Actas ROBOT 2011. 28-29 de Noviembre de 2011. Sevilla (España)

Aprendizaje de Controladores Difusos para Seguimiento de Trayectorias en Robots Multiarticulados

F.J. Marín, J. Casillas, M. Mucientes, A.A. Transeth, S.A. Fjerdingen, I. Schjølberg

***Abstract*— Los robots multiarticulados con ruedas activas proporcionan oportunidades interesantes en muchas áreas, tales como inspección o mantenimiento de tuberías o sistemas de ventilación. Una funcionalidad clave para poder realizar estas tareas es que el robot pueda seguir una ruta predefinida de forma rápida y precisa. En este artículo se presenta un algoritmo para aprender el seguimiento de rutas para un conjunto de primitivas de movimiento. Estas primitivas pueden ser usadas por un planificador para construir caminos más largos. El algoritmo se presenta dividido en dos pasos: un primer aprendizaje del controlador, basado en ejemplos y un posterior refinamiento basándose en una función objetivo con simulaciones del proceso de seguimiento de rutas. Los controladores se han probado en el simulador de un robot multiarticulado con varias rutas complejas, consiguiendo un excelente rendimiento.**

I. INTRODUCCIÓN

Los robots móviles constituyen plataformas versátiles para un amplio rango de operaciones. En particular, los robots multiarticulados constituyen una clase importante de robots móviles con un gran potencial, ya que son flexibles y ágiles. Gracias al empleo de ruedas activas a lo largo de su cuerpo articulado tienen una gran capacidad de movimiento en ambientes muy complejos, tales como estructuras de tuberías, sistemas de ventilación y aberturas de gran tamaño, los cuales pueden ser o muy complicados o muy peligrosos para las personas que operan en estas estructuras. Una funcionalidad clave para realizar estas operaciones es que el robot pueda seguir una ruta predefinida de forma rápida y precisa. Además, el robot debe ser capaz también de recuperar la trayectoria cuando es necesario desviarse (por ejemplo, tras esquivar un obstáculo situado en el camino).

Entre las posibles aplicaciones de este tipo de robots se encuentra la inspección y el mantenimiento de estructuras de tuberías y sistemas de ventilación y en las operaciones de búsqueda y rescate. La inspección de tuberías es un mercado en crecimiento, caracterizado por los requisitos de una mayor seguridad y conciencia de calidad por parte de la industria y la ciudadanía. Las refinerías, plantas químicas, plantas nucleares, la industria del petróleo, servicios públicos, los hogares y edificios tienen millones de metros de tuberías, algunos de los cuales están expuestos a condiciones ambientales interiores o exteriores muy fuertes. Como resultado, la industria está continuamente renovándose para garantizar que la calidad y estado de las tuberías cumpla con las normas establecidas por los organismos reguladores. En estos casos, estos robots proporcionarían varias ventajas:

- permitiría inspeccionar tuberías complejas que no pueden tratarse con los sistemas actuales,
- robustez debido a la forma de serpiente y el hecho de que el robot puede estar protegido por una capa adecuada para su aplicación,
- fiabilidad debido a la integración de datos de sensores a lo largo del cuerpo, en la cabeza y el extremo final,
- flexibilidad gracias a la autonomía dotada al robot.

Para este último punto, el seguimiento de rutas es una funcionalidad clave, ya que el propio robot puede decidir el camino que debe seguir y atravesarlo con la menor desviación posible, ya que tanto las tuberías como los sistemas de ventilación son espacios estrechos en los que no hay mucho margen para el error.

Los robots multiarticulados pueden aplicarse también en operaciones de búsqueda y rescate ya que su cuerpo estrecho y flexible le permite atravesar terrenos difíciles, como por ejemplo edificios colapsados tras un derrumbamiento. Además, estos robots se pueden aplicar para el análisis de la situación en caso de incendios en edificios cuando son inaccesibles para los seres humanos.

El estudio del seguimiento de rutas para robots móviles es amplio, pero gran parte de estos experimentos se han limitado a robots móviles simples [1], [2], [3], [4], [5]. Estos robots también cuentan con restricciones en sus movimientos, pero la cinemática no es tan compleja como la de los robots serpiente y, por ello, la complejidad para el seguimiento de rutas es menor.

En este artículo presentamos un algoritmo para el aprendizaje de seguimiento de rutas para robots multiarticulados con ruedas activas. A diferencia de [4], el robot intenta reducir tanto el error en la orientación como la desviación en cada paso, no de forma independiente, ajustando la velocidad lineal y angular. En esta propuesta, se reduce la complejidad del aprendizaje dividiendo los caminos largos en una serie de pequeñas primitivas de movimiento [6] (figura 1a) con la que se puede alcanzar casi cualquier punto cercano. Por eso se aprende una serie de controladores (uno por cada primitiva) en vez de un solo controlador. La combinación de estas primitivas de movimiento permiten construir caminos complejos entre cualquier par de puntos en el entorno (figura 1b). Un planificador simple puede realizar tanto la construcción del camino a partir de las primitivas como la selección del controlador apropiado para cada sección de la ruta compleja. Este método favorece la precisión ya que cada controlador está dedicado a una sola primitiva de movimiento.

F.J. Marín and J. Casillas are with the Dept. Computer Science and Artificial Intelligence, University of Granada (Spain). M. Mucientes is with the Dept. Electronics and Computer Science, University of Santiago de Compostela (Spain). A.A. Transeth, S.A. Fjerdingen and I. Schjølberg are with SINTEF ICT Applied Cybernetics (Norway).

(a) (b)

Fig. 1: a) Conjunto de primitivas de movimiento. b) Un patrón repetitivo de primitivas construyen todo el camino (representado en negro)

Se ha usado un robot multiarticulado con ruedas activas llamado PIKo (Pipe Inspection Konda) como base para los experimentos en simulación del algoritmo presentado. Los resultados de la simulación muestran que este robot puede seguir las rutas con gran precisión. Además, es capaz de recuperarse rápidamente cuando se sitúa con una desviación inicial. Este método de aprendizaje se puede aplicar a otros robots multiarticulados sólo cambiando el modelo cinemático y algunos valores iniciales.

Este artículo se divide en las siguientes secciones: la sección 2 presenta el robot multiarticulado usado en los experimentos en simulación y su simulador, la sección 3 detalla el algoritmo de aprendizaje de los controladores para el seguimiento de rutas, la sección 4 analiza los resultados obtenidos y la sección 5 presenta las conclusiones.

II. Descripción del Robot PIKo

Se ha utilizado un robot serpiente con ruedas activas (figura 2) llamado PIKo (Pipe Inspection Konda)[1] para validar el algoritmo presentado en este artículo. PIKo ha sido desarrollado por la organización noruega SINTEF [7]. Este robot está compuesto de cinco módulos interconectados con uniones con dos grados de libertad. Las ruedas proporcionan un tercer grado de libertad por cada módulo. Cada módulo tiene un momento máximo de 11Nm y un peso de 1.2kg. También incluye como sensores codificadores de ángulo para cada una de las uniones, codificadores de velocidad y una cámara TOF (time-of-flight) 3D. Las ventajas de este robot son las siguientes:

- El largo y articulado cuerpo de los robots serpiente los hace ideales para inspecciones internas de estructuras complejas u otros espacios estrechos. PIKo, además, puede moverse a través de estos espacios manteniendo el movimiento directo de un vehículo con ruedas. Esto representa una gran ventaja frente a los robots con ruedas pasivas ya que su movimiento es más simple.
- El robot tiene muchos grados de libertad (velocidad, inclinación horizontal y vertical por cada módulo) pero su movimiento puede simular el problema del n-trailer [8] con gran precisión a través de un conjunto de ecuaciones cinemáticas, reduciendo el número de parámetros necesarios para controlarlo [9].
- Contamos con un simulador del robot PIKo, basado en un motor físico de código abierto llamado ODE (Open Dynamics Engine) [10] que ha sido desarrollado por SINTEF y la Universidad Noruega de Ciencias y Tecnología (NTNU). Hemos usado este simulador para probar los controladores generados y en la fase de refinamiento del algoritmo presentado en este artículo.

[1] http://www.sintef.no/home/Information-and-Communication-Technology-ICT/Applied-Cybernetics/Projects/The-Pipe-Inspection-robot-PiKo/

Fig. 2: PIKo, el robot serpiente

II-A. Simulador del Robot PIKo

La velocidad en la generación y prueba de los diferentes controladores hace estrictamente necesaria la utilización de un simulador realista de este robot. Este simulador ha sido proporcionado por SINTEF (figura 3) y está basado en motores de código abierto, tanto para la física como para gráficos.

La librería ODE [10] (Open Dynamic Engine) controla el motor físico. Ofrece un buen rendimiento en la simulación de la dinámica de cuerpos rígidos y sobretodo en la simulación de articulaciones. ODE cuenta con varios tipos de articulaciones simples y complejas, desde articulaciones fijas hasta pistones y proporciona su propio detector de colisiones para varios tipos de objetos. Es muy útil para la simulación de vehículos y elementos articulados.

Como motor gráfico utiliza Irrlicht [11]. Es una librería muy versátil con la que se pueden manejar desde líneas y polígonos simples hasta escenarios complejos y ofrece gran rendimiento en animaciones.

El simulador de SINTEF ha sido modificado para ofrecer soporte también a otros robots de la fundación, como WheeKo, un robot con ruedas pasivas cuyo movimiento se basa en el de serpientes reales, y se le ha añadido soporte para rastrear el seguimiento del movimiento de los robots.

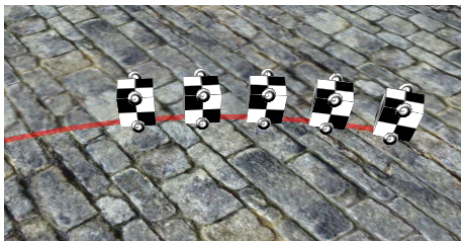

Fig. 3: Simulador del robot PIKo

III. Algoritmo de Aprendizaje para el Seguimiento de Rutas

En este trabajo describimos un método dividido en dos fases para realizar el aprendizaje del seguimiento de rutas. Esta técnica combina aprendizaje inductivo y deductivo en métodos de optimización con un primer aprendizaje basado en ejemplos adaptado de [12] y una fase de refinamiento con datos extraídos del simulador, que mejora la precisión de los controladores obtenidos. Esta propuesta es válida para cualquier algoritmo de aprendizaje basado en la optimización de una función objetivo (por ejemplo, algoritmos genéticos o redes neuronales artificiales).

III-A. Aprendizaje Basado en Ejemplos

En esta fase se aprende, a través de un conjunto de ejemplos, un controlador difuso general que servirá como semilla en la fase de refinamiento. Cada ejemplo del conjunto de entrenamiento se compone de una combinación de variables de estado y de acción. Los ejemplos se generan en el espacio de las variables de entrada (estado), empezando desde el valor mínimo e incrementando el valor en una cantidad p_i hasta que se alcanza el valor máximo. El conjunto de ejemplos se crea después combinando estos valores para todas las variables de entrada. Por otra parte, los valores de las variables de salida (acción) se calculan probando todas las posibles combinaciones de las variables de salida (discretizadas con precisión p_i) y seleccionando aquellos valores que sitúan al robot en el estado más cercano al ideal, de acuerdo con una función de puntuación. Estos ejemplos son usados posteriormente como conjunto de entrenamiento de un algoritmo avanzado que aprende el controlador difuso (base de datos y de reglas) que mejor se ajusta a los datos.

Es necesario definir los siguientes elementos para poder detallar esta fase:

- Modelo cinemático del robot.
- Varaibles de entrada y de salida.
- Límites, número de conjuntos difusos y precisión (p_i) de cada variable.
- La función de puntuación (*SF*) que evalúa la acción del controlador difuso para cada ejemplo.
- La función de prueba, que evalúa la calidad del controlador generado.

III-A.1. Modelo cinemático.: El modelo cinemático del robot PIKo se describe en [7], [9]. Para poder calcular la posición siguiente de la cabeza del robot, son necesarias las siguientes funciones:

$$\theta_1(t+1) = \theta_1(t) + \frac{V_{P1}}{L_{PJ}} \cdot tan\,(\delta_1) \cdot \Delta t, \tag{1}$$

$$\begin{aligned} x_1(t+1) &= x_1(t) + V_{P1} \cdot sin\,(\theta_1) \cdot \Delta t, \\ y_1(t+1) &= y_1(t) + V_{P1} \cdot sin\,(\theta_1) \cdot \Delta t, \end{aligned} \tag{2}$$

$$\begin{aligned} \phi_2(t+1) = \; & \phi_2(t) - \frac{V_{P1}}{L_{PJ}} \cdot (sin\,(\phi_2(t)) + \\ & \Big(\frac{L_{JP}}{L_{PJ}} \cdot cos\,(\phi_2(t)) + 1\Big) \cdot tan\,(\delta_1)\Big) \cdot \Delta t \end{aligned} \tag{3}$$

donde x_1, y_1 y θ_1 son la posición y orientación del primer módulo del robot, y ϕ_2 es el ángulo entre el primer y segundo módulo del robot. V_{P1} y δ_1 son la velocidad lineal y angular del primer módulo y L_{PJ} y L_{JP} son las longitudes de los segmentos $P_i\ J_i$ y $P_i\ J_i$, siendo P_i el punto central del eje de las ruedas del módulo i y J_i el punto central situado en el frente del módulo i.

III-A.2. Variables de entrada y de salida.: La desviación del robot respecto de la ruta ideal debe ser mínima en cada paso. Podemos usar las fórmulas de Frenet-Serret para encontrar la desviación en la posición y orientación de la cabeza del robot con respecto del camino. Por ello, dos de las variables de entrada deben ser la distancia de la posición del robot al punto más cercano de la ruta (Δz) y el error de orientación ($\Delta\theta$). Para la orientación, en vez de usar el punto más cercano de la ruta a la posición actual del robot, se utiliza el punto más cercano de la ruta a la posición de la serpiente en el siguiente instante de tiempo. Este método tiene algunas ventajas:

- El robot puede controlar desviación y orientación al mismo tiempo.
- El proceso de recuperación es suave.

La otra variable de entrada es la velocidad linear actual (v). Finalmente, las variables de salida son aceleración (a) y velocidad angular (δ_1).

Esta selección difiere a la propuesta en [14]. Se ha prescindido de ϕ_2 en las variables por motivos de rapidez en el aprendizaje. ϕ_2 estará limitado en este caso por la función cinemática del robot.

III-A.3. Límites y Precisiones de las Variables.: Para poder generar el conjunto de entrenamiento, es necesario establecer los límites y precisiones de las variables. Para algunas de las variables los límites pueden ser muy altos o incluso infinitos, como por ejemplo: $\Delta z \in [-\infty, \infty]$, $\Delta\theta \in ([-180^{\text{o}},\ 180^{\text{o}}]$. Para estas variables, el rango debe ser una versión reducida de los límites reales y debe contener aquellos valores de la variable que sean realmente útiles para el aprendizaje (valores grandes para las distancias no son útiles, porque para todos ellos el robot ejecutará la misma acción).

Los límites y precisiones para cada variable son: $\Delta z \in [-0{,}09,\ 0{,}09]$ (se usan los valores negativos si el robot se encuentra desviado a la izquierda y positivos si se desvía a la derecha), $p_{\Delta z} = 0{,}0075$; $\Delta\theta \in [-90,\ 90]$, $p_{\Delta\theta} = 7{,}5$; $v \in [0{,}05,\ 0{,}2]$, $p_v = 0{,}05$; $a \in [-0{,}1,\ 0{,}1]$, $p_a = 0{,}025$; $\delta_1 \in [-20,\ 20]$, $p_{\delta_1} = 1$. El uso de pesos dinmicos de la función de puntuación permite estos límites amplios aunque el movimiento del robot sea ms reducido.

III-A.4. Función de puntuación.: Un aspecto importante de la técnica de generación del conjunto de entrenamiento es la definición de la función de puntuación (*SF*), una función que evalúa la acción del controlador difuso sobre un ejemplo. El papel de la función *SF* es medir la desviación del valor de cada variable del valor ideal. Para el seguimiento de rutas, el robot necesita alcanzar el punto más cercano al camino deseado manteniendo un error de orientación bajo. Esto tiene dos grandes problemas:

- Si el robot se encuentra en un punto sobre la ruta, la mejor opción es quedarse en ese mismo sitio, porque otras acciones pueden incrementar el error en la orientación o la distancia.
- Es crucial encontrar un buen balance entre la mejora del error de orientación y de la desviación, ya que estas dos variables son excluyentes entre sí: si queremos reducir la desviación, el error en la orientación debe ser incrementado.

A diferencia de [14], la velocidad ha sido acotada para que exista una velocidad mínima y el robot no se detenga, aunque las velocidades más bajas son penalizadas, de forma que el robot tienda a alcanzar velocidades cercanas a la máxima. El segundo problema puede solucionarse parcialmente con pesos dinámicos: los pesos de la desviación y del error de orientación dependen de sus respectivos errores iniciales. Cuando la distancia inicial es pequeña, su peso es menor, incrementando la importancia del error de orientación y la velocidad. La elección de pesos dinámicos además cumple otra función: permite ampliar los límites de las variables para el aprendizaje ya que no siempre se escogerá la misma acción para valores altos, dependerá de la importancia de las otras variables.

La función de puntuación se define de la siguiente manera:

$$SF(RB(e^l)) = \alpha_1 + \alpha_2 + \alpha_3, \tag{4}$$

donde e^l es el l-ésimo ejemplo y *SF* es la puntuación del estado alcanzado por el robot, empezando en el estado definido por e^l y aplicando la acción propuesta por la combinación de las variables de salida de la base de reglas del controlador (RB). α_1, α_2 y α_3 se calculan del siguiente modo:

$$\begin{aligned} &\alpha_1 = \omega_1 \cdot \frac{e^l_{\Delta z}}{max_{\Delta z}} \cdot \frac{\Delta z}{p_{\Delta z}}, \\ &\Delta z = \sqrt{\left(x_{robot} - x_{path}\right)^2 + \left(y_{robot} - y_{path}\right)^2} \end{aligned} \tag{5}$$

$$\alpha_2 = \omega_2 \cdot \frac{e^l_{\Delta\theta}}{max_\theta} \cdot \frac{\Delta\theta}{p_{\Delta\theta}},\ \Delta\theta = |\theta_{robot} - \theta_{path}| \tag{6}$$

$$\alpha_3 = \omega_3 \cdot \frac{(max_v - v)}{p_v} \tag{7}$$

ω_1, ω_2 y ω_3 son tres pesos usados para definir la importancia de cada variable en la función de puntuación. La elección de estos pesos no es trivial ya que dependen de los límites de las variables escogidos y de las propias características de movimiento del robot. $\frac{e^l_{\Delta z}}{max_{\Delta z}}$ es el peso dinámico para la distancia. $\frac{\Delta z}{p_{\Delta z}}$ determina la puntuación de la acción dividida por la precisión de la variable. Esto hace posible la comparación de las desviaciones de las diferentes variables. La puntuación para la orientación se calcula del mismo modo. Finalmente, las velocidades más bajas son penalizadas en α_3. Con esta fórmula se puede comprobar que las mejores acciones son aquellas que alcanzan las mayores velocidades, reduciendo el error de desviación y de orientación. Por lo tanto, el valor de la función de puntuación debe ser el mínimo posible.

III-A.5. Función de prueba.: Tras el aprendizaje del controlador, la calidad de éste debe ser evaluada. Esto se ha realizado probando todos los controladores con el simulador del robot PIKo en varias rutas de ejemplo. El desvío (Δz) y la velocidad (v) son registrados en cada paso hasta que el robot alcanza el final de la ruta o un límite máximo de pasos. Después se calcula la distancia media, su desviación típica y la velocidad media y se presentan junto a una variable que indica si se ha alcanzado el final de la ruta (1) o no (0).

III-B. Fase de Refinamiento

Aprender todo el controlador (base de datos y de reglas) con ejemplos es más fácil y rápido que aprenderlo a través de una función objetivo y datos del simulador. Los controladores obtenidos no son perfectos, pero son muy buenos como base para la fase de refinamiento. En esta fase, el controlador semilla (obtenido de la fase anterior) se refina basándose en varias primitivas de movimiento, creando un conjunto de controladores difusos (uno por primitiva). Al igual que en la primera fase, se ha utilizado un algoritmo de aprendizaje. Este algoritmo usa el controlador seleccionado previamente e intenta mejorar su base de conocimiento a través de una función objetivo.

Para esta fase es necesario establecer los siguientes elementos:

- El conjunto de primitivas de movimiento (figura 1a).
- La función objetivo. Esta función evalúa la calidad del controlador.

III-B.1. Función objetivo.: En la fase de refinamiento necesitamos mejorar tanto el mantenimiento de la ruta como la recuperación: el primero minimizando el error en la distancia mientras se mantiene una velocidad alta, y el segundo reduciendo el número de pasos necesario para alcanzar el camino con un estado estable (que no vuelva a producir grandes desvíos). La función objetivo (que debe ser minimizada) considera las siguientes variables:

$$ObjF = \omega_{dev} \cdot dev + \omega_{rec} \cdot rec \tag{8}$$

donde ω_{dev} y ω_{rec} son los pesos que determinan la importancia de los comportamientos de mantenimiento de la ruta y de recuperación, y dev y rec son las variables que miden la calidad del controlador en ambos comportamientos:

$$dev = \omega_{\Delta z} \cdot \overline{\Delta z} + \omega_v \cdot (max_v - \overline{v}) \tag{9}$$

$$rec = \omega_{\Delta z_r} \cdot \overline{\Delta z_r} + \omega_{rv} \cdot (max_{rv} - \overline{rv}) \tag{10}$$

La ecuación 9 usa los mismos parámetros que la función de prueba: error medio en la desviación ($\overline{\Delta z}$) y velocidad media ($\overline{v}$) durante una prueba completa del controlador (hasta que el robot alcanza el final de la ruta o un número máximo de pasos). A estos valores se le aplican los pesos $\omega_{\Delta z}$ y ω_v respectivamente.

La ecuación 10 utiliza el número de pasos necesario para alcanzar un punto cercano a la ruta ideal (rv), y la desviación media desde que se alcanza ese punto hasta el final de la prueba ($\overline{\Delta z_r}$). rv se calcula como sigue:

$$rv = \frac{n_r}{n_{steps}} \tag{11}$$

donde n_r es el número de pasos necesario para alcanzar el punto cercano a la ruta ideal y n_{steps} representa el número de pasos que se ha tardado en terminar la prueba.

IV. RESULTADOS

IV-A. Descripción

Se han realizado pruebas con cinco pesos diferentes para ω_1, ω_2 y ω_3 para la generación del conjunto de entrenamiento y cinco semillas diferentes para cada conjunto de pesos. Los controladores aprendidos se han probado en el simulador del robot PIKo, con varias rutas de diversa complejidad. Es importante remarcar que estas rutas no se han usado durante el entrenamiento ya que en la primera fase el conjunto de entrenamiento está compuesto solamente por una lista de ejemplos que cubren todo el espacio de las variables de entrada con una precisión adecuada. Se han realizado nueve evaluaciones para cada controlador: uno sin desvío inicial y ocho más con diferentes desvíos iniciales (para las pruebas de recuperación), y se ha registrado la distancia media y su desviación típica, la velocidad media y el número de pasos necesarios para recuperación en la tabla I.

Se ha utilizado un algoritmo genético avanzado para aprendizaje difuso llamado EGLFP [13] para el aprendizaje y refinamiento de los controladores. Hemos usado los siguientes valores para los parámetros de este algoritmo: 50,000 evaluaciones para aprendizaje y 5,000 para refinamiento, 50 individuos, 0.8 para la probabilidad de cruce y 0.3 como probabilidad de mutación.

Para la fase de refinamiento se ha utilizado el conjunto de primitivas de movimiento presentado en la figura 1. Está compuesto de 11 primitivas diferentes, por lo que se han generado 11 controladores diferentes. Hemos usado una versión modificada de EGLFP para esta tarea, reemplazando el aprendizaje por ejemplos con una función objetivo, definida en la ecuación 8 y datos extraídos del simulador de PIKo. Los valores de los pesos para esta fase son los siguientes: ω_{dev} = 0.6, ω_{rec} = 0.4, $\omega_{\Delta z}$ = 0.90, ω_v = 0.10, $\omega_{\Delta z_r}$ = 0.975 y ω_{rv} = 0.025. Se han usado cinco semillas para cada primitiva y cada controlador se ha evaluado 9 veces, como en la primera fase. La velocidad media, el número de pasos necesario para la recuperación y la desviación media tras la recuperación son recogidos y presentados en la tabla II.

IV-B. Resultados

En esta sección se presentan los resultados obtenidos por los diferentes controladores aprendidos, tanto en la primera fase como en la segunda, y un pequeño estudio sobre los cinco pesos de la función de puntuación de la primera fase. La tabla I recoge los resultados de estos pesos para el problema del seguimiento de rutas con desvío inicial.

Como se puede ver en la tabla I, un excesivo peso asignado a la distancia (por ejemplo, 0.95) permite una recuperación más rápida pero se vuelve mucho más inestable. Esto ocurre con los controladores de las dos primeras combinaciones de pesos. Los controladores de la tercera y cuarta combinación, aunque también son inestables, pueden ser mejorados en la siguiente fase. A partir de los pesos (0.75, 0.25, 1) y conforme se va dando mayor peso al error en la orientación, la recuperación se realiza de forma más suave llegando a no recuperar la trayectoria ideal si este peso es excesivo. Sin pesos dinámicos no hubiera sido posible encontrar un balance justo. Podemos seleccionar cualquiera de los controladores generados con los pesos de la quinta fila, para ser la semilla de la fase refinamiento.

TABLE I: Datos de recuperación de la ruta ideal. El desvío se mide en metros y la velocidad en m/s

Pesos (ω_1, ω_2, ω_3)	Right (2,2,90)		
	Desv	Vel	Rec
(0.95, 0.05, 1)	0.098 ± 0.057	0.20	0.15
(0.9, 0.1, 1)	0.044 ± 0.027	0.20	0.18
(0.85, 0.15, 1)	0.021 ± 0.016	0.19	0.18
(0.8, 0.2, 1)	0.021 ± 0.013	0.19	0.28
(0.75, 0.25, 1)	0.013 ± 0.009	0.19	0.27
Pesos (ω_1, ω_2, ω_3)	Left (1,2,0)		
	Desv	Vel	Rec
(0.95, 0.05, 1)	0.098 ± 0.062	0.19	0.14
(0.9, 0.1, 1)	0.050 ± 0.036	0.20	0.15
(0.85, 0.15, 1)	0.025 ± 0.019	0.19	0.29
(0.8, 0.2, 1)	0.021 ± 0.014	0.19	0.33
(0.75, 0.25, 1)	0.013 ± 0.006	0.19	0.31
Pesos (ω_1, ω_2, ω_3)	Left (1,1,90)		
	Desv	Vel	Rec
(0.95, 0.05, 1)	0.105 ± 0.059	0.19	0.30
(0.9, 0.1, 1)	0.066 ± 0.036	0.20	0.30
(0.85, 0.15, 1)	0.050 ± 0.030	0.19	0.45
(0.8, 0.2, 1)	0.037 ± 0.020	0.18	0.43
(0.75, 0.25, 1)	0.028 ± 0.020	0.18	0.48
Pesos (ω_1, ω_2, ω_3)	Right (2,2,45)		
	Desv	Vel	Rec
(0.95, 0.05, 1)	0.103 ± 0.061	0.20	0.13
(0.9, 0.1, 1)	0.042 ± 0.028	0.20	0.17
(0.85, 0.15, 1)	0.022 ± 0.019	0.20	0.21
(0.8, 0.2, 1)	0.016 ± 0.012	0.19	0.30
(0.75, 0.25, 1)	0.010 ± 0.006	0.19	0.31

Los resultados del refinamiento tras la segunda fase se presentan en la tabla II, en la que se presentan la desviación (media y desviacíon típica), la velocidad y los pasos necesarios para recuperar el camino tras esta fase. Cada primitiva de movimiento se identifica con su dirección, desplazamientos x e y, y cambio en la orientación (en grados). Cada controlador ha sido probado con su correspondiente primitiva.

TABLE II: Datos de seguimiento y recuperación tras la fase de refinamiento. La distancia se recoge en metros y la velocidad en m/s

Primitiva	Desv	Rec	Vel
Line (2,0,0)	0.002 ± 0.002	0.25	0.2
Left (1,1,90)	0.004 ± 0.003	0.29	0.18
Right (1,1,90)	0.005 ± 0.004	0.31	0.17
Left (2,2,90)	0.003 ± 0.003	0.14	0.19
Right (2,2,90)	0.004 ± 0.003	0.15	0.19
Left (2,2,45)	0.005 ± 0.003	0.18	0.19
Right (2,2,45)	0.007 ± 0.006	0.13	0.18
Left (1,2,45)	0.005 ± 0.003	0.20	0.19
Right (1,2,45)	0.004 ± 0.005	0.17	0.18
Left (1,2,0)	0.009 ± 0.005	0.23	0.19
Right (1,2,0)	0.009 ± 0.005	0.23	0.19

Como podemos ver en la tabla II, el proceso es muy preciso: sólo unos pocos milímetros de desviación media para todas las primitivas, con un mantenimiento estable de la

trayectoria (desviación típica baja) y a una buena velocidad. El proceso de recuperación es rápido también: sólo un 15-30 % del total de los pasos son necesarios para recuperarse de desvíos iniciales de 0.2 m. Con respecto a [14] los resultados de desviación son parecidos, pero se ha conseguido a una mayor velocidad rozando casi siempre la velocidad máxima del robot. Este sería el comportamiento ideal para un robot que debe moverse por sitios estrechos como tubos o sistemas de ventilación ya que controla en cada momento a que distancia se encuentra y trata de corregirla para que sea la mínima posible y no choque con las paredes.

A continuación se presentan varias figuras (4, 5 y 6) con ejemplos de recuperación para distintas primitivas (la trayectoria del robot se representa en línea continua, la trayectoria de la primitiva en línea de puntos):

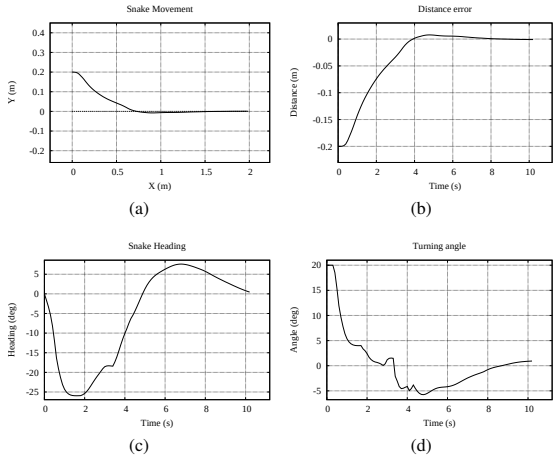

Fig. 4: Prueba de recuperación para la primitiva line(2,0). a) Trayectoria, b) Distancia (Δz), c) Orientación (θ), d) Velocidad angular (δ_1)

V. CONCLUSIONES

En este artículo presentamos un método de aprendizaje de trayectorias para robots multiarticulados con ruedas activas. El método se divide en dos fases: un primer aprendizaje basado en ejemplos y un posterior refinamiento para mejorar la precisión. Este proceso se aplica a un conjunto de primitivas de movimiento, obteniendo un controlador difuso para cada una de ellas, que pueden combinarse para crear caminos más largos. La fase de refinamiento proporciona controladores muy precisos para el problema del seguimiento de rutas: los errores de desviación se reducen a varios milímetros. Esto es muy importante, ya que estos robots deben moverse por espacios estrechos como tuberías o conductos de ventilación, donde grandes desvíos no son posibles. Además consigue que en caso de desvío importante, el robot pueda recuperar rápidamente la trayectoria.

El uso de primitivas de movimiento permite al robot alcanzar casi todos los puntos del espacio con la combinación

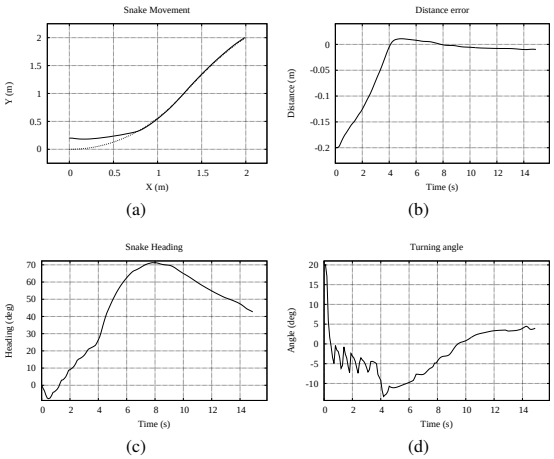

Fig. 5: Prueba de recuperación para la primitiva left(2,2,45). a) Trayectoria, b) Distancia (Δz), c) Orientación (θ), d) Velocidad angular (δ_1)

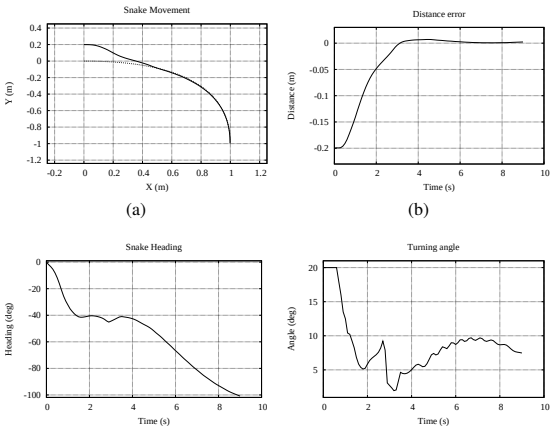

Fig. 6: Prueba de recuperación para la primitiva right(1,1,90). a) Trayectoria, b) Distancia (Δz), c) Orientación (θ), d) Velocidad angular (δ_1)

de unas pocas de éstas. Esto facilita el diseño del planificador y reduce la complejidad de las rutas para el aprendizaje. Las áreas de aprendizaje en robótica y robots multiarticulado se expanden rápidamente y con el tiempo proveerán sistemas autónomos para inspección y mantenimiento. Los resultados obtenidos en este artículo son pequeños pasos para poder obtener esta funcionalidad.

REFERENCES

[1] Lapierre, L., Zapata, R., Lepinay, P.: Combined Path-following and Obstacle Avoidance Control of a Wheeled Robot. The International Journal of Robotics Research, vol. 26:4, pp. 361-375, 2007.

[2] Ashoorirad, M., Barzamini, R., Afshar, A., Jouzdani, J.: Model Reference Adaptive Path Following for Wheeled Mobile Robots. International Conference on Information and Automation (ICIA 2006), pp. 289-294, Dec. 2006.
[3] Campani, M., Capezio, F., Rebora, A., Sgorbissa, A., Zaccaria, R.: A Minimalist Approach to Path Following among Unknown Obstacles. IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS 2010), pp. 3604-3610, Oct. 2010.
[4] Liu, N.: Intelligent Path Following Method for Nonholonomic Robot Using Fuzzy Control. ICINIS'09, Second International Conference on Intelligent Networks and Intelligent Systems, pp. 282-285, Nov. 2009.
[5] Fierro, R., Lewis, F. L.: Control of A Nonholonomic Mobile Robot Using Neural Networks. IEEE Transactions on Neural Networks, vol. 9:4, pp. 589-600, July 1998.
[6] Pivtoraiko, M., Knepper, R. A., Kelly, A.: Differentially Constrained Mobile Robot Motion Planning in State Lattices. Journal of Field Robotics, vol. 26:3, pp. 308-333, March 2009.
[7] Fjerdingen, S. A., Liljebäck, P., Transeth, A. A.: A Snake-Like Robot for Internal Inspection of Complex Pipe Structures. IROS'09 Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems, pp. 5665 - 5671, St. Louis, USA, Oct. 2009.
[8] Altafini, C.: Some properties of the general n-trailer. International Journal of Control, vol. 74:4, pp. 409-424, 2001.
[9] Murugendran, B., Transeth, A. A., Fjerdingen, S. A.: Modeling and Path-Following for a Snake-Robot with Active Wheels. IROS'09 Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems, pp. 3643-3650, St. Louis, USA, Oct. 2009.
[10] Open Dynamics Engine, `http://www.ode.org`
[11] Irrlicht Engine, `http://irrlicht.sourceforge.net`
[12] Mucientes, M., Casillas, J.: Quick Design of Fuzzy Controllers With Good Interpretability in Mobile Robotics. IEEE Transactions on Fuzzy Systems, vol. 15:4, pp. 636-651, Aug. 2007.
[13] Casillas, J.: Embedded genetic learning of highly interpretable fuzzy partitions. Joint 2009 International Fuzzy Systems Association World Congress and 2009 European Society of Fuzzy Logic and Technology Conference (IFSA-EUSFLAT 2009), pp. 1631-1636, Lisbon, Portugal (2009).
[14] Marín, F. J., Casillas, J., Mucientes, M., Transeth, A. A., Fjerdingen, S. A., Schjølberg, I.: Learning Intelligent Controllers for Path-following Skills on Snake-like Robots. International Conference on Intelligent Robotics and Applications, Por Aparecer, Aachen, Germany (2011).