Control tolerante a fallas en tiempo discreto basado en conjuntos probabilísticos. Aplicación.

Nicolás Soncini*, Noelia Pizzi†, Ernesto Kofman*†,
*FCEIA - Universidad Nacional de Rosario
Email: soncininicolas@gmail.com
†CIFASIS - CONICET
Email: pizzi, kofman@cifasis-conicet.gov.ar

***Abstract*—En este trabajo se presenta un caso de aplicación de un esquema de control tolerante a fallas basado en cotas finales probabilísticas a un modelo de un robot móvil. Por un lado se realiza el diseño de los componentes del esquema mencionado y luego se analiza su correcto funcionamiento y robustez al aplicarlo sobre un modelo realista no lineal del vehículo tanto de forma *offline* como en tiempo real, incluyendo condiciones de incertidumbre paramétrica, ruido y retardos temporales.**

***Index Terms*—control tolerante a fallas, conjuntos probabilísticos, simulación en tiempo real, software-in-the-loop**

I. Introducción

Los sistemas de control tolerante a fallas tienen por objeto garantizar el correcto funcionamiento del sistema aún cuando éste presente cambios inesperados en su comportamiento. Muchos esquemas logran esto diagnosticando de forma automática la aparición de fallas y reconfigurando el sistema de control para que se adecúe a la nueva situación. El diagnóstico se suele realizar comparando las trayectorias observadas con las esperadas según las posibles situaciones de fallas factibles, produciendo una señal de residuo cuyo valor indica la presencia de una falla determinada.

En la primera parte de este trabajo [1] se presentó una estrategia novedosa de control tolerante a fallas de actuadores en tiempo discreto, en la cual se calcula la señal de residuo como la diferencia entre el estado de un observador de la planta y el estado de una planta de referencia. Para este residuo, se calculan conjuntos (determinados por cotas finales probabilísticas [2], o PUB por *Probabilistic Ultimate Bounds*) que dependen de la situación de falla para la que está configurado actualmente el sistema y de las distintas fallas que podrían ocurrir. Luego, el diagnóstico de las fallas se realiza determinando en cuál de estos conjunto el residuo permanece más frecuentemente y una vez hecho el diagnóstico, el sistema completo se reconfigura para la nueva situación detectada.

En esta segunda parte del trabajo aplicaremos la estrategia mencionada al modelo de un robot móvil con el objetivo de ilustrar la metodología, verificar su robustez y validar su viabilidad.

Si bien el diseño se realiza sobre un sistema linealizado (dado que la estrategia es lineal), en las simulaciones realizadas se utiliza como planta un modelo no lineal complejo que incluye diversos fenómenos incluido el deslizamiento de las ruedas del vehículo. Como análisis adicional de robustez se estudia también el desempeño del esquema ante variaciones de parámetros.

Finalmente, como etapa previa al uso sobre una plataforma robótica real, el esquema de control se implementa sobre una simulación en tiempo real del modelo, lo que permite una mejor validación de la viabilidad del uso de la estrategia en la práctica [3], [4].

II. Estrategia de Control Tolerante a Fallas

La Figura 1 muestra el esquema de la estrategia de control tolerante a fallas presentado en la primera parte de este trabajo [1]. El mismo se compone de la planta o sistema que se desea controlar, un observador que estima el estado de la planta 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 diagnóstico de fallas que calcula la situación de falla detectada y la transmite a los otros componentes del esquema para su subsecuente reconfiguración.

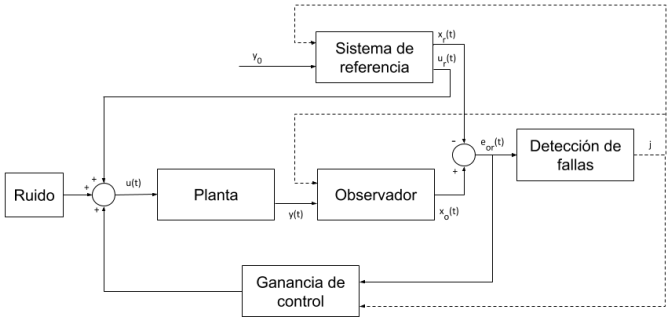


Fig. 1. Esquema de control tolerante a fallas propuesto en [1].

El sistema de diagnóstico de fallas en cada instante determina si la señal de residuo $e_{or}(t)$ (calculada como la diferencia entre el estado del sistema de referencia y el estado del observador) se encuentra dentro de ciertos conjuntos $S_{or}^{i,j}$ calculados a partir de cotas finales probabilísticas. En el trabajo citado se demuestra que si el sistema está configurado para la falla j y ocurre la falla i, tras cierto tiempo el residuo

va a tener una probabilidad mayor que p (el parámetro de los PUBs, típicamente un valor cercano a 1) de estar dentro del conjunto $S_{or}^{i,j}$

A partir de esto, el diagnóstico se realiza determinando en qué conjunto $S_{or}^{i,j}$ el residuo $e_{or}(t)$ se encuentra más frecuentemente. Para esto, en cada instante de tiempo se calcula la función indicatriz del residuo en cada conjunto $S_{or}^{i,j}$ (que vale 1 si $e_{or}(t)$ está en dicho conjunto y 0 en otro caso) y se calcula la media móvil de las indicatrices de cada conjunto.

De esta manera, el conjunto $S_{or}^{i,j}$ cuya indicatriz tenga la mayor media móvil será el que fue visitado más frecuentemente por el residuo y el sistema de diagnóstico determinará que ocurre la falla i correspondiente. El Teorema 1 de [1] muestra que si los PUBs son disjuntos, agrandando la ventana de la media móvil puede lograrse que la probabilidad de que el diagnóstico sea incorrecto se torne arbitrariamente pequeña.

III. Diseño de la Estrategia

A. Modelo del Vehículo

La Figura 2 muestra el esquema del robot sobre el cual diseñaremos un control de trayectorias tolerante a fallas. El mismo consiste en un vehículo que se desplaza en un plano mediante 4 ruedas unidas a un chasis en común, cada una con rotación y motor independientes. Adicionalmente, las ruedas delanteras cuentan con un sistema de dirección que responde a una geometría de Ackermann.

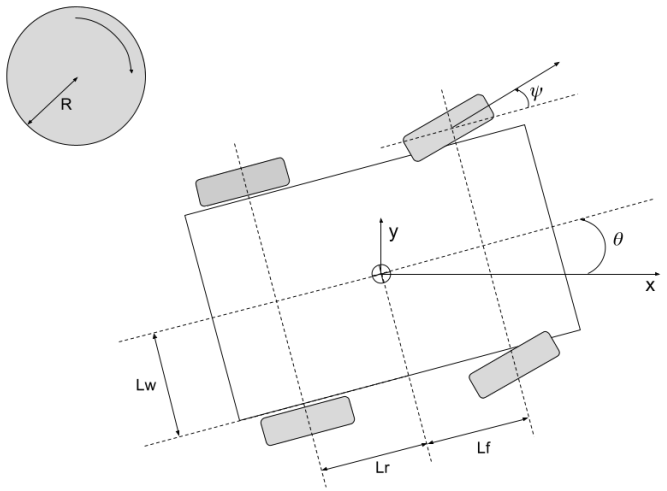


Fig. 2. Representación gráfica de la cinemática del "Robot Desmalezador"

Cada una de las 4 ruedas se modela teniendo en cuenta fricción contra el piso y desplazamiento los que nos permitirá considerar mas adelante la falla parcial o total de las ruedas (durante las cuales la o las ruedas afectadas deslizan).

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:

$$\mathbf{x}(t) = \begin{bmatrix} y & \theta & vx & vy & \omega & wlr & wrr & wlf & wrf \end{bmatrix}^T.$$

El objetivo de control será el de conducir el robot a velocidad constante por una línea recta que consideraremos alineada con el eje x. De esta forma, ignoraremos aquí el estado de la posición en el eje x ya que su valor no influirá en el control.

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 la dirección del eje delantero:$\mathbf{u}(t) = \begin{bmatrix} Tlr & Trr & Tlf & Trf & \psi \end{bmatrix}^T$ y permite medir 6 salidas, que se corresponden a la distancia al eje y, el ángulo θ del vehículo respecto a dicho eje y las velocidades de las cuatro ruedas.

B. Linealización y Discretización del Modelo

Dado que la estrategia depende desarrollada en la primera parte de este trabajo está desarrollada para modelos lineales de tiempo discreto, el primer paso del diseño consiste en linealizar y discretizar el modelo.

Calculando entonces la matriz jacobiana del sistema en torno al punto de operación correspondiente a una velocidad constante en el eje x $vx = 1$ y el resto de las variables calculadas de manera acorde, y suponiendo un período de muestreo $T_s = 0.1$ segundos y usando el método de Forward Euler para la discretización se obtiene un sistema lineal y estacionario de tiempo discreto

$$\begin{aligned} \mathbf{x}(t+1) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t+1) &= C\mathbf{x}(t), \end{aligned},$$

donde las matrices A, B y C toman los valores:

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

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

C. Perturbaciones y Fallas

El modelo con perturbaciones y fallas tendrá la forma

$$\begin{aligned} \mathbf{x}(t+1) &= A\mathbf{x}(t) + BP\mathbf{u}(t) + G\mathbf{w}(t) \\ \mathbf{y}(t+1) &= C\mathbf{x}(t), \end{aligned},$$

En este caso, $\mathbf{w}(t)$ será una señal de ruido blanco gaussiano con una matriz de covarianza incremental $\Sigma_w = 2$, el cual ingresa al sistema multiplicado por la matriz:

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

Las fallas de los actuadores estarán representados por la matriz P, que podrá tomar distintos valores según la situación de falla considerada. En la situación sana será $P = P^0 = I$. Las

restantes situaciones se corresponden a fallas en los motores que impulsan las ruedas, con la restricción de que a lo sumo 2 motores puedan fallar simultáneamente. De esta manera, habrá 11 situaciones de falla posibles: una correspondiente a la planta sin fallas, 4 correspondientes a un único motor fallando y las 6 restantes a los distintos pares de motores. Estas situaciones se representan por las siguientes matrices P^i:

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

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

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

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

D. Sistema de referencia

El sistema de referencia consiste en un modelo ideal de la planta

$$\begin{aligned} \mathbf{x_r}(t+1) &= A\mathbf{x}(t) + BP^j\mathbf{u_r}(t) \\ \mathbf{y_r}(t+1) &= C_r\mathbf{x}(t), \end{aligned},$$

con un controlador que garantiza que la salida de este sistema $\mathbf{y_r}(t)$ siga exponencialmente una señal de referencia $\mathbf{y_0}$. Dado que lo que buscamos es que el robot se mueva paralelo al eje x de coordenadas, esta señal de referencia, en nuestro caso, se corresponderá a una posición en el eje y y una velocidad constante en el eje x

Para garantizar el seguimiento diseñamos un regulador lineal cuadrático con término integral para cada posible situación de falla representada por P^j. Este controlador produce las señales de entrada de referencia $\mathbf{u}_r(t)$ que garantizan que la salida del modelo ideal de la planta de referencia siga a la señal de referencia deseada.

El resultado del diseño es una matriz Kr_j para cada situación de falla $j = 0, ..., 10$, con forma $Kr_j = [K_{Pj}\ K_I]^T$, que conforma la ley de control $\mathbf{u_r}(t) = -Kr_j\left[\mathbf{x_r}(t), \mathbf{x}_I(t)\right]^T$ donde $\mathbf{x_I}$ es la integral discreta del error calculada como $\mathbf{x_I}(t+1) = \mathbf{x_I}(t) + T_s(\mathbf{y_0} - \mathbf{y_r}(t))$).

La entrada y el estado de referencia $\mathbf{u_r}(t)$ y $\mathbf{x_r}(t)$ que se calculan en este sistema son los utilizados para calcular la entrada de la planta real y para diagnosticar las fallas como se muestra en la Figura 1.

E. Observador

Para estimar el estado de la planta se diseña un observador de Luenberger con matriz L_j, siendo j la situación de falla diagnosticada. El diseño de L_j para cada situación de falla se realizó de manera que los autovalores que introduce el observador sean 10 veces más rápidos que los correspondientes al sistema en lazo cerrado.

F. Control por Realimentación

La ganancia de control de la planta K_j se diseñó mediante un regulador lineal cuadrático (LQR) discreto para cada situación de falla posible $j = 0, ..., 10$. Los matrices P y Q del LQR se eligieron de manera de penalizar el error del estado 100 veces más que el error en las entradas de control.

Con esta elección pudo verificarse que se cumplía la condición de estabilidad respecto a la matriz de lazo cerrado

$$A_\ell^{i,j} = \begin{bmatrix} A + BP^jK_j & L_jC \\ B(P^i - Pj)K_j & A - L_jC \end{bmatrix} \quad (1)$$

para todo par $i, j \in \{0, ..., 10\}$ lo que garantiza la validez de los resultados teóricos.

G. Sistema de diagnóstico

Las distintas cotas finales probabilísticas del sistema de diagnóstico se calcularon utilizando una probabilidad $p = 0.99$, según el procedimiento explicado en la primera parte de este trabajo [1].

Dado que se consideraron 11 posibles situaciones de falla, para cada configuración se calcularon 11 conjuntos que resultaron disjuntos. La Figura 3 ilustra la proyección de estos conjuntos sobre un plano de dos variables de estado, en este caso wlr y wrr, para la situación libre de falla. Debe tenerse en cuenta que si aquí los conjuntos se superponen es porque se omiten 7 dimensiones correspondientes a las variables de estado restantes.

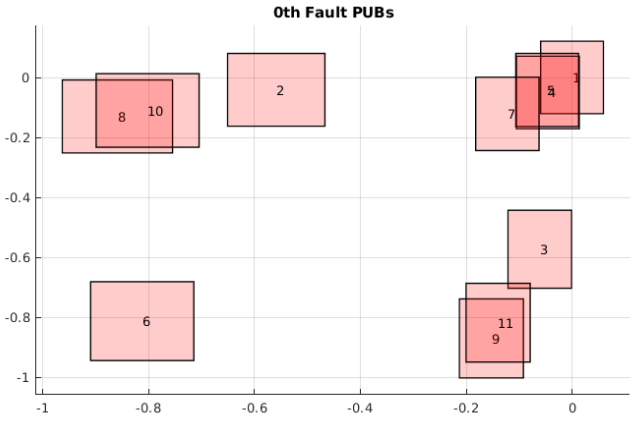


Fig. 3. Proyección de los conjuntos probabilísticos en las componentes correspondientes a las variables de estado wlr y wrr.

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 de longitud 10, que

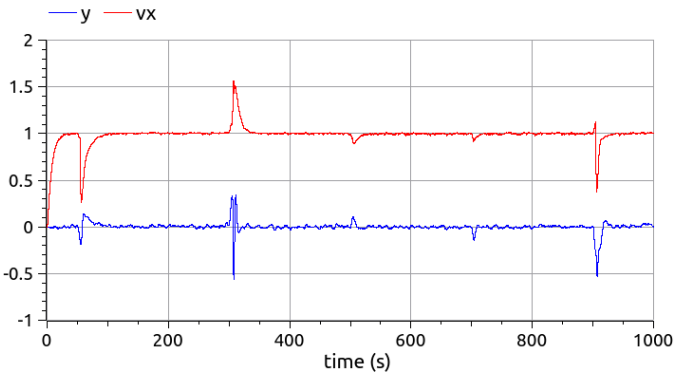
luego, tomando el índice de máximo valor de esta media, nos permite encontrar qué falla está en ocurrencia.

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.

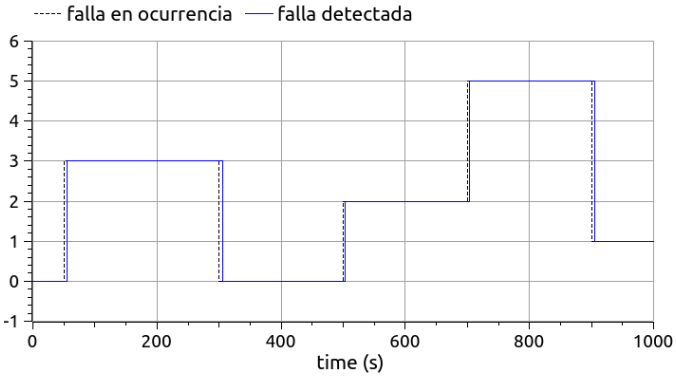
IV. Resultados offline

Para verificar el funcionamiento del esquema, simulamos el sistema completo tomando como planta el modelo no lineal original, y adoptando una referencia en la cual el vehículo se debe mover a una velocidad $vx = 1m/s$ paralelo al eje y.

En el escenario simulado, se introdujo la secuencia de fallas ilustradas por la Figura 4b, donde el índice de fallas se corresponde con los índices j de las matrices P^j antes detalladas. Puede verse en la misma figura que todas las fallas son diagnosticadas de manera correcta, siendo el tiempo máximo desde la ocurrencia de una falla hasta su detección de 5.4 segundos.



(a)



(b)

Fig. 4. La Figura 4a muestra la simulación utilizando el modelo Modelica®del vehículo y el sistema de control en C compilado como parte de la simulación, y 4b muestra la presencia y detección de fallas respectiva.

La Figura 4a además muestra que al realizar la reconfiguración el sistema sigue correctamente la referencia en todos los casos mientras que el control original que no considera la presencia de fallas, cuyos resultados de simulación se muestran en la Figura 5, muestra un error considerable.

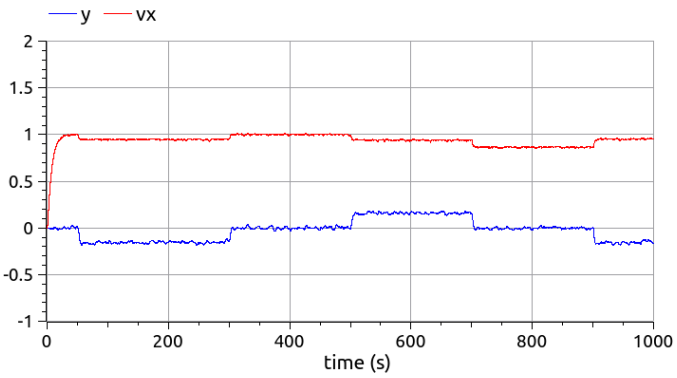


Fig. 5. Se muestra el resultado de simulación utilizando el modelo Modelica®del vehículo y el sistema de control en C compilado como parte de la simulación pero sin hacer uso del sistema de detección y reconfiguración ante fallas.

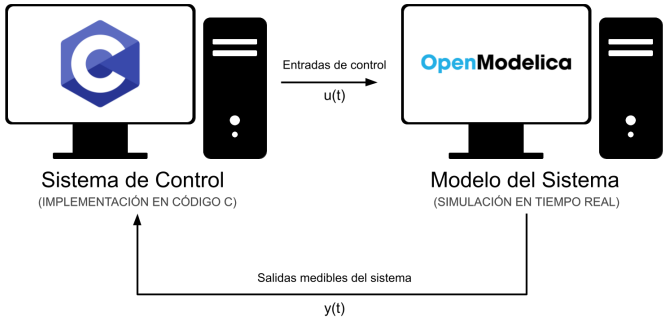


Fig. 6. Funcionamiento de la metodología Software-in-the-Loop

V. Implementación en tiempo real

Al llevar la implementación del esquema de control a un sistema de cómputo digital debemos garantizar que la misma funcionará correctamente aún ante la presencia de diferencias entre el modelo del sistema utilizado y el sistema físico, como pueden ser retrasos en la respuesta de control o defectos de modelado de la planta.

Para verificar esto hicimos uso de la metodología de *Software-in-the-Loop*, que nos proporciona una forma de evaluar el funcionamiento del software de control haciéndolo funcionar controlando una versión simulada de la planta corriendo en tiempo real.

Como muestra la Figura 6, el sistema de control implementado en forma de software es ejecutado bajo entradas provistas por una simulación del robot a controlar, y la simulación del robot toma sus entradas a partir de la salida del sistema de control. Ambos sistemas corren en tiempo real y de forma independiente uno del otro (en la forma que explicaremos mas adelante).

En nuestro caso el modelo del vehículo se desarrolló en el lenguaje Modelica®y la simulación en el entorno de OpenModelica [5]. Por su parte, el sistema de control es implementado en código C haciendo uso de las librerías GSL (GNU Scientific Library) [6] para todos los cálculos matriciales y/o vectoriales, ya 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) [7] para operaciones entre dichos tipos.

A. Sincronización con el tiempo real

Para asegurar que la simulación por computadora corre en tiempo real necesitamos algún método de sincronización entre el tiempo simulado y el reloj físico. Para nuestros propósitos, y suponiendo que la simulación de la planta siempre correrá más rápido que el tiempo real, basta con conectarla con una función externa que periódicamente mida el tiempo físico y pause la simulación hasta que este tiempo alcance al tiempo de simulación, por ejemplo mediante lo conocido como "espera activa" por parte del motor de simulación.

La implementación propuesta conecta a la simulación de la planta, implementada en el motor de simulación de OpenModelica con una función externa, implementada en C, mediante una estructura que lleva dos variables de tiempo, una que se utilizará para guardar el tiempo inicial y otra en la cual se guardará el tiempo real. Al comenzar la simulación, se inicializa la variable de tiempo inicial y en cada instante de sincronización de la simulación con el tiempo físico la función actúa esperando suficiente tiempo hasta que el tiempo de simulación se haya cumplido, es decir, hasta que se sincronice con el tiempo real, como muestra el Algoritmo 1.

Es imperioso recordar que esto funcionará siempre y cuando los cálculos correspondientes al algoritmo de simulación se realicen más rápido que lo indicado por el tiempo real.

```
repeat
  | cur_time ← clock_gettime();
until cur_time - initial_time < sim_time;
```

Algoritmo 1: Sincronización con el tiempo real

B. Comunicación entre planta y controlador

Para concretar la simulación en tiempo real necesitamos que la planta reciba las entradas de control provistas por el controlador, y por su parte el controlador debe poder leer las salidas medibles de la planta.

Para emular las condiciones reales de operación entre el sistema de control y la planta, la comunicación en ambas direcciones debe ser asíncrona. Esto es, la simulación no debe esperar a que el controlador termine de procesar la entrada para la planta, ni el controlador debe esperar al siguiente instante de simulación en que la planta provea sus salidas. Por lo tanto utilizamos un sistema a base de memoria compartida en la cual tanto las salidas de la planta como las entradas del controlador son escritas a un espacio de memoria accesible por ambos procesos, mediada por un semáforo que controla el acceso a dicha memoria para prevenir que uno de los procesos escriba en el mismo instante que el otro está leyendo.

C. Resultados en tiempo real

Se realizó entonces una simulación del sistema en tiempo real con el esquema de *Software-In-The-Loop* descripto bajo el mismo escenario de la simulación offline (es decir, emulando la misma secuencia de fallas). Los resultados pueden verse en la Figura 7.

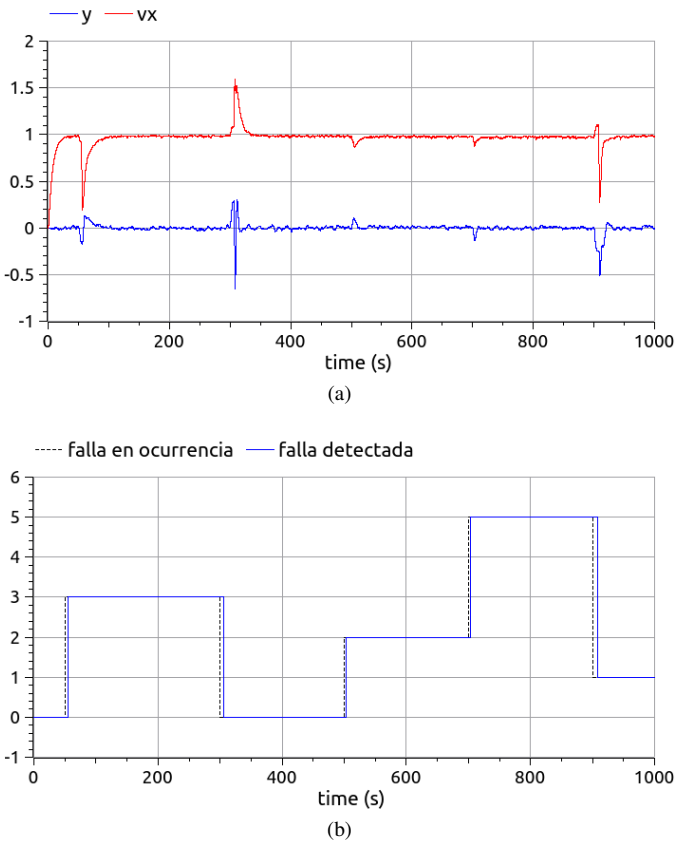


Fig. 7. La Figura 7a muestra la simulación utilizando el modelo Modelica®del vehículo y el sistema de control en C corriendo con el esquema de Software-in-the-Loop antes descrito, mientras que la Figura 7b muestra la presencia y detección de fallas respectiva.

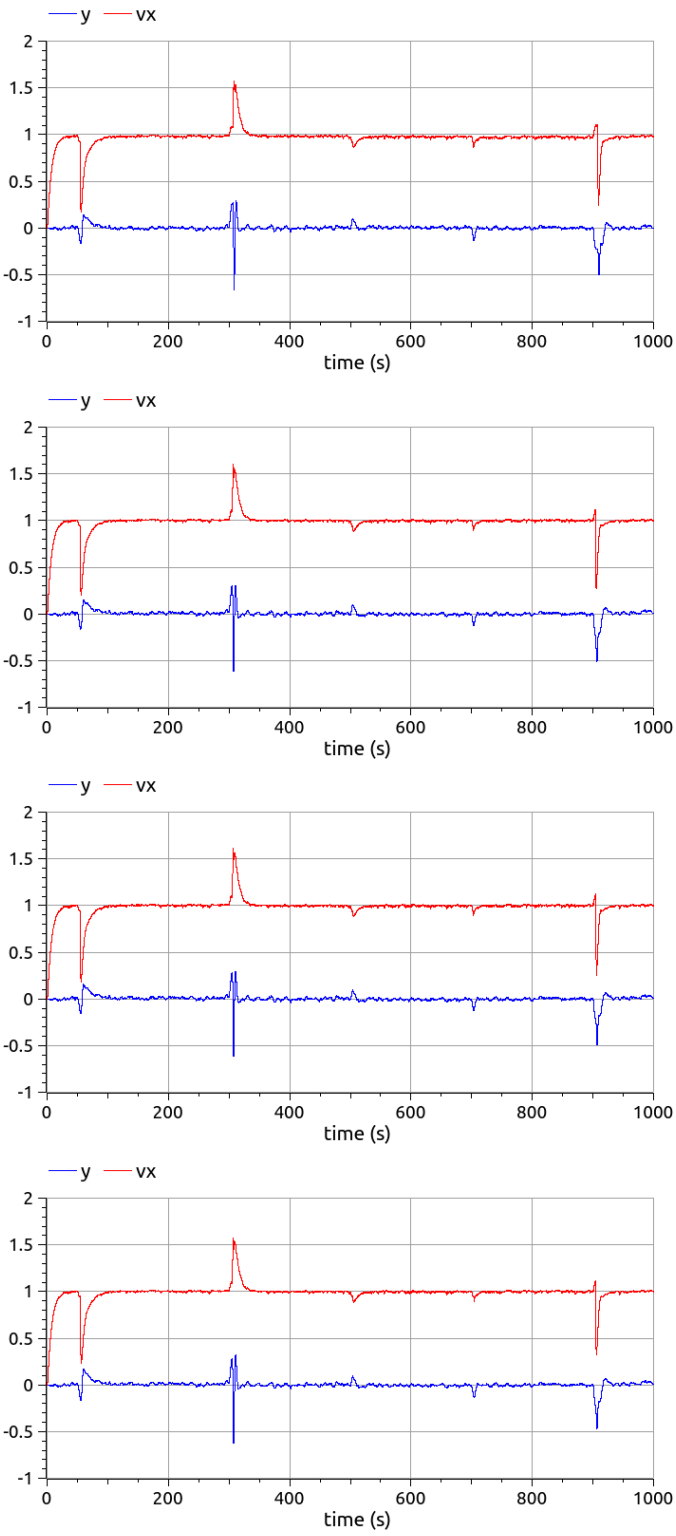
Como se puede apreciar, el comportamiento del esquema funcionando en tiempo real se aproxima al desempeño del esquema offline, si bien se torna un poco más lenta la detección. En particular en la simulación vemos que el tiempo máximo entre la ocurrencia y la detección de fallas es ahora de 8 segundos (un 50% más lento que lo obtenido offline) pero se mantiene dentro de valores tolerables.

D. Análisis de Robustez

Las simulaciones anteriores muestran que la estrategia es robusta, ya que el diseño se basó en una linealización y el esquema se simuló con un modelo no lineal más complejo e incluyendo los retardos que introduce la implementación en tiempo real.

Para ilustrar más las características robustas del enfoque, repetimos los experimentos de tiempo real introduciendo ahora cambios en distintos parámetros del modelo del robot manteniendo el diseño original de todo el esquema. Los resultados se

muestran en la Figura 8, donde se ensayaron cuatro escenarios distintos para la variación de parámetros.



(d) Ambos (a), (b) y $Lr - 10\%, Lf + 10\%, m + 25\%$

Fig. 8. Resultados de simulación en tiempo real con variaciones de parámetros respecto al modelo utilizado para el diseño.

En el primer caso alteramos el coeficiente de roce de contacto de las ruedas con el piso bf en un 10% de su valor original y como muestra la Figura 8a los resultados son similares al original. Por otro lado modificamos el coeficiente de fricción bw de las ruedas con su eje en un 11.7% de su valor original, este suele ser un parámetro difícil de predecir y modelar, sin embargo como vemos en la Figura 8b los resultados son de igual forma similares a los mostrados anteriormente.

Luego aplicamos ambos cambios en simultáneo y, aunque el comportamiento comienza a ser menos estable como muestra la Figura 8c, el sistema sigue detectando fallas y siguiendo el objetivo de forma correcta. Finalmente en la Figura 8d 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 un 10% de sus valores originales para Lr y Lf) y la masa m que acarrea el vehículo (en un 25% de su valor original), con comportamiento correcto y similar al mostrado en los casos anteriores.

VI. Conclusiones

Este trabajo presenta una aplicación de la estrategia de control tolerante a fallas propuesto en [1].

Los resultados de simulación obtenidos sobre un modelo cuya dinámica es no-lineal, junto a las técnicas de simulación *Software-on-the-Loop* muestran en primer lugar la robustez del esquema y proveen la validación necesaria para asegurar el correcto funcionamiento de la implementación del esquema de control corriendo sobre un sistema de cómputo digital como entrada a un sistema físico.

Finalmente, los resultados obtenidos frente a variaciones de los parámetros más inciertos del modelo ratifican la característica robusta del esquema.

Proponemos como trabajo futuro llevar la implementación de esta estrategia a un vehículo real, específicamente al prototipo de robot desmalezador del CIFASIS-CONICET [8] que se corresponde con el modelo utilizado en este trabajo.

Referencias

[1] N. Soncini, N. Pizzi, and E. Kofman, "Control tolerante a fallas en tiempo discreto basado en conjuntos probabilísticos. Teoría." 2021, enviado a RPIC 2021. Disponible en https://labdcc.fceia.unr.edu.ar/~nsoncini/control_tolerante_a_fallas_2021/.

[2] E. Kofman, J. A. D. Doná, and M. M. Seron, "Probabilistic set invariance and ultimate boundedness," *Automatica*, vol. 48, no. 10, pp. 2670 – 2676, 2012.

[3] S. Han, S.-G. Choi, and W.-H. Kwon, "Real-time software-in-the-loop simulation for control education," *International Journal of Innovative Computing, Information and Control*, vol. 7, 11 2011.

[4] X. Chen, M. Salem, T. Das, and X. Chen, "Real time software-in-the-loop simulation for control performance validation," *Simulation*, vol. 84, pp. 457–471, 08 2008.

[5] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons, 2014.

[6] M. Galassi, "Gnu scientific library reference manual," 2018. [Online]. Available: https://www.gnu.org/software/gsl/

[7] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry *et al.*, "An updated set of basic linear algebra subprograms (blas)," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, 2002.

[8] T. Pire, M. Mujica, J. Civera, and E. Kofman, "The rosario dataset: Multisensor data for localization and mapping in agricultural environments," *The International Journal of Robotics Research*, vol. 38, no. 6, pp. 633–641, 2019.