$ tiene todos los autovalores en el círculo unitario.

4.3.3 Sistema de Diagnóstico

Debido a la elección de leyes de retroalimentación de control podemos pre-calcular los conjuntos PUB parcialmente, independientes de y_r, difiriendo el cálculo de la entrada $\bar{u}_r$ al sistema de referencia a tiempo de ejecución. Lo cual nos permitirá modificar el objetivo de referencia del sistema en tiempo real.
Esto lo logramos pre-calculando las pseudo-inversas de las matrices del sistema aumentado

$$A3_i := \text{pinv}\left(\begin{bmatrix} A-I & \varnothing_{n,pref} & BP^i \\ Cr_i & \varnothing_{pref,pref} & \varnothing_{pref,m} \\ & Kr_i & -I \end{bmatrix}\right); \quad i = 0, ..., nfaults$$

, donde Cr_i es la matriz de salidas calculada para el sistema de referencia con la falla i, y dejando calculados parcialmente los valores de $\bar{e}$ de la forma

$$\bar{e}^{i,j}_{part} := (I - A^{i,j}_{\ell})^{-1}B^{i,j}_{\ell}; \quad i,j = 0, ..., nfaults$$

y los $b^{i,,j}_k$ como en (3.3.2).
Luego en tiempo de ejecución, con un valor de y_0 dado, y el sistema configurado para la falla j, calculamos

$$\begin{bmatrix} \_ \\ \_ \\ u^j_r \end{bmatrix} = A3_j \begin{bmatrix} \varnothing \\ y_0 \\ \varnothing \end{bmatrix} \tag{4.3.2}$$

(donde ignoramos las primeras filas porque sólo nos interesa u^j_r) lo que nos permite calcular entonces los PUB asociados como describe (3.3.14).
Una vez calculados los PUB, verificamos mediante la función indicatriz dentro de qué conjunto evoluciona el error y generamos un vector de componentes nulas, excepto en el índice de la falla que se sucede (si existe). Este vector es entrada de un filtro de tipo media móvil, que luego, tomando el índice de máximo valor de esta media, nos

permite encontrar qué falla está en ocurrencia. Una decisión de diseño es que si el valor máximo de la media es cero, es decir que en el "historial" que considera la media móvil el error evolucionó fuera de cualquier conjunto, consideramos que la falla se mantiene en el último valor detectado.

4.3.4 Control

Calculamos las ganancias de control con el comando `dlqr` de MATLAB®, que implementa un regulador lineal cuadrático discreto, devolviendo las matrices de ganancia óptima K_i para cada $i = 0, ..., nfaults$.
Las mismas minimizan la función de costo cuadrático

$$J(u) = \sum_{t=1}^{\infty} \left(x(t)^T Q x(t) + u(t)^T R u(t) \right)$$

para la entrada $u(t) = -K_i x(t)$ en el sistema

$$x(t+1) = Ax(t) + BP^i u(t),$$

donde las matrices Q y R son elegidas tal que se penaliza el error del estado 100 veces más que el error en las entradas de control.

4.4 Resultados

Se realiza una simulación del vehículo moviéndose en un plano. Los parámetros se consideran a semejanza de Pizzi et al. [1], para poder comparar los resultados obtenidos.
El objetivo de control es mover el vehículo a $1m/s$ manteniendo la dirección del eje x, y sin desviarse del cero en el eje y. Se contemplan las situaciones de falla en las que sólo dos motores pueden dejar de funcionar al mismo tiempo, es decir, 11 situaciones de falla, de las cuales una corresponde a la planta sin fallas, 4 corresponden a un único motor fallando y las 6 restantes a los pares de motores posibles, representados por las matrices P^i:

$$P^0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, P^1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, P^2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$P^3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, P^4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, P^5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$P^6 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, P^7 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, P^8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$P^9 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, P^{10} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

En el modelo se introduce ruido blanco gaussiano w a la entrada al sistema con una matriz de covarianza incremental $\Sigma_w \, \mathrm{d}t = 2 \, \mathrm{d}t$, el cual ingresa al sistema multiplicado por la matriz:

$$G = \begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.05 \end{bmatrix},$$

en la forma $x(t+1) = Ax(t) + BPu(t) + Gw(t)$.

El cálculo de PUBs se realiza utilizando una probabilidad $p = 0.99$, y para la detección de fallas se utiliza una media móvil simple de 10 muestras.

Durante la simulación el sistema transcurre las fallas que se muestran en *Fig.* 4.2, donde el índice de fallas se corresponde con los índices i de las matrices P^i antes detalladas. El tiempo de muestra es de 0.1 segundos, por lo cual se presenta el eje x en Segundos / Tiempo de Muestras (o en inglés Time / Sample Time), donde en este caso hay 10 muestras por segundo, por lo tanto el eje x tiene una unidad de 100ms.

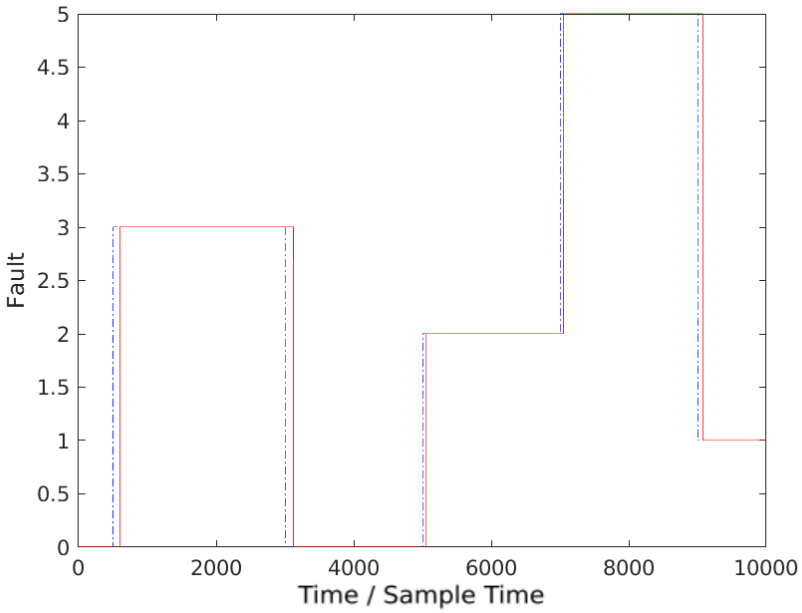
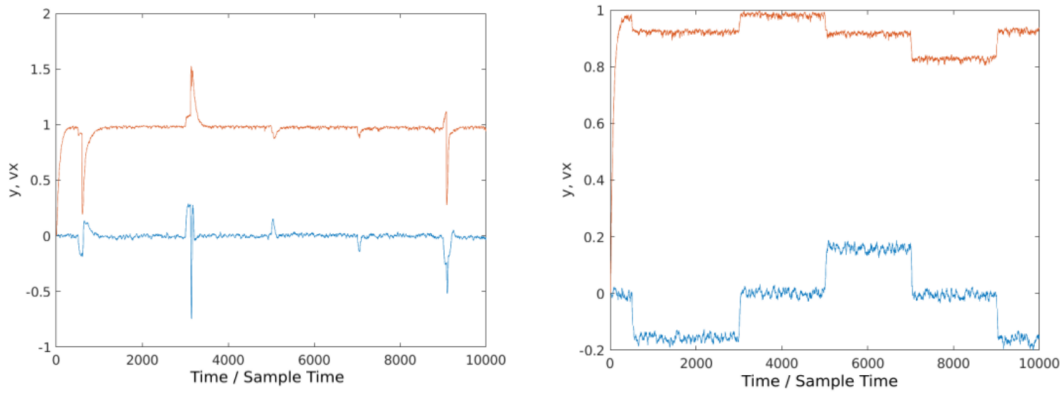


Figure 4.2: Fallas en ocurrencia (azul, guión y punto) y fallas detectadas (rojo, continuo).

La *Fig.* 4.2 muestra que las fallas son correctamente detectadas con un mínimo delay entre su ocurrencia y su detección. Por su parte la *Fig.* 4.3 muestra que la reconfiguración, implementada como se explica en 4.3.3, del sistema funciona como es esperado, con un buen seguimiento de la trayectoria deseada.

Al comparar los resultados con los obtenidos en Pizzi et al. [1] podemos ver que el tiempo de respuesta para la detección y reconfiguración, así como las salidas del sistema son sumamente similares. El resultado es significativo ya que contamos con un sistema de control trabajando en tiempo discreto, generado a partir de una linealización y discretización del modelo original, sin embargo la planta controlada se simula a partir del modelo no-lineal y en tiempo continuo.



(a) **(b)**

Figure 4.3: Las figuras 4.3a y 4.3b muestran las salidas del sistema con y sin reconfiguración respectivamente. Se muestra la posición en y (azul, cercano al cero) y la velocidad en x (rojo, cercano al uno).

CAPÍTULO 5

Implementación

En este capítulo se presenta la implementación del esquema de control presentado anteriormente, en código C, tal cual correrá como control del robot en cuestión. Además se muestra una simulación del mismo corriendo como entrada a un modelo Modelica®del robot, al cual se lo conecta en forma de hardware-in-the-loop, lo cual garantiza la correctitud al utilizarlo como esquema de control del robot real.
Hardware-in-the-loop (HiL) es una técnica usada para el desarrollo y comprobación de sistemas embebidos en tiempo real mediante la simulación de sensores y actuadores como interfaz entre el modelo de planta y el sistema integrado bajo prueba. La simulación por HiL consiste en el desarrollo de un sistema de simulación de tiempo real que modele algunas partes del sistema embebido bajo testeo y todas las interacciones significativas con su ambiente operacional, como se detalla en [9].
En nuestro caso modelaremos la comunicación de tiempo real y asíncrona entre la simulación del robot en cuestión y la implementación del sistema de control, lo cual nos permitirá analizar la resiliencia del esquema completo ante los retardos de tiempo de comunicación no modelados.

5.1 Consideraciones generales

Para la implementación en código C reutilizamos, siempre que sea posible, las matrices previamente computadas en la preparación para la simulación MATLAB®/Simulink®. Esto es las matrices que representan al sistema linealizado y discretizado, las matrices que describen los conjuntos PUB, las matriz del observador, etc. De esta forma sólo debemos diseñar e implementar el sistema de tiempo de ejecución del sistema de control.
Para todos los cálculos matriciales y/o vectoriales hacemos amplio uso de la librería GSL (GNU Scientific Library)[10] que provee tipos de datos para una eficiente representación de vectores y matrices, así como una interfaz estable de interacción con librerías BLAS (Basic Linear Algebra Submodules) [11] para operaciones entre dichos

tipos.
El código presente en el Apéndice B se propone como implementación del esquema de control (sin considerar la planta, por supuesto). Dicha implementación se encuentra lista para correr sobre un sistema de cómputo digital como control al robot físico, pero la imposibilidad de acceder al robot ante la situación actual[1] nos lleva a utilizar una simulación en su lugar.
Como parte de la implementación se presentan los siguientes archivos (se muestra el nombre sin sufijo para tener en cuenta ambos `.h` y `.c`):

- `control_agrobot`: en este archivo se implementa el esquema de control propuesto, esto es, observador, referencia, diagnóstico y ganancia de control.
- `control_agrobot_server_rt`: en este archivo se implementa la lógica de servidor que servirá para comunicar el resultado de control al destinatario, en nuestro caso mediante memoria compartida.
- `real_time_modelica`: aquí presentamos los métodos utilizados por la librería Modelica®para sincronizar el tiempo de simulación con el tiempo real.
- `semaphore_memory`: este archivo presenta una estructura de datos y métodos que permiten la comunicación mediante memoria compartida del sistema de control y el robot.

5.2 Sistema de control

La implementación del sistema de control está dada por las cabeceras de `control_agrobot.h` y el cuerpo en `control_agrobot.c`.
En las cabeceras podemos encontrar definiciones de constantes necesarias para la correcta asignación de espacio y operación del sistema, como por ejemplo los tamaños de las matrices, el tiempo de muestreo en segundos, el numero de fallas y el tamaño de la media móvil, entre otras. Además encontramos estructuras de C que separan los distintos estados internos que requiere cada parte de nuestro esquema de control, con cada estructura conteniendo una o más matrices y/o vectores. Cada estructura viene acompañada de un constructor y un destructor de la misma, que asignan y liberan espacio de memoria respectivamente.
Los componentes del esquema son funciones que toman sus entradas, su estado interno (representado como los punteros a estructuras C antes mencionadas), y sus salidas (representadas como punteros a memoria, para minimizar las asignaciones de espacio de memoria del sistema). Además algunos componentes requieren conocer las matrices

[1] Se refiere al lector a interiorizarse en la situación pandémica que atraviesan los autores en `https://es.wikipedia.org/wiki/COVID-19`

que modelan al sistema en espacio de estados, las cuales se proveen como parámetro a través de un puntero a una estructura que las contiene.

5.3 Sincronización con el tiempo real

Para hacer uso de la técnica HiL decidimos correr la simulación del robot en Modelica®sincronizada con tiempo real. Esto se realiza conectando al modelo con una función externa en C que realiza lo conocido como espera activa (busy-waiting) hasta que el tiempo transcurrido en la simulación se corresponde con el real (siempre y cuando la simulación corra al menos tan rápido como el tiempo real).
En las cabeceras del código encontramos una estructura C que lleva dos variables de tiempo, una que se utilizará para guardar el tiempo y otra en la cual se lleva el tiempo inicial en forma de real de doble precisión. Además encontramos una macro de C que nos permite transformar una estructura de tiempo `timespec` en un real de doble precisión.

real_time_modelica.h

```
#define TIMESPEC_USEC(t) ((double)t.tv_sec + (double)t.tv_nsec / 1e9)

typedef struct {
    struct timespec time;
    double initial_time;
} TimeKeeper;
```

Los métodos definidos nos permiten construir la estructura antes mencionada al inicio de la simulación, guardando en ella el tiempo inicial provisto por el sistema operativo, y sincronizar la simulación con el tiempo real esperando suficiente tiempo hasta que el tiempo de simulación se haya cumplido. Si el tiempo de simulación es más lento que el tiempo real, nada puede asegurarse aquí.

real_time_modelica.c

```
void *real_time_constructor_modelica()
{
    TimeKeeper *timekeeper = (TimeKeeper *) malloc(sizeof(TimeKeeper));

    // get current time
    clock_gettime(CLOCK_MONOTONIC, &(timekeeper->time));

    // fill initial time
    timekeeper->initial_time = TIMESPEC_USEC(timekeeper->time);

    return (void *)timekeeper;
}

void real_time_destructor_modelica(void* object)
{
    TimeKeeper *timekeeper = (TimeKeeper *) object;
    free(timekeeper);
}

void synchronize(void *object, double sim_time)
{
    TimeKeeper *timekeeper = (TimeKeeper *) object;
    double cur_time;
```

```
    do {
        if (0 != clock_gettime(CLOCK_MONOTONIC, &(timekeeper->time))){
            exit(420);
        }
        cur_time = TIMESPEC_USEC(timekeeper->time);
    } while (cur_time - timekeeper->initial_time < sim_time);
}
```

5.4 Conexión de control y planta

Para concretar la técnica de HiL tenemos que comunicar la simulación Modelica®con el proceso que está corriendo el sistema de control. Para esto implementamos un sistema de comunicación por medio de memoria compartida[2] en el cual el proceso de control lee de un espacio de memoria las salidas del sistema y escribe en otro espacio de memoria la entrada de control, esperando un tiempo arbitrario (en nuestro caso 0.1s) hasta la siguiente iteración.
En las cabeceras del archivo de comunicación encontramos una estructura que lleva el identificador del espacio de memoria `shmid`, un puntero al espacio de memoria `shmdata` y dos punteros a espacios de memoria que serán ubicados dentro de la memoria compartida: el puntero a los valores de interés `vals` y el puntero a un semáforo `sem` que actuará de árbitro de lecturas y escrituras a la memoria.

semaphore_memory.h

```
#define SHM_SIZE 1024

typedef struct {
    int shmid;
    void *shmdata;
    double *vals;
    sem_t *sem;
} SemaphoreMemory;
```

En el código se observan las funciones de construcción y destrucción de la estructura, donde se pide el espacio de memoria compartida al sistema operativo y se inicializa el semáforo. Y las funciones de lectura y escritura a este espacio de memoria en donde se toma el semáforo cuando esté disponible y se libera luego de operar en ella.

semaphore_memory.c

```
void *semmem_constructor(const char* filename)
{
    SemaphoreMemory *semmem = (SemaphoreMemory *) malloc(sizeof(SemaphoreMemory));

    key_t key;
    // create shared key
    if((key = ftok(filename, 'R')) == -1){
        exit(99);
    }

    // create shared memory segment
    if((semmem->shmid = shmget(
```

[2]Decidimos hacerlo así para aminorar la latencia de escritura y lectura en archivos ya que la simulación se realiza en un sistema Linux sin garantías de tiempo.

```
        (key_t)key, sizeof(double) * SHM_SIZE + sizeof(sem_t), IPC_CREAT | 0666)) == -1){
        exit(100);
    }

    // attach to the segment to get an address
    if((semmem->shmdata = shmat(semmem->shmid, NULL, 0)) == (void *)(-1)){
        exit(101);
    }

    // get the relevant sections pointers (that is, values and semaphore)
    semmem->vals = (double *)semmem->shmdata;
    memset(semmem->vals, 0, sizeof(double) * SHM_SIZE);
    semmem->sem = (sem_t *)(semmem->shmdata + sizeof(double) * SHM_SIZE);

    //initialize semaphore
    if(sem_init(semmem->sem, 1, 1) == -1){
        exit(102);
    }

    return (void *)semmem;
}

void semmem_destructor(void *object)
{
    SemaphoreMemory *semmem = (SemaphoreMemory *)object;

    shmdt(semmem->shmdata);

    free(semmem);
}

void semmem_write(void *object, double *data, size_t data_sz)
{
    SemaphoreMemory *semmem = (SemaphoreMemory *)object;

    //get semaphore
    sem_wait(semmem->sem);

    memcpy(semmem->vals, data, sizeof(double)*data_sz);

    //release semaphore
    sem_post(semmem->sem);
}

void semmem_read(void *object, double *data, size_t data_sz)
{
    SemaphoreMemory *semmem = (SemaphoreMemory *)object;

    //get semaphore
    sem_wait(semmem->sem);

    memcpy(data, semmem->vals, sizeof(double)*data_sz);

    //release semaphore
    sem_post(semmem->sem);
}
```

Ambas estructuras para ser utilizadas dentro de modélica se implementan como una extensión a la clase `ExternalObject` de Modelica®, que nos permite mantener un objeto externo (por ejemplo un puntero a memoria C, o a una estructura C). Es necesario definir dos métodos para todo objeto externo, un constructor y un destructor, que serán llamados adecuadamente cuando la simulación lo considere apropiado.
En nuestro caso encapsulamos tanto el tiempo real como la memoria compartida en dos clases `RealTime` y `SemMem` de la siguiente forma

RealTime.mo

```
class RealTime
```

```
  extends ExternalObject;

  function constructor
    output RealTime timekeeper;

    external "C" timekeeper = real_time_constructor_modelica() annotation(Library = "modelica_hil");
  end constructor;

  function destructor
    input RealTime timekeeper;

    external "C" real_time_destructor_modelica(timekeeper) annotation(Library = "modelica_hil");
  end destructor;
end RealTime;
```

y

SemMem.mo

```
class SemMem
extends ExternalObject;

  function constructor
    input String dir;
    output SemMem mem;
    external "C" mem = semmem_constructor(dir) annotation(Library="modelica_hil");
  end constructor;

  function destructor
    input SemFile mem;
    external "C" semmem_destructor(mem) annotation(Library="modelica_hil");
  end destructor;

end SemMem;
```

Desde el punto de vista del controlador C implementamos un servidor que lee las salidas de la planta mediante los métodos de memoria compartida, calcula la entrada de control y la escribe en otra memoria compartida mediante los mismos métodos:

control_agrobot_server_rt.c

```
int main(void) {

    SemaphoreMemory *control_mem = semmem_constructor(CONTROL_FILE);
    SemaphoreMemory *outputs_mem = semmem_constructor(OUTPUTS_FILE);

    double outputs_data[_P + _M];

    // Ready control states
    FullState fs = (FullState)control_state_constructor();

    while (1) {

        // Read data
        semmem_read(outputs_mem, outputs_data, _P + _M);

        // Process data
        // view input as vectors
        Vec_v y_vec = gsl_vector_view_array(outputs_data, _P);
        PVEC(&y_vec.vector)
        Vec_v yr_vec = gsl_vector_view_array(outputs_data + _P, _M);
        PVEC(&yr_vec.vector)
        // allocate memory for output
        double u[_M];
        Vec_v u_vec = gsl_vector_view_array(u, _M);
        //perform control step
        control_agrobot(&y_vec.vector, &yr_vec.vector, fs, &u_vec.vector);
        PVEC(&u_vec.vector)

        // Write response
        semmem_write(control_mem, u, _M);
```

```
        usleep((unsigned int)100000); //TODO use macro TS from control_agrobot?
    }

    semmem_destructor(control_mem);
    semmem_destructor(outputs_mem);

    return 0;
}
```

los archivos aquí no listados con los nombres `CONTROLS_FILE` y `OUTPUTS_FILE` se utilizan tanto en la simulación como aquí para generar una clave que identifique de forma única a los espacios de memoria compartida que utilizan ambos.
De esta forma el esquema de HiL que realizamos se comporta como muestra la *Fig.* 5.1

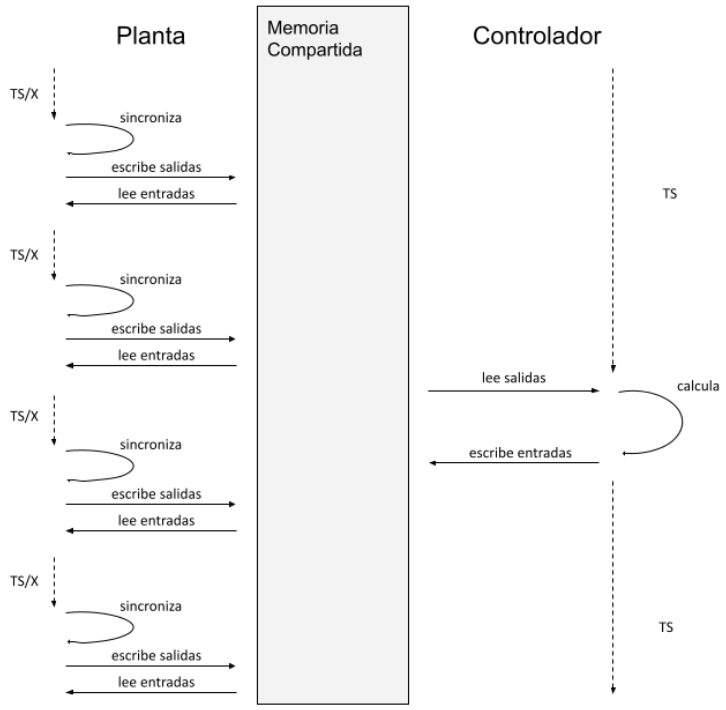
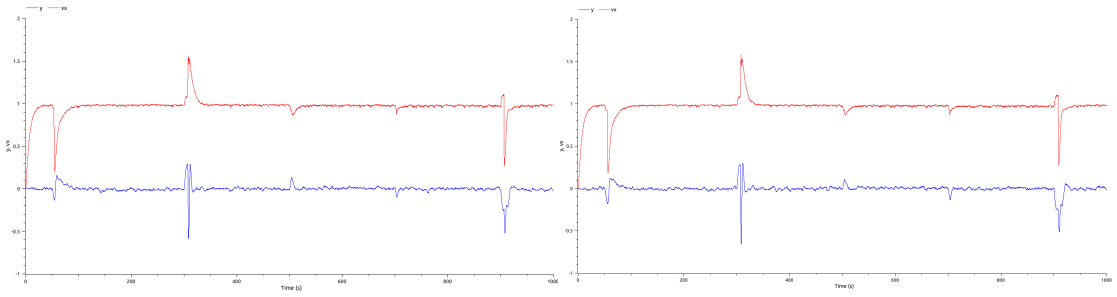


Figure 5.1: Esquema de comportamiento. El paso del tiempo del modelo es menor ya que debe sincronizar con el tiempo real. Ningún componente sabe del otro, sólo conocen la memoria compartida. La planta leerá las mismas entradas de control múltiples veces hasta que suceda un paso del controlador.

5.5 Resultados

Se realiza una simulación del vehículo idéntica a la presentada en la sección 4.4, donde la dinámica del vehículo es simulada con un modelo modélica y el sistema de control es el descrito en este capítulo, el cual corre como entrada al modelo mediante el sistema de memoria compartida antes descrito.



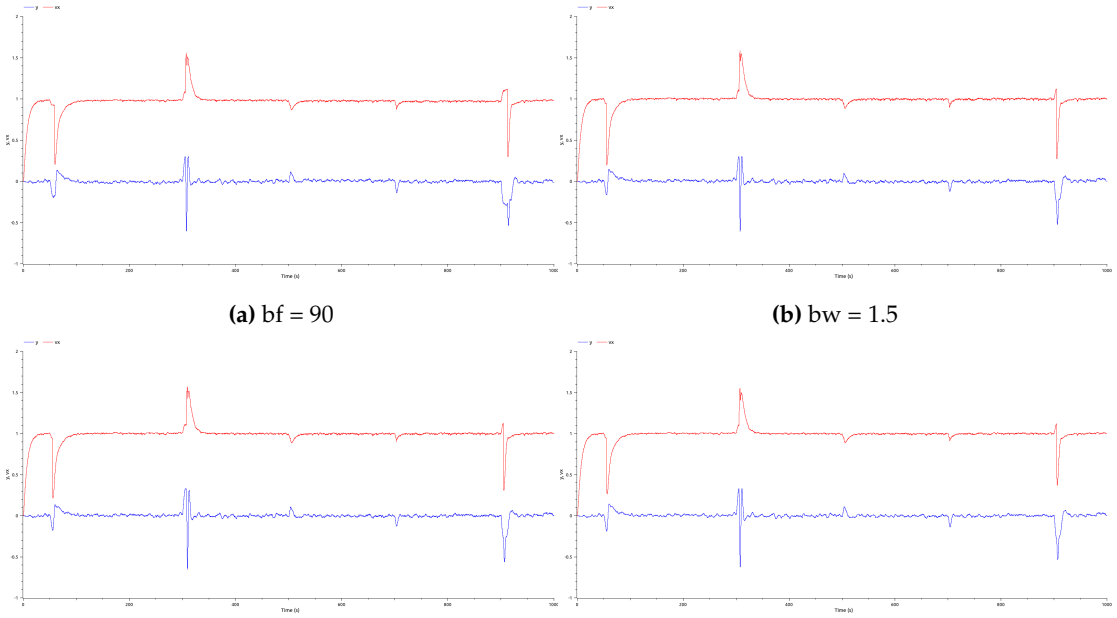
(a) **(b)**

Figure 5.2: Las figuras 5.2a y 5.2b muestran la simulación utilizando el modelo Modelica®del vehículo y el sistema de control en C, corriendo en tiempo de simulación y en tiempo real respectivamente.

Como se puede apreciar en la *Fig.* 5.2, y su análoga en MATLAB®ya analizada en *Fig.* 4.3 el comportamiento de este esquema de detección y reconfiguración funciona correctamente aún ante pequeños retrasos en la respuesta al sistema, como sucederían en la implementación del mismo como control digital a un sistema físico.

5.5.1 Simulación con defectos de modelado

Ahora bien, a la hora de correr el código en un sistema real, como ser el robot desmalezador presente en la institución huésped, ciertos defectos de modelado sobre el modelo en que se basa el sistema de control pueden comenzar a afectar la estabilidad del sistema. Estos defectos pueden deberse a estimaciones deficientes, parámetros descartados y problemas físicos, entre otros. Es por esto que nos propusimos realizar simulaciones con modelos del robot que difieren parcialmente del utilizado al calcular el sistema de control, lo cual nos dará cierta seguridad al utilizarlo en el sistema físico, que puede no coincidir con el modelo por las razones antes mencionadas.



(a) bf = 90 **(b)** bw = 1.5

(c) bf = 90 y bw = 1.5 **(d)** bf=90, bw = 1.5, Lr=0.9, Lf=1.1 y m=250

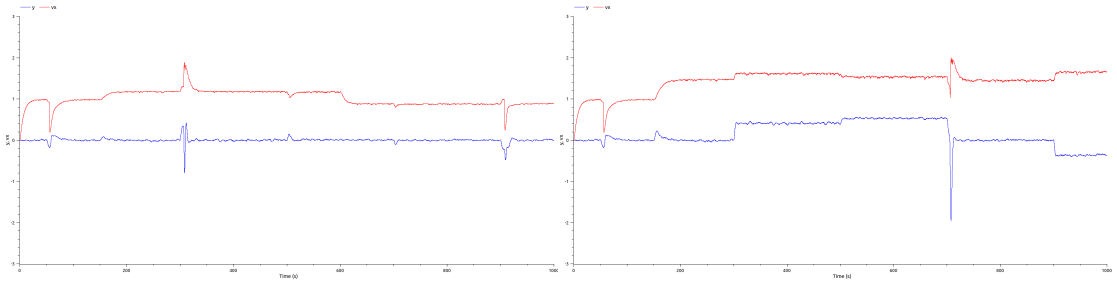
Figure 5.3: Se muestran 4 casos de simulación con método HiL con diferencias de parámetros en el modelo del vehículo y los mismos objetivos de control presentados anteriormente.

La *Fig.* 5.3 muestra resultados de cuatro simulaciones con modelos del vehículo que difieren del original en valores de cieros parámetros, pero cuyo sistema de control y las fallas que transcurren son idénticas que las presentadas en la sección anterior.
En el primer caso alteramos el coeficiente de roce de contacto bf en $+10N/(m/s)$ de las ruedas con el piso y como muestra la *Fig.* 5.3a los resultados son similares al original.
Luego alteramos el coeficiente de fricción bw de las ruedas con su eje en $-0.2(N*m)/(rad/s)$, este suele ser un parámetro difícil de predecir y modelar, sin embargo como vemos en la *Fig.* 5.3b los resultados son positivos al igual que antes.
En la *Fig.* 5.3c mostramos ambos cambios en simultáneo, y aunque el comportamiento comienza a ser menos estable, el sistema sigue detectando fallas y siguiendo el objetivo de forma correcta.
Y finalmente en la *Fig.* 5.3d mostramos un caso extremo, aunque no imposible, en que se modifican además de lo antes mencionado, la distancia del centro de masa a los ejes (en -0.1 y +0.1 metros para Lr y Lf respectivamente) y la masa m que acarrea el vehículo (en +50kg) (situación que puede darse al agregar elementos encima del mismo). Los resultados son positivos de igual forma que antes.

5.5.2 Simulación con objetivo de referencia cambiante

Por último mostramos un ejemplo de simulación del robot corriendo el esquema de control con un objetivo de referencia que varía en el tiempo. Se deja en claro que el objetivo de referencia no puede variar demasiado del utilizado para la linealización del

sistema ya que se pierden propiedades deseables, es por esto que decidimos mostrar resultados para variaciones de $< 0.2\text{m/s}$, y para resaltar esto último lo comparamos con una variación de 0.5m/s.



(a) **(b)**

Figure 5.4: Las figuras 5.4a y 5.4b presentan los resultados para la mismas situaciones de falla que en las simulaciones anteriores con dos cambios de objetivo de control. Para la figura 5.4a se realizan dos cambios: en el segundo 150 se aumenta la velocidad en 0.2m/s y en el segundo 600 se decrementa en 0.3m/s. En la figura 5.4b se realiza un cambio de 0.5m/s en el segundo 150.

Como podemos ver en la *Fig.* 5.4, un mayor análisis de la estabilidad del sistema se necesita para utilizar un esquema con objetivo de referencia cambiante. Sin embargo para valores de referencia cercanos a los antes analizados la estabilidad del sistema se mantiene, como vemos en la *Fig.* 5.4a. Ahora bien al aumentar en gran medida la velocidad del vehículo comenzamos a tener problemas de estabilidad luego de la presencia de fallas, como muestra la *Fig.* 5.4b.

CAPÍTULO 6

Conclusiones y trabajo futuro

En este trabajo presentamos un esquema de control basado en cotas finales probabilísticas para tiempo discreto junto a los enunciados que respaldan su correcto funcionamiento. El mismo presenta el marco teórico para proveer el soporte adecuado al implementar el esquema de control en un sistema digital.
La subsecuente implementación y simulación, y la tolerancia a discrepancias con el modelo que aquí se demuestra, sustentan las intenciones del grupo de trabajo de utilizar dicho esquema para el funcionamiento de un vehículo autónomo que se lleva a cabo en la institución huésped. Las pruebas sobre el modelo linealizado de un robot cuya dinámica es no-lineal y los resultados positivos obtenidos en las pruebas de estilo Hardware-in-the-Loop proveen el sustento necesario para asegurar el correcto funcionamiento de la implementación como sistema de control digital del mismo.
En resumen, el esquema es lo suficientemente robusto para tolerar discrepancias de modelado, ruido externo y demoras en el procesamiento de la respuesta de control.
Como continuación de esta tesina se propone estudiar en forma cuantitativa y cualitativa los efectos de retrasos en el cómputo de control sobre la estabilidad del sistema y el cómputo de fallas, lo que permitiría entender los requerimientos necesarios a la hora de elegir el sistema de cómputo y diseñar la conexión con el sistema. Por supuesto se plantea implementar el esquema de control para el robot antes mencionado y analizar su comportamiento al correr fuera de una simulación, es decir, como control del vehículo físico. Por último se sugiere extender tanto el marco teórico como la implementación a la detección de fallas en sensores con la misma metodología.

APPENDIX A

Modelo de la Dinámica del Robot Desmalezador

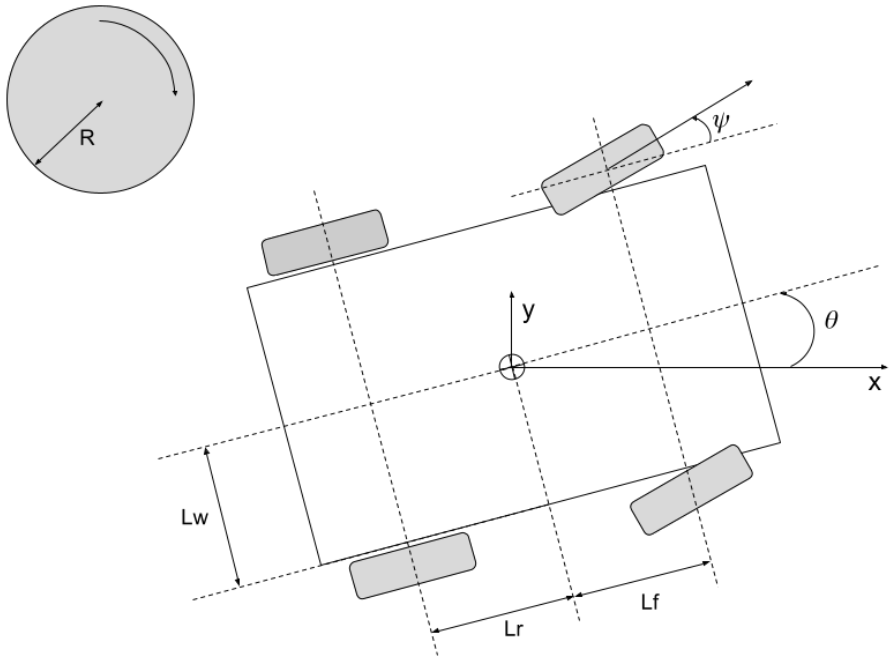


Figure A.1: Estructura de cinemática de referencia para el robot desmalezador.

Las ecuaciones que modelan la dinámica del robot desmalezador se presenta en Modelica®como:

cifacar.mo

```
model cifacar
  parameter Real Lr = 1, Lf = 1, Lw = 0.6, R = 0.3;
  parameter Real m = 200, J = 50;
  parameter Real Jw = 5, bw = 1.7;
  parameter Real bf = 100;
  Real x, y, th, vx, vy, w, d;
```

```
  Real Fxlr, Fxrr, Fxlf, Fxrf, Fylr, Fyrr, Fylf, Fyrf;
  Real taulr, taurr, taulf, taurf;
  Real Tlr, Trr, Tlf, Trf;
  Real wlr, wrr, wlf, wrf;
  Real vxlr, vylr, vxrr, vyrr, vxlf, vylf, vxrf, vyrf;
  Real psi;
  Real v;
equation
  vxlr = vx + (Lr * sin(th) - Lw * cos(th)) * w;
  vxrr = vx + (Lr * sin(th) + Lw * cos(th)) * w;
  vxlf = vx + ((-Lf * sin(th)) - Lw * cos(th)) * w;
  vxrf = vx + ((-Lf * sin(th)) + Lw * cos(th)) * w;
  vylr = vy + ((-Lr * cos(th)) - Lw * sin(th)) * w;
  vyrr = vy + ((-Lr * cos(th)) + Lw * sin(th)) * w;
  vylf = vy + (Lf * cos(th) - Lw * sin(th)) * w;
  vyrf = vy + (Lf * cos(th) + Lw * sin(th)) * w;
  Fxlr = bf * (R * wlr * cos(th) - vxlr);
  Fylr = bf * (R * wlr * sin(th) - vylr);
  Fxrr = bf * (R * wrr * cos(th) - vxrr);
  Fyrr = bf * (R * wrr * sin(th) - vyrr);
  Fxlf = bf * (R * wlf * cos(th + psi) - vxlf);
  Fylf = bf * (R * wlf * sin(th + psi) - vylf);
  Fxrf = bf * (R * wrf * cos(th + psi) - vxrf);
  Fyrf = bf * (R * wrf * sin(th + psi) - vyrf);
  taulr = Fxlr * sin(th) * Lr - Fxlr * cos(th) * Lw - Fylr * cos(th) * Lr - Fylr * sin(th) * Lw;
  taurr = Fxrr * sin(th) * Lr + Fxrr * cos(th) * Lw - Fyrr * cos(th) * Lr + Fyrr * sin(th) * Lw;
  taulf = (-Fxlf * sin(th) * Lf) - Fxlf * cos(th) * Lw + Fylf * cos(th) * Lf - Fylf * sin(th) * Lw;
  taurf = (-Fxrf * sin(th) * Lf) + Fxrf * cos(th) * Lw + Fyrf * cos(th) * Lf + Fylf * sin(th) * Lw;
  der(x) = vx;
  der(y) = vy;
  der(th) = w;
  der(vx) = (Fxlr + Fxrr + Fxlf + Fxrf) / m;
  der(vy) = (Fylr + Fyrr + Fylf + Fyrf) / m;
  der(w) = (taulr + taurr + taulf + taurf) / J;
  der(wlr) = (Tlr - Fxlr * R * cos(th) - Fylr * R * sin(th) - bw * wlr) / Jw;
  der(wrr) = (Trr - Fxrr * R * cos(th) - Fyrr * R * sin(th) - bw * wrr) / Jw;
  der(wlf) = (Tlf - Fxlf * R * cos(th + psi) - Fylf * R * sin(th + psi) - bw * wlf) / Jw;
  der(wrf) = (Trf - Fxrf * R * cos(th + psi) - Fyrf * R * sin(th + psi) - bw * wrf) / Jw;
  v = (wlr + wrr + wlf + wrf) * R / 4;
  der(d) = abs(v);
end cifacar;
```

Donde

x, y := posición en x e y respectivamente,

th := dirección/rotación,

vx, vy := velocidad en x e y respectivamente,

w := velocidad rotacional,

Fx, Fy := fuerzas de las ruedas en x e y respectivamente,

T := torques de entrada a cada rueda,

tau := torque transmitido a las ruedas,

psi := ángulo de dirección de entrada,

d := distancia recorrida por el robot,

v := velocidad del robot,

$*lr$, $*rr$:= sufijo indicando ruedas izquierda y derecha trasera respectivamente,

$*lf$, $*rf$:= sufijo indicando ruedas izquierda y derecha delantera respectivamente,

y los parámetros fijos:

Lr := distancia del centro de masa a eje trasero,

Lf := distancia del centro de masa a eje delantero,

Lw := distancia desde el centro de cada eje hacia las ruedas de dicho eje,

R := radio de las ruedas,

m := masa del vehículo,

J := momento de inercia del vehículo respecto del centro de masa,

Jw := momento de inercia de cada rueda respecto a su eje,

bw := coeficiente de fricción en el eje de cada rueda,

bf := coeficiente de fricción entre el punto de contacto de la rueda y el piso.

APPENDIX B

Implementación en C

Se muestra parte del código de implementación, obviando cabeceras, ciertas funciones de construcción y destrucción de estructuras, y algunos comentarios. Donde se elimine código se mostrará el símbolo `[...]`, con un comentario opcional explicando lo suprimido, el cual no forma parte del código real en ninguno de los módulos aquí presentados.

control_agrobot.h

```
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_blas.h>
#include <math.h>            //fabs

/** CONSTANT DEFINITIONS & MACROS**/
#define _N 9
#define _M 5
#define _P 6

#define TS 0.1
#define NUMFAULTS 11
#define SMA 10
#define SMAINV 0.1

[...]

/** TYPE & STRUCTURE DECLARATIONS **/
typedef gsl_matrix Mat;
typedef gsl_vector Vec;
typedef gsl_vector_view Vec_v;
typedef gsl_matrix_view Mat_v;


typedef struct StateSpace_ {
    Mat *A; //(n*n) state matrix
    unsigned int n;
    Mat *B; //(n*m) input matrix
    unsigned int m;
    Mat *C; //(p*n) output matrix
    unsigned int p;
} *StateSpace;

typedef struct Observer_ {
    Mat **P; //(m*m*numfaults) actuator fault matrix
    Mat *L;  //(n*p) observer matrix
    Vec *xobs_; //(n*1) old state
} *Observer;

typedef struct Reference_ {
    Mat **P;  //(m*m*numfaults) actuator fault matrix
    Mat **Kr; //(m*(n+m)*numfaults) reference control matrix
```

```
    Mat **Cr; //(m*n*numfaults) reference output matrix
    Vec *dti; //(m*1) discrete time integrator state
    Vec *xref_; //(n*1) old xref
    Vec *yref_; //(m*1) old yref
    Vec *uref_; //(m*1) debug uref
} *Reference;

typedef struct Diagnosis_ {
    Mat **A3s;  //((n+2m)*(n+2m)*numfaults)
    Mat ***zbs; //(2n*m*numfaults*numfaults)
    Vec ***zub; //(2n*numfaults*numfaults)
    Vec **sma;  //(numfaults*SMA) simple moving average
    int sma_i;  //(1) simple moving average counter
} *Diagnosis;

typedef struct Controller_ {
    Mat **K; //(m*n*numfaults) control matrix array
} *Controller;

typedef struct FullState_ {
    StateSpace sp;
    Observer ob;
    Reference rf;
    Diagnosis dg;
    Controller ct;
    int fault;      //current fault
    Vec *u;         //last input to system
    Vec *zhat;

} *FullState;
```

control_agrobot.c

```
[...]

void control_agrobot(Vec *y, Vec *yr, FullState st, Vec *u)
{
    Vec *xobs = gsl_vector_calloc(st->sp->n);
    Vec *xref = gsl_vector_calloc(st->sp->n);
    Vec *uref = gsl_vector_calloc(st->sp->m);
    Vec *yref = gsl_vector_calloc(st->sp->m);

    observer(y, st->u, st->ob, st->sp, st->fault, xobs);
    reference(yr, st->rf, st->sp, st->fault, xref, uref, yref);
    gsl_vector_memcpy(st->zhat, xobs);
    gsl_vector_sub(st->zhat, xref);
    diagnosis(st->zhat, yr, st->dg, st->sp, st->fault, &st->fault);
    controller(uref, st->zhat, st->ct, st->fault, st->u);
    gsl_vector_memcpy(u, st->u);

    [...]
}

[... constructores y destructores de estructuras]

void observer(Vec *yp, Vec *u, Observer ob, StateSpace sp, int fault, Vec *xobs)
{
    Vec *tmp1 = gsl_vector_alloc(sp->m); // (m*1) zero vector
    Vec *tmp2 = gsl_vector_alloc(sp->p); // (p*1) zero vector

    //Pj (m*m) fault matrix for the jth fault
    Mat *Pj = ob->P[fault];
    //tmp = Pj * u
    gsl_blas_dgemv(CblasNoTrans, 1.0, Pj, u, 0.0, tmp1);
    //xobs = B * (Pj * u)
    gsl_blas_dgemv(CblasNoTrans, 1.0, sp->B, tmp1, 0.0, xobs);

    //xobs = A*xobs_ + (B * (Pj * u))
    gsl_blas_dgemv(CblasNoTrans, 1.0, sp->A, ob->xobs_, 1.0, xobs);

    //tmp = C*xobs_
    gsl_blas_dgemv(CblasNoTrans, 1.0, sp->C, ob->xobs_, 0.0, tmp2);
    //tmp = (C*xobs_) - yp
    gsl_vector_sub(tmp2,yp);
```

```
    //xobs = -L*(C*xobs_ - yp) + (A*xobs_ + (B * (Pj * u)))
    gsl_blas_dgemv(CblasNoTrans, -1.0, ob->L, tmp2, 1.0, xobs);

    gsl_vector_memcpy(ob->xobs_, xobs);

    gsl_vector_free(tmp1);
    gsl_vector_free(tmp2);
}


void reference(Vec *yr, Reference rf, StateSpace sp, int fault, Vec *xref, Vec *uref, Vec *yref)
{
    Vec *tmp1 = gsl_vector_calloc(sp->n+sp->m); // ((n+m)*1) zero vector
    Vec *tmp2 = gsl_vector_calloc(sp->m); // (m*1) zero vector

    Mat *Krj = rf->Kr[fault];
    Mat *Pj = rf->P[fault];
    Mat *Crj = rf->Cr[fault];

    // tmp2 = yr - yref_
    gsl_vector_memcpy(tmp2, yr);
    gsl_vector_sub(tmp2, rf->yref_);

    // discrete time integration
    Vec_v low_tmp1 = gsl_vector_subvector(tmp1, sp->n, sp->m);
    gsl_vector_memcpy(&low_tmp1.vector, rf->dti);
    gsl_vector_scale(tmp2, TS);
    gsl_vector_add(rf->dti, tmp2);

    // tmp1 = [xref_ , yr-yref_]
    Vec_v high_tmp1 = gsl_vector_subvector(tmp1, 0, sp->n);
    gsl_vector_memcpy(&high_tmp1.vector, rf->xref_);

    // uref = Krj*[x,y]
    gsl_blas_dgemv(CblasNoTrans, 1.0, Krj, tmp1, 0.0, uref);

    // tmp = P*uref
    gsl_blas_dgemv(CblasNoTrans, 1.0, Pj, uref, 0.0, tmp2);
    // tmp = B*(P*uref)
    gsl_blas_dgemv(CblasNoTrans, 1.0, sp->B, tmp2, 0.0, xref);

    // xref = A*xref_ + (B*P*uref)
    gsl_blas_dgemv(CblasNoTrans, 1.0, sp->A, rf->xref_, 1.0, xref);

    // yref = Crj * xref
    gsl_blas_dgemv(CblasNoTrans, 1.0, Crj, xref, 0.0, yref);

    // Copy states to memory
    gsl_vector_memcpy(rf->xref_,xref);
    gsl_vector_memcpy(rf->yref_,yref);
    gsl_vector_memcpy(rf->uref_,uref);

    gsl_vector_free(tmp1);
    gsl_vector_free(tmp2);
}


void diagnosis(Vec *zhat, Vec *yr, Diagnosis dg, StateSpace sp, int fault, int *faultdet)
{
    Vec *tmp1 = gsl_vector_calloc(sp->n+2*sp->m); //((n+2m)*1) zero vector
    Vec *tmp2 = gsl_vector_alloc(2*sp->n);        //(2n*1) vector
    Vec *tmp3 = gsl_vector_calloc(NUMFAULTS);     //(numfaults*1) zero vector
    Vec *tmp4 = gsl_vector_alloc(sp->n);          //(n*1) vector

    // z = A3s[j] * yref
    Mat_v A3sj = gsl_matrix_submatrix(dg->A3s[fault], 0, sp->n,
        sp->n+2*sp->m, sp->m);
    gsl_blas_dgemv(CblasNoTrans, 1.0, &A3sj.matrix, yr, 0.0, tmp1);

    for (int i = 0; i < NUMFAULTS; i++) {
        // zb = zbs * z
        Mat *zbsi = dg->zbs[i][fault];
        Vec_v tmp1m = gsl_vector_subvector(tmp1, sp->n+sp->m, sp->m);
        gsl_blas_dgemv(CblasNoTrans, 1.0, zbsi, &tmp1m.vector, 0.0, tmp2);
```

```
        //Calculate indicator
        // max(abs(zhat - zb) - zub)
        Vec_v zbn = gsl_vector_subvector(tmp2, 0, sp->n);
        gsl_vector_memcpy(tmp4, zhat);
        gsl_vector_sub(tmp4, &zbn.vector);
        gsl_vector_abs(tmp4);
        Vec_v zubn = gsl_vector_subvector(dg->zub[i][fault], 0, sp->n);
        gsl_vector_sub(tmp4, &zubn.vector);

        // indicator vector
        if (gsl_vector_max(tmp4) < 0.0) {
            gsl_vector_set(tmp3, i, 1.0);
        }
    }

    //Simple Moving Average
    // sma[i] = tmp3
    gsl_vector_memcpy(dg->sma[dg->sma_i], tmp3);
    dg->sma_i = (dg->sma_i + 1) % SMA;
    gsl_vector_set_zero(tmp3);
    // tmp3 = sum(sma)
    for (int k = 0; k < SMA; k++) {
        gsl_vector_add(tmp3, dg->sma[k]);
    }
    // tmp3 = sum(sma) / (1/SMA)
    gsl_vector_scale(tmp3, SMAINV);

    //Detect fault
    // new index if max != 0, else old fault
    size_t index = gsl_vector_max_index(tmp3);
    *faultdet = (gsl_vector_get(tmp3,index) == 0. ? fault : index);

    gsl_vector_free(tmp1);
    gsl_vector_free(tmp2);
    gsl_vector_free(tmp3);
    gsl_vector_free(tmp4);
}

void controller(Vec *uref, Vec *e_or, Controller ct, int fault, Vec *u)
{
    // Kj (m*n) control vector for the jth fault
    Mat *Kj = ct->K[fault];
    //u = uref
    gsl_vector_memcpy(u, uref);
    // u = K*(e_or) + uref
    gsl_blas_dgemv(CblasNoTrans, 1.0, Kj, e_or, 1.0, u);
}

[... funciones auxiliares]
```

APPENDIX C

Implementación en Modelica®

Se muestra parte del código de implementación. Donde se elimine código se mostrará el símbolo `[...]`, con un comentario opcional explicando lo suprimido, el cual no forma parte del código real en ninguno de los módulos aquí presentados.

RealTime.mo

```
[...]

class RealTime
  extends ExternalObject;

  function constructor
    output RealTime timekeeper;

    external "C" timekeeper = real_time_constructor_modelica() annotation(
      Library = "modelica_hil");
  end constructor;

  function destructor
    input RealTime timekeeper;

    external "C" real_time_destructor_modelica(timekeeper) annotation(
      Library = "modelica_hil");
  end destructor;
end RealTime;
```

SemFile.mo

```
[...]

class SemFile
extends ExternalObject;

  function constructor
    input String dir;
    output SemFile file;
    external "C" file = semfile_constructor(dir) annotation(Library="modelica_hil");
  end constructor;

  function destructor
    input SemFile file;
    external "C" semfile_destructor(file) annotation(Library="modelica_hil");
  end destructor;

end SemFile;
```

controled_car_noisy_rt.mo

```
[...]
```

```
model controled_car_noisy_rt
  cifacar car "Plant model";
  // Be sure to have the same parameters on the matlab matrix generation (car.mdl)
  parameter ControlState state = ControlState() "Used just to get P";
  parameter SemMem outputs = SemMem("/outputsFile");
  parameter SemMem control = SemMem("/controlFile");
  parameter RealTime timekeeper = RealTime() "synchronization for HiL";
  parameter Integer n = 9, m = 5, p = 6;
  parameter Real yr[m] = {0, 0, 1, 0, 0} "Reference tracking";
  Modelica.Blocks.Noise.BandLimitedWhiteNoise blwn(noisePower=1, samplePeriod=0.1);
  Modelica.Blocks.Sources.TimeTable tab(table
       =[0,0;50,0;50,3;300,3;300,0;500,0;500,2;700,2;700,5;900,5;900,1;1000,1]) "Pizzi et.al.";
  Real u[m] "Control input";
  Real y[p] "Plant output";
  Real P[m, m];
  Real G[m] = {0.1,0.1,0.1,0.1,0.05};
  Integer i;
  Real psir;
algorithm
  when sample(0, 0.01) then
    synchronize(timekeeper, time);
    y[1] := car.y;
    y[2] := car.th;
    y[3] := car.wlr;
    y[4] := car.wrr;
    y[5] := car.wlf;
    y[6] := car.wrf;
    semmem_write(outputs, {y[1], y[2], y[3], y[4], y[5], y[6], yr[1], yr[2], yr[3], yr[4], yr[5]});
    u := semmem_read(control, m);
    i := integer(tab.y);
    P := get_P(state, i);
    u := P * u;
    u := u + G * blwn.y;
    car.Tlr := u[1];
    car.Trr := u[2];
    car.Tlf := u[3];
    car.Trf := u[4];
    if u[5] > 0.7853981 then // > pi/4
      psir := 0.7853981;
    elseif u[5] < -0.7853981 then // < -pi/4
      psir := -0.7853981;
    else
      psir := u[5];
    end if;
    car.psi := psir;
  end when;
equation

  [...]
end controled_car_noisy_rt;
```

Bibliografía

[1] Noelia Pizzi, Ernesto Kofman, José A. De Doná, and Maria M. Seron. Actuator fault tolerant control based on probabilistic ultimate bounds. *ISA Transactions*, 84:20 – 30, 2019. ISSN 0019-0578. doi: https://doi.org/10.1016/j.isatra.2018.08.021. URL http://www.sciencedirect.com/science/article/pii/S0019057818303173.

[2] Karl Johan Astrom and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, USA, 2008. ISBN 0691135762.

[3] Franco Blanchini and Stefano Miani. *Set-Theoretic Methods in Control*. Birkhäuser Basel, 1st edition, 2007. ISBN 0817632557.

[4] E. Kofman, J. A. De Doná, and M. M. Seron. Probabilistic ultimate bounds and invariant sets for lti systems with gaussian disturbances. In *2011 Australian Control Conference*, pages 537–542, 2011.

[5] Ernesto Kofman, José A. De Doná, and Maria M. Seron. Probabilistic set invariance and ultimate boundedness. *Automatica*, 48(10):2670 – 2676, 2012. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2012.06.074. URL http://www.sciencedirect.com/science/article/pii/S0005109812003408.

[6] Mogens Blanke, Michel Kinnaert, Jan Lunze, and Marcel Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer Publishing Company, Incorporated, 3rd edition, 2015. ISBN 3662479427.

[7] Karl J. Åström. *Introduction to Stochastic Control Theory*. Mathematics in Science and Engineering. Elsevier, 1970. ISBN 9780120656509.

[8] Edson Coayla, Salah-Eldin Mohammed, and Paulo Ruffino. Hartman-grobman theorems along hyperbolic stationary trajectories. *Articles and Preprints*, 17, 02 2006. doi: 10.3934/dcds.2007.17.281.

[9] Jim Ledin. *Embedded Control Systems in C/C++*. CRC Press, Inc., USA, 2003. ISBN 1578201276.

[10] M. Galassi. Gnu scientific library reference manual, 2018. URL https://www.gnu.org/software/gsl/.

[11] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[12] Peter Fritzson and Vadim Engelson. Modelica — a unified object-oriented language for system modeling and simulation. In Eric Jul, editor, *ECOOP'98 — Object-Oriented Programming*, pages 67–90, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-69064-1.

[13] M. M. Seron, J. A. De Dona, and S. Olaru. Fault tolerant control allowing sensor healthy-to-faulty and faulty-to-healthy transitions. *IEEE Transactions on Automatic Control*, 57(7):1657–1669, 2012. doi: 10.1109/TAC.2011.2178716.

[14] E. Chow and A. Willsky. Analytical redundancy and the design of robust failure detection systems. *IEEE Transactions on Automatic Control*, 29(7):603–614, 1984. doi: 10.1109/TAC.1984.1103593.

[15] C. Ocampo-Martinez, J. A. De Doná, and M. M. Seron. Actuator fault-tolerant control based on set separation. *International Journal of Adaptive Control and Signal Processing*, 24(12):1070–1090, 2010. doi: https://doi.org/10.1002/acs.1181. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/acs.1181.

[16] Raymond H. Kwong. Control design for set point tracking, 2008. URL https://www.control.utoronto.ca/people/profs/kwong/ece410/2008/notes/chap5.pdf.