CEA Comité Español de Automática
XLVII JORNADAS DE AUTOMÁTICA CÓRDOBA 2026
UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Propuesta de tutoría cobótica mediante Torres de Hanói

Alberto Cebrián-García*, Inés Tejado, Blas M. Vinagre

Universidad de Extremadura, Escuela de Ingenierías Industriales, 06006 Badajoz, España.

To cite this article: Cebrián-García, A., Tejado, I., Vinagre, B. M. 2026. Proposal for cobotic tutoring through the Towers of Hanoi. Jornadas de Automática, 47. https://doi.org/10.17979/ja-cea.2026.47.13812

Resumen

Este trabajo presenta como ejemplo de Sistema Ciberfísico con Humano en el Lazo (SCHL) un banco de pruebas ciberfísico simulado para interacción humano-robot con asistencia pedagógica. El caso de uso son las Torres de Hanói, problema discreto con solución óptima conocida para evaluar decisiones, tolerancia a fallos y recuperación ante interrupciones. La arquitectura separa un motor de control en MATLAB/Stateflow de un modelo digital en Python/Pygame que imita un entorno físico hipotético. El supervisor combina legalidad, ruta óptima y una regla de paciencia que permite errores exploratorios antes de la corrección activa.

Palabras clave: Sistemas embebidos de control por computador y aplicaciones, Control supervisor y autómatas, Robótica inteligente, Ingeniería de sistemas centrada en humanos, Laboratorios virtuales y remotos

Proposal for cobotic tutoring through the Towers of Hanoi

Abstract

This paper presents a cyber-physical testbed for human-robot interaction with pedagogical assistance as an example of a Human-in-the-Loop Cyber-Physical Systems (HiL-CPS). The use case is the Towers of Hanoi, a discrete problem with a known optimal solution for assessing decisions, fault tolerance and recovery from interruptions. The architecture separates a MATLAB/Stateflow control engine from a Python/Pygame digital model of a hypothetical physical workspace. The supervisor combines legality checking, optimal-route comparison and a patience rule that allows exploratory mistakes before active correction.

Keywords: Embedded computer control systems and applications, Supervisory control and automata, Intelligent robotics, Human-centered systems engineering, Virtual and remote labs

1. Introducción

Los sistemas ciberfísicos (CPS) integran cómputo, comunicación, sensores, actuadores y procesos físicos bajo una misma lógica de operación. Esta integración exige diseñar sistemas capaces de coordinar recursos distribuidos, reaccionar ante cambios del entorno y mantener garantías de seguridad en presencia de incertidumbre humana (Lee, 2008; Mosterman and Zander, 2016). En robótica colaborativa, además, el problema no se limita al control de trayectorias: la máquina debe interpretar acciones humanas, regular su autoridad y comunicar su estado de forma comprensible. Este artículo desarrolla un prototipo preliminar para estudiar dichas dinámicas mediante el rompecabezas de las Torres de Hanói. La elección del caso de uso se justifica por tres razones. Primero, el problema posee reglas formales simples y una ruta óptima conocida, lo que permite evaluar decisiones humanas sin ambigüedad. Segundo, su estructura de apilamiento reproduce operaciones industriales de tipo *pick-and-place*. Tercero, ya ha sido empleado tanto como caso educativo de CPS como en escenarios de robótica y evaluación cognitiva asistida (Mosterman et al., 2013; Resing et al., 2020; Iqdymat and Stama-

*Autor para correspondencia: albertocg@unex.es

tescu, 2023). La contribución principal es una arquitectura de interacción humano-robot que combina evaluación algorítmica, tolerancia a fallos e intervención adaptativa. A diferencia de un sistema que únicamente bloquea acciones incorrectas, el prototipo permite que el usuario explore y se equivoque; si el patrón de error persiste, el robot pasa de una intervención pasiva a una acción correctiva que enseña el siguiente paso óptimo. La seguridad se trata como una prioridad superior a la productividad: cualquier intrusión en la zona compartida detiene temporizadores y evita movimientos automáticos.

El enfoque de esta propuesta se centra en la abstracción de la lógica de supervisión de alto nivel, centrándose en el aspecto de tutoría realizada mediante una máquina de estados finitos (FSM) que se aplicaría a un brazo robot. Por ello, la detección de intrusiones se modela mediante una señal booleana de entrada, dejando deliberadamente fuera del alcance del estudio la implementación física de sensores, la dinámica de los actuadores y el análisis de cumplimiento de normativas de seguridad industrial. Asimismo, la validación del sistema se aborda desde una perspectiva determinista de los estados lógicos de la FSM, asumiendo que las métricas cuantitativas de rendimiento físico y seguridad recaerían sobre el controlador del robot en un despliegue real.

2. Estado del arte

El diseño de CPS plantea retos de sincronización, control distribuido, integración de dominios y verificación de comportamiento emergente. Mosterman and Zander (2016) analizan estos retos desde la perspectiva de sistemas colaborativos embebidos y utilizan una variante distribuida de las Torres de Hanói para ilustrar dificultades de coordinación, visión, control y comunicación. De forma complementaria, Mosterman et al. (2013) proponen las Torres de Hanói como caso de estudio educativo de complejidad media, con suficiente fidelidad para explorar problemas auténticos de diseño basado en modelos. En el ámbito robótico, Iqdymat and Stamatescu (2023) estudian la resolución de las Torres de Hanói mediante un manipulador industrial simulado, integrando planificación de pasos, cinemática inversa en Simulink y validación de trayectorias en RobotStudio. Su enfoque se centra en la eficiencia física del manipulador. En cambio, el presente trabajo se centra en la capa cognitiva de supervisión: qué debe hacer el robot cuando el humano se equivoca, interrumpe la tarea o necesita una pista física. La literatura de interacción humano-robot subraya que la autonomía robótica debe diseñarse como una relación dinámica entre percepción, comunicación, autoridad compartida y confianza (Goodrich and Schultz, 2007; Sheridan, 2016). Además, las tareas de Torres de Hanói se han usado para analizar planificación y aprendizaje humano con ayuda de robots sociales. Resing et al. (2020) muestran que un robot puede proporcionar indicaciones graduadas durante la evaluación dinámica de resolución de problemas. Este antecedente apoya la idea de tratar al robot no solo como actuador, sino como asistente de aprendizaje.

3. Planteamiento del problema

Se considera una tarea de tres discos y tres pilares. Los discos se indexan por tamaño creciente, de modo que d_1 es el menor y d_3 el mayor. El estado discreto del tablero en el instante k se representa como

$$x(k) = [p_1(k), p_2(k), p_3(k)]^T, \tag{1}$$

donde $p_i(k) \in \{1, 2, 3\}$ indica el pilar ocupado por el disco d_i. El estado inicial es $x(0) = [1, 1, 1]^T$ y la meta es $x^\star = [3, 3, 3]^T$. Una transición humana es admisible si modifica exactamente un disco y respeta la restricción física de no colocar un disco grande sobre otro menor. Para n discos, el número mínimo de movimientos es

$$N(n) = 2^n - 1. \tag{2}$$

En el caso estudiado, $N(3) = 7$. La matriz de ruta óptima almacenada por el supervisor contiene la secuencia de la Tabla 1. Esta ruta no se emplea para impedir toda desviación humana, sino como referencia para distinguir entre progreso eficiente, exploración subóptima e infracciones físicas.

Tabla 1: Ruta óptima para tres discos.

Paso	Estado $x(k)$	Movimiento
0	$[1, 1, 1]$	Inicio
1	$[3, 1, 1]$	$d_1 : 1 \to 3$
2	$[3, 2, 1]$	$d_2 : 1 \to 2$
3	$[2, 2, 1]$	$d_1 : 3 \to 2$
4	$[2, 2, 3]$	$d_3 : 1 \to 3$
5	$[1, 2, 3]$	$d_1 : 2 \to 1$
6	$[1, 3, 3]$	$d_2 : 2 \to 3$
7	$[3, 3, 3]$	$d_1 : 1 \to 3$

El juego se realizará por turno, comenzando el turno del humano cuando el robot se queda esperando en un estado estable con el humano fuera de la zona de trabajo y acaba cuando el humano abandona la zona de trabajo tras mover un solo disco. El objetivo de control no es resolver automáticamente el rompecabezas, sino priorizar tres propiedades: seguridad del espacio compartido, corrección operativa del usuario y continuidad de la tarea. Por tanto, el sistema debe aceptar la imprevisibilidad humana dentro de límites seguros y convertir el error repetido en una oportunidad de instrucción.

4. Arquitectura del sistema

La arquitectura separa de forma explícita el motor de control de una representación ejecutable del entorno físico hipotético. Esta separación reduce el acoplamiento entre lógica de decisión y visualización/interacción, una práctica coherente con el diseño basado en modelos y con la posibilidad futura de sustituir la simulación por sensores y actuadores reales (MathWorks, 2025).

4.1. Motor de control

El motor de control se implementa en MATLAB, Simulink y Stateflow. Su núcleo es una FSM que actualiza memoria situacional, evalúa transiciones y selecciona el grado de intervención. La FSM recibe dos entradas principales: la topología actual de los discos y la señal booleana de presencia humana en la zona compartida, que en una implementación física futura podrían proceder de visión artificial y sensores de seguridad. Sus salidas combinan comandos de actuación, estado objetivo y telemetría para la interfaz de monitorización.

4.2. Modelo digital en Pygame

La capa Python/Pygame opera como modelo digital del entorno de trabajo hipotético. No se plantea como una réplica sincronizada de un activo físico existente (gemelo digital), sino como una representación ejecutable que imita restricciones de la realidad —por ejemplo, el apilamiento de discos—. Su función es capturar eventos de usuario, simular señales de sensores y animar las órdenes del supervisor. La lógica del rompecabezas queda centralizada en Stateflow, evitando divergencias entre lo que la máquina evalúa y lo que el operador observa.

4.3. Diccionario de variables de interfaz

La comunicación entre Simulink/Stateflow y Python se organiza mediante un diccionario de variables con entradas sensoriales y salidas de actuación o telemetría. Las entradas son escritas por Python en el modelo de Simulink; las salidas son generadas por la FSM y leídas por Python para actualizar el modelo digital.

Tabla 2: Variables de entrada hacia Stateflow.

Variable	Tipo	Función
`estado_humano`	Array 1 × 3, double	Topología actual del puzle: indica el pilar ocupado por cada disco, ordenados de menor a mayor. Imita una señal de visión artificial o sensores de posición.
`humano_en_zona`	Escalar, boolean	Señal binaria de intrusión: vale 1 si el operario invade el área colaborativa y 0 si la zona está despejada. Imita una barrera óptica o un escáner de seguridad.

Tabla 3: Variables de salida desde Stateflow.

Variable	Tipo	Función
`comando_robot`	Escalar, double	Orden de actuación para el agente simulado: reposo, intervención preventiva, corrección o movimiento automático.
`estado_objetivo`	Array 1 × 3, double	Topología destino que debe aplicarse durante una intervención, ya sea para deshacer un error o enseñar el siguiente paso correcto.
`codigo_msg`	Escalar, double	Etiqueta semántica de estado. Python traduce el código numérico en alertas, advertencias o confirmaciones del HMI.
`paso_actual`	Escalar, double	Seguimiento del progreso dentro de `ruta_optima`, usada para mostrar la compleción de la tarea en el HUD.
`mov_realizados`	Escalar, double	Contador acumulativo de movimientos físicos válidos realizados por el operador; permite comparar la eficiencia humana con la ruta ideal.

5. Algoritmo de supervisión y tutoría

El supervisor combina validación de seguridad, verificación de legalidad y evaluación de eficiencia. La Tabla 4 resume las variables críticas. Su uso conjunto permite diferenciar un nuevo error de una interrupción sobre un error ya detectado. La Figura 1 resume el comportamiento global de la FSM: la seguridad se evalúa antes que cualquier actuación, y las acciones correctivas dependen primero de la legalidad del movimiento y después de su eficiencia respecto a la ruta óptima.

Tabla 4: Memoria situacional del supervisor.

Memoria	Papel en la FSM
`estado_anterior`	Último estado aceptado como legal y eficiente; punto de restauración.
`ruta_optima`	Secuencia de referencia para evaluar progreso.
`paso_actual`	Índice de avance del usuario en la ruta óptima.
`fallos_consecutivos`	Contador de ineficiencias usado por la regla de paciencia.
`estado_fallido`	Registro del estado de los discos al suceder el último error para evitar penalizaciones dobles.

5.1. Evaluación multicapa

Cuando el usuario libera un disco y abandona la zona de trabajo, la FSM evalúa el cambio observado. La primera capa comprueba legalidad: detecta manipulaciones con más de un disco, violaciones de apilamiento o transiciones físicamente imposibles. La segunda capa compara el estado legal con la fila esperada de `ruta_optima`. Así se obtienen cuatro clases de evento: movimiento óptimo, movimiento subóptimo, infracción algorítmica (se ha movido más de un disco en el mismo turno) e infracción de las normas (colocar un disco sobre otro de menor tamaño). Un movimiento óptimo actualiza `estado_anterior`, incrementa `paso_actual`, actualiza `mov_realizados` y reinicia `fallos_consecutivos`. Una infracción o un movimiento legal subóptimo activa la política de asistencia, la cual permite 3 intentos por turno para realizar el movimiento óptimo. Esta decisión se toma en el motor de control; Python solo recibe `comando_robot`, `estado_objetivo` y la semántica visual asociada mediante `codigo_msg`.

5.2. Regla de paciencia

La asistencia adaptativa se basa en permitir exploración antes de asumir el control. Si el usuario realiza uno o dos movimientos subóptimos consecutivos, el agente robótico simulado realiza una intervención y restaura `estado_anterior`. La interfaz comunica que la decisión no progresa de forma eficiente, pero deja al operador la responsabilidad de encontrar el siguiente paso. Si el error se repite por tercera vez, la FSM interpreta que existe bloqueo cognitivo. En ese caso, el agente robótico simulado no deshace el movimiento únicamente, sino que aplica `estado_objetivo` correspondiente al siguiente paso óptimo. La autoridad cambia de pasiva a activa y la máquina actúa como tutor operativo dentro del entorno simulado. Esta política se inspira en la evaluación dinámica con ayudas graduadas: la asistencia aumenta cuando el desempeño del usuario muestra necesidad de apoyo (Resing et al., 2020). La Figura 2 detalla esta ayuda gradual: los dos primeros fallos conservan la responsabilidad de autocorrección en la persona, mientras que el tercer fallo activa una ayuda directiva hacia el siguiente estado óptimo. El funcionamiento más detallado puede verse en el pseudocódigo Algoritmo 1

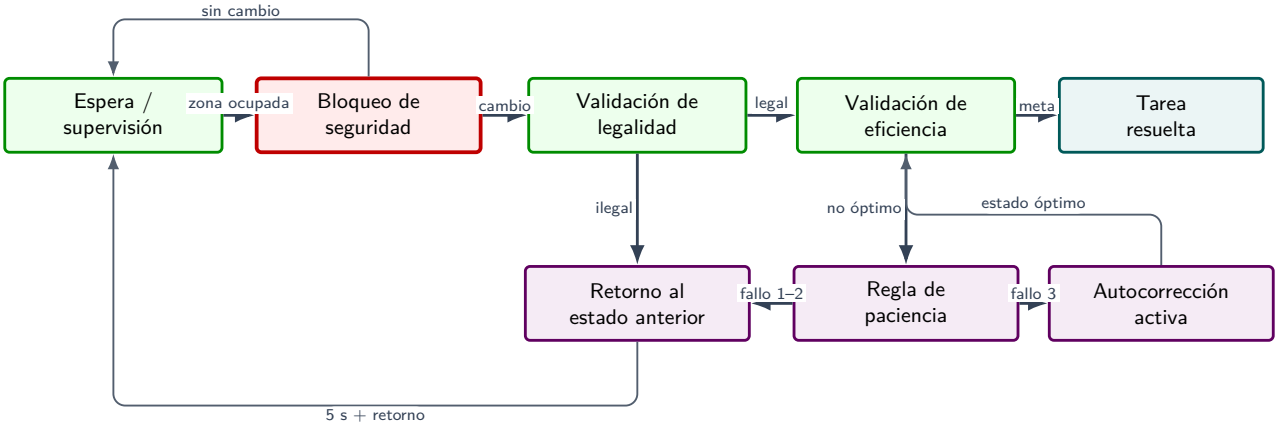

Figura 1: Flujo global simplificado del supervisor en Stateflow. El bloqueo de seguridad tiene prioridad sobre las correcciones automáticas; las esperas de 5 s representan el tiempo supuesto de intervención del agente robótico simulado al ocupar la zona de trabajo.

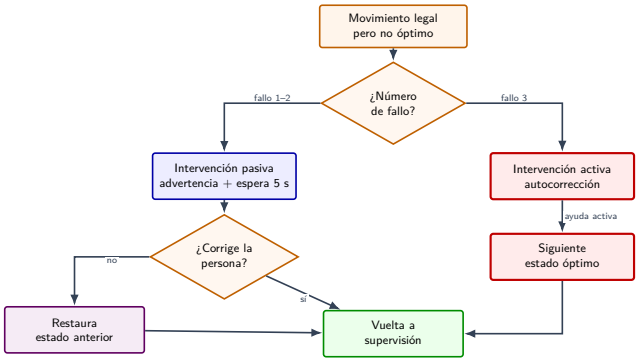

Figura 2: Regla de paciencia para movimientos legales no óptimos. Los fallos 1–2 generan advertencia, espera de 5 s y retorno al estado anterior si la persona no corrige; el fallo 3 activa la transición al siguiente estado óptimo.

Algoritmo 1 Algoritmo de supervisión y asistencia (regla de paciencia)

Require: x_h (estado humano), h (presencia), x_v (estado válido actual), j (paso óptimo), c (fallos), x^* (estado final)

if $h = 1$ **then**
  Mantener x_v y bloquear acción automática
else if el movimiento es ilegal **then**
  $c \leftarrow c + 1$
  Restaurar x_v tras espera
else if el movimiento es legal pero subóptimo **then**
  $c \leftarrow c + 1$
  Avisar y restaurar x_v tras espera
  if $c \geq c_{\text{máx}}$ **then**
    Ejecutar corrección hacia ruta$[j + 1]$
    $c \leftarrow 0$
  end if
else if el movimiento es legal y óptimo; $x_h =$ ruta$[j + 1]$ **then**
  $x_v \leftarrow x_h$
  $j \leftarrow j + 1$
  $c \leftarrow 0$
end if
if $x_v = x^*$ **then**
  Declarar éxito y preservar estado resuelto
end if

La política de paciencia se parametriza mediante $c_{\text{máx}}$ y T_r. El valor de 3 intentos se seleccionó arbitrariamente para dar al usuario más de una oportunidad de acertar el movimiento. Por su parte, la espera de 5 segundos se fijó como un tiempo suficiente para probar los escenarios de seguridad (dando margen al usuario para entrar en la zona de trabajo mientras el robot simula la intervención) , pero lo bastante corto para no hacer excesivamente lenta la simulación.

5.3. *Prioridad de seguridad y lógica temporal*

Las intervenciones no se ejecutan instantáneamente. La espera de 5 segundos simula el tiempo que tardaría un robot real en reposicionar los discos al invadir la zona de trabajo. Durante ese intervalo, cualquier activación de `humano_en_zona` transfiere la FSM a un bloqueo de seguridad de prioridad superior. La transición de seguridad domina sobre la aplicación automática, por lo que el actuador simulado cede siempre el espacio de trabajo al humano. Al desaparecer la intrusión, el sistema compara el estado actual con `estado_fallido`. Si el tablero sigue en la misma configuración errónea, se reanuda la cuenta atrás sin aumentar `fallos_consecutivos`, porque no se trata de un nuevo movimiento. Si el humano corrige manualmente durante el bloqueo, el supervisor borra `estado_fallido` y vuelve a espera, pero mantiene el registro del fallo y contabiliza el movimiento. Esta decisión evita incentivar entradas en la zona compartida mientras el agente simulado se dispone a intervenir.

6. Implementación del modelo digital

El modelo digital se implementa en Python 3.11 con Pygame. Su diseño busca representar de forma simplificada el entorno que existiría si se construyera una célula robótica real, evitando el coste temporal y material de fabricar el robot durante esta fase. Por tanto, no pretende ser un gemelo digital, sino un simulador interactivo suficiente para probar la mecánica colaborativa y docente del algoritmo. Los discos se almacenan como pilas LIFO en cada pilar, por lo que solo puede manipularse el disco superior. Esta restricción reproduce la interacción física básica de una tarea de ensamblaje o clasificación. La Figura 3 muestra la interfaz Pygame, separada de los diagramas de control para distinguir la representación visual del entorno y la lógica de decisión ejecutada en Stateflow.

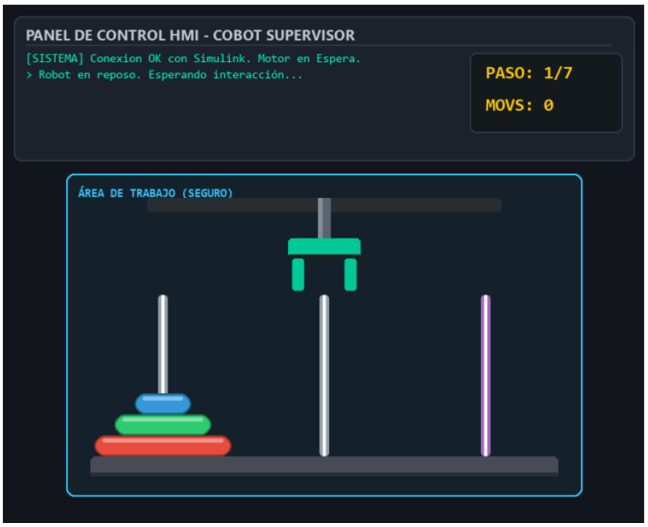

Figura 3: Interfaz del modelo digital en Python/Pygame. La captura muestra el panel HMI, el área de trabajo segura, los pilares, los discos y el actuador simulado utilizado para representar la intervención sobre el entorno hipotético.

La sensorización virtual incluye dos familias de señales. La primera es la detección de intrusión, calculada mediante colisión entre la posición del cursor y un rectángulo que representa el área compartida. Esta señal imita el papel que podría desempeñar una cortina de luz, un escáner de seguridad o una alfombra sensible. La segunda es la estimación de posición de discos, codificada como vector discreto después de cada liberación. En un montaje físico futuro, esta información podría obtenerse mediante visión artificial, etiquetas RFID o sensores inductivos en los pilares.

6.1. *Arquitectura de comunicación MATLAB–Python*

La transferencia de datos entre el modelo digital en Python y el controlador en Simulink se realiza mediante *MATLAB Engine API for Python*. La comunicación se divide en dos procesos con frecuencias distintas. En el sentido Python → Simulink, las señales `estado_humano` y `humano_en_zona` se escriben por eventos: Python detecta flancos y solo ejecuta `eng.set_param()` sobre bloques *Constant* cuando se registra un cambio real en el tablero o en la presencia humana. Así se evita enviar información redundante en cada fotograma.

En el sentido Simulink → Python, las salidas de Stateflow se exportan al *Base Workspace* mediante instrucciones `assignin` incluidas en bloques MATLAB Function. Python recupera `comando_robot`, `estado_objetivo`, `codigo_msg`, `paso_actual` y `mov_realizados` con `eng.evalin()` a una frecuencia fija de 5 Hz (200 ms). Esta lectura por *polling* conserva la fluidez del renderizado de Pygame a 60 FPS y mantiene desacoplado el tiempo gráfico del tiempo lógico de Simulink.

6.2. *Sesión compartida de MATLAB*

Durante el desarrollo se priorizó una sesión compartida de MATLAB frente al arranque clásico con `matlab.engine.start_matlab()`. El arranque clásico crea una instancia nueva en segundo plano, inicializa la máquina virtual de Java (JVM), carga rutas y recompila el modelo, lo que produce un tiempo de arrancado del sistema completo de 3 a 5 minutos y puede dejar procesos residuales si Python se interrumpe de forma abrupta. La alternativa consiste en abrir MATLAB previamente, ejecutar `matlab.engine.shareEngine` y enlazar Python con `matlab.engine.connect_matlab()`. MATLAB actúa entonces como servidor persistente y Python como cliente ligero. Esta estrategia reduce el tiempo de conexión a pocos segundos, facilita la depuración visual del diagrama de Stateflow, permite observar *Scopes* y errores de compilación en Simulink, y delega la gestión de memoria del archivo `.slx` al propio MATLAB. Desde el punto de vista experimental, esta configuración funciona como un banco de pruebas SIL virtual: MATLAB permanece activo como controlador lógico y Python puede conectarse o desconectarse como periférico de simulación. No obstante, al no existir todavía hardware físico conectado, esta caracterización debe entenderse como una analogía de despliegue y no como una validación software-in-the-loop estricta. Si en una fase posterior se construyera una célula física, la lógica de Stateflow podría reutilizarse conservando la misma interfaz de variables. La variable `humano_en_zona` podría conectarse a un relé de seguridad; `estado_humano`, a un subsistema de visión o identificación; y `estado_objetivo`, a un controlador de robot mediante ROS, Modbus TCP, EtherCAT u otro bus industrial. En esta fase preliminar, esta afirmación debe entenderse como viabilidad arquitectónica, no como validación de un robot real ni como certificación de seguridad funcional.

7. Validación funcional preliminar

La validación se ha planteado como una serie de pruebas sobre escenarios. El interés no está todavía en medir tiempos de usuario o carga cognitiva, sino en comprobar que la FSM mantiene seguridad y coherencia ante secuencias humanas no ideales. La Tabla 5 resume los casos cubiertos.

Tabla 5: Casos de validación funcional.

Caso	Condición	Respuesta esperada
Ruta óptima	El usuario sigue la Tabla 1.	Avance de `paso_actual` hasta éxito.
Error subóptimo	Movimiento legal fuera de ruta.	Advertencia y estado anterior tras 5 s.
Fallo repetido	Tres fallos consecutivos.	Corrección hacia el siguiente estado óptimo.
Trampa	Más de un disco cambia en el mismo turno.	Restauración del estado válido previo.
Intrusión	`humano_en_zona`=1 durante espera.	Aborto temporal de intervención automática.
Autocorrección	Corrección humana durante bloqueo.	Mantenimiento del castigo y olvido del error.
Cambio de idea	Cambio a movimiento diferente durante intervención.	Mantenimiento del castigo y del estado anterior a restaurar.

Los resultados preliminares muestran tres comportamien-

tos relevantes. Primero, la prioridad de seguridad es estanca: incluso cuando existe una intervención programada, la presencia humana cancela la ejecución de la acción automática. Segundo, la memoria `estado_fallido` permite tolerar interrupciones sin incrementar artificialmente el número de fallos. Tercero, el cierre de tarea preserva el estado resuelto; si el usuario desordena el tablero después de alcanzar la meta $x^\star$, el sistema lo identifica como desviación y restaura la solución.

8. Conclusión

El valor del prototipo reside en trasladar el énfasis desde la cinemática de un manipulador hacia la arquitectura condicional de la interacción humano-robot. La tarea de Torres de Hanói permite separar con claridad legalidad, eficiencia y seguridad. Esta separación es útil para diseñar robots colaborativos con grados variables de autoridad: observador, corrector pasivo y tutor activo. El enfoque presenta limitaciones. La validación solo atiende a la funcionalidad la FSM y no incluye experimentos con usuarios, métricas de aprendizaje, tiempos de respuesta reales ni incertidumbre sensorial. Además, el modelo digital simplifica la dinámica física hipotética: no modela fricción, oclusiones, fallos de agarre ni calibración de sensores. Por ello, la siguiente etapa debe incorporar un protocolo experimental con participantes, métricas objetivas de desempeño y una comparación entre asistencia pasiva, asistencia activa y ausencia de tutoría, además de una implementación física que aplique un control a más bajo nivel, teniendo en cuenta sensores, actuación y control robótico completo. Las mejoras previstas a más corto plazo incluyen la implementación de un mayor número de discos conforme el usuario mejora en el juego y un sistema fiable de evaluación del rendimiento del humano a medida que es tutorizado. Pese a estas limitaciones, la arquitectura es extensible. La ruta óptima puede sustituirse por un planificador para problemas mayores; la regla de paciencia puede transformarse en una política adaptativa basada en historial individual; y la interfaz Python podría reemplazarse por una planta física futura sin alterar el núcleo de decisión de la FSM de Stateflow. Esta modularidad favorece el uso del prototipo como banco de pruebas para seguridad, enseñanza y recuperación ante fallos en entornos colaborativos, centrándose en el funcionamiento del algoritmo de tutoría.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Agencia Estatal de Investigación (Ministerio de Ciencia e Innovación) a través del proyecto PID2022-141409OB-C22/AEI/10.13039/501100011033/FEDER, UE, por la Consejería de Educación, Ciencia y Formación Profesional (Junta de Extremadura) a través de la Ayuda a Grupos GR24059, y por Fondos Europeos de Desarrollo Regional (FEDER) “Una forma de hacer Europa”.

Referencias

Goodrich, M. A., Schultz, A. C., 2007. Human-robot interaction: A survey. Foundations and Trends in Human-Computer Interaction 1 (3), 203–275. DOI: 10.1561/1100000005

Iqdymat, A., Stamatescu, G., 2023. Analysis of trajectory optimization for solving tower of hanoi problem using industrial manipulator. In: 2023 24th International Conference on Control Systems and Computer Science (CSCS). pp. 11–15. DOI: 10.1109/CSCS59211.2023.00010

Lee, E. A., 2008. Cyber physical systems: Design challenges. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC). pp. 363–369. DOI: 10.1109/ISORC.2008.25

MathWorks, 2025. Cyber-physical systems with MATLAB and Simulink. Recurso técnico en línea.

Mosterman, P. J., Zander, J., 2016. Industry 4.0 as a cyber-physical system study. Software & Systems Modeling 15, 17–29. DOI: 10.1007/s10270-015-0493-x

Mosterman, P. J., Zander, J., Han, Z., 2013. The towers of hanoi as a cyber-physical system education case study. In: Proceedings of the First Workshop on Cyber-Physical Systems Education at CPSWeek. pp. 1–4.

Resing, W. C. M., Vogelaar, B., Elliott, J. G., 2020. Children’s solving of ‘tower of hanoi’ tasks: Dynamic testing with the help of a robot. Educational Psychology 40 (9), 1136–1163. DOI: 10.1080/01443410.2019.1684450

Sheridan, T. B., 2016. Human-robot interaction: Status and challenges. Human Factors 58 (4), 525–532. DOI: 10.1177/0018720816644364