CEA Comité Español de Automática
XLVI JORNADAS AUTOMÁTICA CARTAGENA 2025
UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Integración modular de herramientas gestionadas por LLMs en el robot TIAGo++

Menéndez, Elisabeth*, Garcia-Haro, Juan Miguel, Martínez, Santiago, Balaguer, Carlos.

RoboticsLab, Dpto. de Ingeniería de Sistemas y Automática, Universidad Carlos III de Madrid, Av. De la Universidad 30, 28911 Leganés, España.

To cite this article: Menendez, E., García-Haro, J. M., Martínez, S., Balaguer, C. 2025. Modular integration of LLM-managed tools for the TIAGo++ robot. Jornadas de Automática, 46. https://doi.org/10.17979/ja-cea.2025.46.12229

Resumen

Este artículo presenta una adaptación modular de un sistema basado en grandes modelos de lenguaje (LLMs) al robot TIAGo++, con el objetivo de reutilizar el mismo agente en distintas plataformas robóticas sin modificar su lógica de razonamiento. El sistema interpreta comandos del usuario y utiliza herramientas semánticas organizadas en cuatro categorías: consulta, diagnóstico, expresión y acción. Estas herramientas permiten obtener información del entorno, generar respuestas verbales y ejecutar tareas físicas, todo mediante una interfaz común que facilita su uso y portabilidad. En particular, las herramientas de acción han sido implementadas mediante un servidor de manipulación compatible con MoveIt, que permite ejecutar tareas como entregar objetos, colocarlos cerca de una persona, empujarlos o verter contenido entre recipientes. Cada herramienta proporciona un resultado estructurado que permite al modelo valorar el éxito de la acción y decidir cómo proceder. Esta arquitectura modular facilita la reutilización, la portabilidad y una interacción más eficaz, y además permite la incorporación de nuevas herramientas.

Palabras clave: Manipuladores robóticos, Inteligencia artificial, Interacción humano-robot, Robótica Asistencial, Modelos de lenguaje grandes

Modular integration of LLM-managed tools for the TIAGo++ robot

Abstract

This article presents a modular adaptation of a system based on Large Language models (LLM) to the TIAGo++ robot, with the goal of reusing the same agent across different robotic platforms without modifying its reasoning logic. The agent interprets user commands and employs semantic tools organized into four categories: query, diagnostic, expression and action. These tools enable the system to obtain information from the environment, generate verbal responses and execute physical tasks, all through a common interface that facilitates usability and portability. In particular, the action tools were implemented using a manipulation server compatible with MoveIt, allowing the execution of tasks such as handing over objects, placing them near a person, pushing them, or pouring contents between containers. Each tool provides a structured output allowing the agent to assess the success of the action and decide how to proceed. This modular architecture supports reuse, portability, and more effective interaction, while remaining open to the integration of new tools.

Keywords: Robot manipulators, Artificial intelligence, Human-robot interaction, Assistance robotics, Large Language Models.

1. Introducción

La colaboración humano-robot en entornos asistenciales plantea retos fundamentales de interacción, especialmente cuando los usuarios no son expertos. Para que estas interacciones resulten naturales e intuitivas, los robots deben ser capaces de interpretar comandos expresados en lenguaje natural y adaptarse al entorno físico. Esta necesidad ha motivado la integración de Grandes Modelos de Lenguaje (LLMs, Large

*Autor para correspondencia: emenende@ing.uc3m.es

Language Models) en robótica, como una vía para enriquecer la comprensión de las peticiones del usuario, facilitar el razonamiento, y generar respuestas adecuadas al contexto.

Sin embargo, un robot es un agente físico, situado, que debe comprender su entorno, ejecutar acciones y comunicarse con sus interlocutores de forma fluida. Para ello, las LLMs precisan de mecanismos que permitan vincular el modelo semántico con acciones físicas y perceptivas. Los modelos como ReAct (Yao et al., 2023) han demostrado que los LLMs pueden alternar entre razonamiento y acción mediante herramientas externas (*tools*), que permiten consultar el entorno, generar respuestas verbales o realizar tareas físicas.

Este trabajo se basa en un agente dotado de herramientas definidas para su uso en robots físicos (Tanneberg et al., 2024). Nuestra aportación consiste en adaptar estas herramientas al robot TIAGo++ de forma modular, sin alterar sus interfaces ni su semántica. El objetivo es garantizar que el mismo agente pueda reutilizarse en distintas plataformas. Para ello, se ha diseñado un servidor de manipulación genérico que aprovecha la infraestructura MoveIt, ampliamente utilizada en manipulación robótica. Este diseño permite mantener inalteradas las llamadas de las herramientas, facilitando su portabilidad. Mientras que la integración con el sistema de manipulación requiere adaptaciones específicas, otras herramientas como la consulta del entorno o la generación de habla pueden mantenerse prácticamente invariantes entre robots, simplificando su uso en nuevas plataformas. Este enfoque facilita la portabilidad de agentes basados en LLMs entre diferentes plataformas, reduce el esfuerzo de integración y promueve la reutilización de componentes.

2. Estado del Arte

Los LLMs están revolucionando la robótica, especialmente en el ámbito de los robots manipuladores, al dotarlos de una comprensión avanzada del lenguaje natural y capacidades de razonamiento para tareas complejas. Esta integración es fundamental para traducir intenciones humanas de alto nivel en acciones robóticas de bajo nivel, mejorando la autonomía y el control de la manipulación (Tanneberg et al., 2024; Ahn et al., 2022; Vemprala et al., 2023).

Los LLMs facilitan la interación humano-manipuladores, permitiendo a usuarios programar tareas de brazos robóticos mediante lenguaje natural. Estos modelos pueden interpretar comandos verbales, mapearlos a módulos funcionales y refinar la intención del usuario para dichas tareas. Trabajos como ELLMER (Mon-Williams et al., 2025) demuestran cómo los LLMs, combinados con infraestructura de generación aumentada por recuperación, pueden descomponer tareas complejas en subtareas ejecutables. ELLMER integra retroalimentación visual y de fuerza, vital para la interacción hábil con objetos y la manipulación precisa, permitiendo adaptarse a entornos impredecibles. A pesar de estos avances, persisten desafíos en la integración de LLMs en la manipulación. La escalabilidad y portabilidad de los sistemas de control de manipuladores se abordan mediante arquitecturas modulares y multi-agente, que permiten la descomposición de tareas y la colaboración entre agentes especializados. Esto incluye la capacidad de diseñar y configurar brazos robóticos de manera eficiente (Hwang et al., 2024; Pekarek Rosin et al., 2024).

3. Interacción humano-robot basada en LLMs

El sistema procesa tanto las peticiones del usuario como la información percibida del entorno para decidir cómo actuar en cada situación. Esta toma de decisiones se realiza mediante un agente basado en un LLM, que combina un carácter definido y un conjunto de herramientas funcionales. La arquitectura general se representa en la Figura 1. El agente LLM actúa como núcleo cognitivo del sistema, encargado de interpretar las peticiones del usuario, razonar sobre el contexto y decidir qué acción ejecutar. Para dotarlo de un comportamiento coherente y adaptado al dominio, se define un carácter a través de una *system prompt* que especifica las normas de interacción, el rol del agente (asistente robótico colaborativo), sus objetivos y sus limitaciones. Esta configuración condiciona tanto el tono del lenguaje como su comportamiento, su forma de consultar el entorno y sus respuestas ante fallos o ambigüedades.

Para que el agente LLM pueda interactuar con el mundo físico, se le dota de un conjunto de herramientas semánticas que encapsulan las funcionalidades disponibles en el sistema robótico. Estas herramientas permiten al agente realizar consultas al entorno, expresarse verbalmente o ejecutar acciones físicas. Cada herramienta se define mediante un nombre, una descripción textual, una lista de los argumentos de entrada, y una respuesta semántica para informar al LLM. Esta estructura uniforme permite que el LLM pueda razonar de forma consistente sobre los resultados, detectar errores y tomar decisiones informadas como reintentar una acción, reformular una petición o pedir aclaraciones al usuario. Los tipos de herramientas disponibles en nuestro sistema se agrupan en cuatro categorías:

- Consulta: permiten obtener información sobre los objetos o las personas en la escena.
- Diagnóstico: usadas para mostrar como el LLM interpreta las peticiones del usuario y su entorno.
- Expresión: permiten que el robot comunique sus intenciones y/o acciones.
- Acción: permiten al LLM controlar la manipulación física del robot.

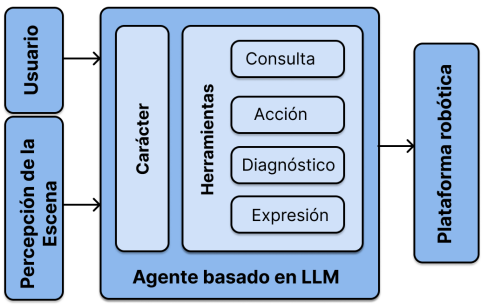

Figura 1: Arquitectura general del sistema basado en un agente LLM.

Este diseño modular permite reutilizar herramientas entre diferentes plataformas robóticas. En particular, las herramientas de consulta y diagnóstico pueden mantenerse invariantes, mientras que las herramientas físicas deben adaptarse a la plataforma utilizando un módulo intermedio.

(a) Integración del agente LLM con TIAGo++

(b) TIAGo++ entregando un objeto a un usuario.

Figura 2: Ejemplo de uso del agente LLM adaptado para el robot TIAGo++.

4. Robot TIAGo++

El TIAGo++ es un robot bimanual de investigación desarrollado por PAL Robotics (Pages et al., 2016). Está basado en la arquitectura modular del robot TIAGo, ampliando sus capacidades mediante dos brazos de 7 grados de libertad que permiten la manipulación bimanual. Este robot dispone de una base móvil de tracción diferencial equipada con un escáner láser y sensores ultrasónicos que facilitan la navegación autónoma en entornos dinámicos. Cada brazo está equipado con pinzas paralelas diseñadas para realizar agarres seguros y estables. Ambos brazos están montados sobre un torso prismático con un recorrido vertical de 35 cm, lo que permite al robot alcanzar objetos a diferentes alturas. La cabeza incorpora un mecanismo de pan-tilt junto con una cámara RGB-D para la percepción de la escena. El sistema funciona sobre el middleware ROS Noetic[1], que proporciona una arquitectura modular, escalable y flexible, facilitando la integración de los distintos componentes y convirtiendo a TIAGo++ en una plataforma versátil para investigación en robótica.

5. Adaptaciones

Para permitir la reutilización del agente LLM en distintas plataformas, las herramientas deben mantener una interfaz uniforme. La Figura 2 muestra cómo se adaptan estas herramientas al robot TIAGo++, mientras que la Tabla 1 resume las herramientas adaptadas, incluyendo sus argumentos y las respuestas generadas para la LLM. Las herramientas de expresión requieren adaptaciones mínimas, mientras que las de acción se encapsulan en un servidor de manipulación basado en MoveIt, lo que permite integrarlas fácilmente en otros robots manipuladores. Esta arquitectura modular facilita la portabilidad sin modificar el razonamiento del agente.

5.1. *Herramientas de expresión*

Como parte de la adaptación del framework original al robot TIAGo++, se ha implementado la herramienta de expresión `speak` usando el servidor nativo *text-to-speech* (TTS) de TIAGo utilizando una acción de ROS (*pal_interaction_msgs/TTsAction*). La herramienta `speak` permite al agente hablar directamente con un usuario específico (o todos los usuarios de la escena) usando lenguaje natural. El texto que el agente desea comunicar se envía al servidor TTS situado en el robot, y el cliente espera a que la acción termine. Esto asegura que el agente puede continuar su razonamiento y actuación solamente una vez que el robot termine de hablar. Como todas las herramientas de este framework, `speak` devuelve un mensaje para informar al LLM el resultado de dicha acción.

Como parte de las herramientas de expresión, se ha implementado la herramienta `look_at`, que permite al robot TIAGo++ mover su cabeza para mirar a un objeto específico en la escena. Esta herramienta se invoca desde el modelo de lenguaje LLM con un comando de la forma `look_at(object_name)` y está implementada como un servicio de ROS que recibe el nombre del objeto como entrada. Al ser llamada, primero solicita al módulo de percepción la posición estimada del objeto. Con esta información, se calculan los ángulos de *pan* y *tilt* que orientan la cabeza hacia el objeto. El movimiento del cuello se realiza mediante el servidor de acciones estándar de ROS, que permite verificar si la acción se completó con éxito para informar al agente si el objeto fue mirado correctamente.

5.2. *Herramientas de acción*

Como parte de esta adaptación, se ha desarrollado un servidor de manipulación de objetos que utiliza MoveIt como *backend* para la planificación y ejecución de movimientos. Este servidor permite ejecutar tareas como entregar objetos, colocarlos sobre superficies, empujar o verter contenido entre recipientes, y ha sido diseñado para ser reutilizable y compatible con cualquier robot que tenga MoveIt correctamente configurado. El servidor toma como entrada un archivo de configuración (Figura 3) donde se definen los manipuladores disponibles, los grupos de planificación (`planning_groups`), los efectores finales (`tip_links`), las tolerancias de orientación y posición para la cinemática inversa (`ik_tolerance`), así como el tiempo máximo de planificación (`planning_timeout`) y el número de intentos (`planning_attempts`). Gracias a es-

[1] `https://wiki.ros.org/noetic`

ta estructura modular, el servidor no está limitado al robot TIAGo++, y puede adaptarse fácilmente a otras plataformas.

Desde el LLM, las herramientas de acción se definen de forma semántica y se comunican mediante una acción de ROS del tipo `ObjectManipulation`. Esta acción incluye la tarea a ejecutar (`task`), uno o dos objetos implicados (`object_for_task_name1`, `object_for_task_name2`), y, en los casos en que la tarea involucra interacción con personas, el identificador del usuario (`user_for_task`). La estructura completa de esta interfaz se muestra en la Figura 3, y las herramientas disponibles se resumen en la Tabla 1. Los argumentos proporcionados por el agente de lenguaje `object_name`, `person_name`, `source_container_name` y `target_container_name`) se asignan directamente a los campos de la acción correspondientes: `object_for_task_name1`, `user_for_task`, `object_for_task_name1` y `object_for_task_name2`.

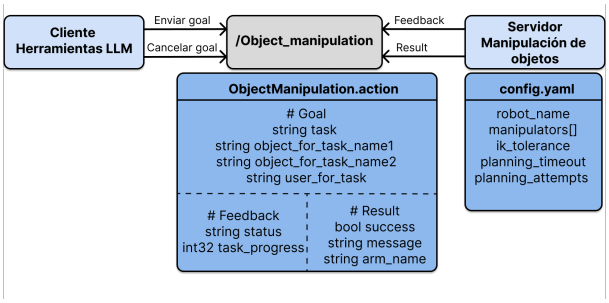

Figura 3: Diagrama de comunicación entre el agente del LLM y el servidor de manipulación. El cliente interpreta las órdenes semánticas y las transforma en peticiones de acción de ROS con sus argumentos correspondientes.

Al finalizar la ejecución, el servidor devuelve al agente un resultado codificado mediante dos campos: `success`, que indica si la tarea ha podido completarse con éxito, y `message`, que proporciona una descripción textual del resultado. Este mensaje se adapta según el contexto, especificando la causa del fallo o confirmando el éxito. Esta retroalimentación es esencial para que el agente del LLM pueda razonar sobre la ejecución, informar al usuario, y decidir si es necesario repetir, adaptar o abortar la acción solicitada.

El servidor de manipulación interpreta esta petición, consulta al módulo de percepción, implementado como un componente independiente, para obtener la posición y forma de los elementos implicados, y selecciona el comportamiento correspondiente según la petición solicitada. Para aquellas tareas que requieren el agarre de objetos, el servidor permite utilizar cualquier sistema externo que genere poses de agarre compatibles con la mano robótica utilizada. En este caso, se emplea un módulo propio desarrollado en un trabajo anterior (Menendez et al., 2024a,b), basado en el ajuste de supercuádricas a las formas estimadas de los objetos. No obstante, este componente es intercambiable, y podrían utilizarse soluciones alternativas basadas en aprendizaje profundo (Morrison et al., 2020) o simuladores como GraspIt (Miller and Allen, 2004), siempre que proporcionen poses de agarre viables para el efector final del robot.

Cada herramienta implementa una lógica de planificación y ejecución distinta, que se adapta a la tarea. A continuación, se describen las principales herramientas implementadas.

5.2.1. *Entrega o colocación de objetos*

Las herramientas `hand_object_over_to_person` y `place_object_near_user` comparten la misma estructura (Algoritmo 1), que combina varias etapas de planificación antes de ejecutar la acción. El sistema evalúa un conjunto de poses de agarre candidatas para el objeto objetivo, y, para cada una, genera de forma anticipada todas las trayectorias necesarias: desde la posición de preagarre hasta la de agarre, de ahí a una posición de retirada, y finalmente hacia las posibles poses de entrega o colocación. Para garantizar un movimiento seguro, se aplica una restricción de orientación vertical durante el trayecto final, evitando giros bruscos que puedan comprometer el contenido del objeto. Si todas las trayectorias planificadas son válidas, la secuencia se ejecuta de manera continua; en caso contrario, se prueba con la siguiente posición candidata. Esta planificación previa minimiza errores en tiempo de ejecución y asegura una transición fluida entre las distintas fases de la tarea.

Algorithm 1 Ejecución de `hand_object_over_to_person` y `place_object_near_user`

Entrada: `object_for_task_name1`, `user_for_task`, `task`
`pos_objeto` ← `obtenerPoseObjeto(nombre_objeto)`
`pos_usuario` ← `obtenerPoseUsuario(nombre_usuario)`
`candidatos_agarre` ← `generarAgarres(nombre_objeto)`
`pose_reposo` ← `obtenerPoseReposo()`
`success` ← `false`
`message` ← `' '`
for all `pos_agarre` en `candidatos_agarre` **do**
  `pos_pre` ← `calcularPreagarre(pos_agarre)`
  `pos_ret` ← `calcularRetirada(pos_agarre)`
  `q_pre` ← `resolverIK(pos_pre)`
  `q_ret` ← `resolverIK(pos_ret)`
  `tray_pre` ← `planificar(q_pre)`
  `tray_aprox` ← `trayectoriaCart(pos_pre, pos_agarre)`
  `tray_ret` ← `trayectoriaCart(pos_agarre, pos_ret)`
  if alguna trayectoria == NULL **then**
    continue
  end if
  `poses_entrega` ← `genEntregas(pos_usuario, pos_ret)`
  `restriccion` ← `crearRestriccionVertical(pos_ret)`
  for all `pos_entrega` en `poses_entrega` **do**
    `q_ent` ← `resolverIK(pos_entrega)`
    if `q_ent` == NULL **then**
      continue
    end if
    `tray_ent` ← `planificar(q_ent, restriccion)`
    `tray_home` ← `planificar(home)`
    if todas las trayectorias son válidas **then**
      `resultado` ← `ejecutarTareaCompleta()`
      if `resultado == fallo_agarre` **then**
        `success` ← `false`
        `message` ← `'No pudiste agarrar el objeto'`
      else if `resultado == fallo_entrega` **then**
        `success` ← `false`
        `message` ← `'No pudiste entregar el objeto'`
      else
        `success` ← `true`
        `message` ← `'Objeto entregado correctamente'`
      end if
      return `success, message`
    end if
  end for
end for
`success` ← `false`
`message` ← `'No pudiste agarrar el objeto'`
return `success, message`

5.2.2. *Vertido de contenido*

La herramienta `pour_into` permite verter el contenido de un recipiente sobre otro (Algoritmo 2). La acción se compone de tres fases: agarre del recipiente, generación de una trayectoria de vertido, y recolocación del recipiente en su posición inicial. Tras agarrar el recipiente, se calcula una trayectoria de vertido en forma de arco, interpolando varias poses que describen un movimiento controlado de rotación sobre el borde del recipiente desde el que se desea verter el contenido.

El borde de vertido se define a partir de un vector de desplazamiento (`offset_borde`) desde la posición del efector hacia el borde físico del recipiente de origen, calculado en función de su geometría estimada y la forma de agarre empleada. Esta forma de agarre hace referencia a la dirección desde la que el recipiente ha sido sujetado. La interpolación se realiza desde el borde de vertido hacia el centro del recipiente objetivo, generando un conjunto de poses orientadas que mantienen el agarre y rotan progresivamente alrededor de un centro de arco común. En el proceso se prueban varios radios del arco en función de la distancia entre los recipientes, seleccionando el primero que permite una trayectoria válida. El sistema ejecuta la trayectoria si se ha podido planificar correctamente sin colisiones. Una vez completado el vertido, se ejecuta la acción `placeObject` para devolver el recipiente a su posición original.

Algorithm 2 Ejecución de `pour_into`

Entrada: `source_container_name`, `target_container_name`
success ← false
message ← ' '
resultado_agarre ← graspObject(source_container_name)
if resultado_agarre == fallo **then**
  message ← 'No pudiste agarrar [source_container_name].'
  return false, message
end if
pos_origen ← obtenerPosEfector()
pos_destino ← obtenerPosObjeto(target_container_name)
centr_dest ← calcCentrDestino(pos_destino)
offs_borde ← calcOffsBorde(pos_origen, source_container_name)
borde_orig ← pose_origen.translation + offs_borde
radio_base ← calcRadioBase(borde_origen, centr_dest)
radios_posibles ← generarRadios(radio_base)
for all rad en radios_posibles **do**
  centr_arco ← calcCentrArc(borde_orig, centr_dest, rad)
  trayectoria ← lista vacía
  for $i = 0$ hasta $N-1$ **do**
    $\theta \leftarrow i \cdot \frac{angulo\_verter}{N-1}$
    pos_borde ← interpEnArco(centr_arco, radio, θ)
    pos_garra ← calcPosGarra(pos_borde, pos_origen)
    trayectoria.append(pos_garra)
  end for
  plan ← resolverTrayectoriaIK(trayectoria)
  if plan != NULL **then**
    break
  end if
end for
if plan == NULL **then**
  message ← 'Pudiste alcanzar [source_container_name]. No alcanzas [target_container_name].'
  return false, message
end if
if ejecutar(plan) == fallo **then**
  message ← 'Pudiste alcanzar [source_container_name]. No alcanzas [target_container_name].'
  return false, message
end if
resultado_colocacion ← placeObject(source_container_name)
success ← true
message ← 'Has vertido [source_container_name] en [target_container_name].'
return success, message

5.2.3. *Empuje de objetos*

La herramienta `push_object_closer_to_person` permite desplazar un objeto sobre una superficie plana acercándolo hacia un usuario determinado (Algoritmo 3). Para ello, se calcula un vector de empuje desde la posición del objeto hasta la del usuario, y se generan varias poses iniciales que permitan al efector contactar lateralmente con el objeto en esa dirección. Una vez seleccionada una pose de contacto válida mediante cinemática inversa, el sistema planifica de forma anticipada todas las trayectorias necesarias: aproximación al punto de contacto, desplazamiento cartesiano en la dirección de empuje y retirada del efector tras el movimiento. Solo si todas estas trayectorias son válidas, se procede a la ejecución del plan. Este enfoque garantiza que el empuje se realice de forma estable y en línea recta, minimizando deslizamientos inesperados y manteniendo siempre el contacto con el objeto.

Algorithm 3 Ejecución de `push_object_closer_to_person`

Require: nombre_objeto, nombre_usuario
success ← false
message ←
pos_objeto ← obtenerPoseObjeto(nombre_objeto)
pos_usuario ← obtenerPoseUsuario(nombre_usuario)
dir_empuje ← calcularDireccionEmpuje(pos_objeto, pos_usuario)
poses_inicio ← generarPosesContacto(pose_objeto, direccion_empuje)
for all pose_inicio en poses_inicio **do**
  q_inicio ← resolverIK(pose_inicio)
  if q_inicio == NULL **then**
    continue
  end if
  tray_aprox ← planificar(q_inicio)
  tray_empuje ← planTrayCart(pos_inicio, dir_empuje)
  tray_retirada ← calcRetiTrasEmpuje(pos_inicio)
  tray_reposo ← planificar(home)
  if todas las trayectorias son válidas **then**
    Guardar trayectorias
    plan_encontrado ← true
    break
  end if
end for
if plan_encontrado **then**
  resultado ← ejecutarTareaEmpujeCompleta()
  if resultado == fallo **then**
    success ← false
    message ← 'No se pudo empujar el objeto'
  else
    success ← true
    message ← 'Objeto empujado correctamente'
  end if
else
  success ← false
  message ← 'No se pudo empujar el objeto'
end if
return success, message

6. Conclusiones y trabajos futuros

Este trabajo ha presentado una arquitectura modular que permite integrar modelos de lenguaje (LLMs) con herramientas de interacción y acción en el robot TIAGo++. La propuesta mantiene la lógica del sistema original, facilitando la reutilización de las herramientas en distintas plataformas sin necesidad de modificaciones. Gracias a una interfaz común, el sistema puede ejecutar tareas físicas como entregar, empujar o verter objetos, manteniendo una interacción flexible y robusta con el entorno. La implementación en TIAGo++ demuestra que es posible encapsular funcionalidades físicas bajo interfaces semánticas, preservando la expresividad y capacidad de razonamiento del modelo. Como líneas futuras, se plantea ampliar el repertorio de herramientas para cubrir un mayor número de tareas manipulativas, y explorar la integración de entradas multimodales como la mirada o los gestos para mejorar la comprensión contextual. Además, se evaluará la arquitectura en diversos escenarios colaborativos y diferentes plataformas robóticas.

Agradecimientos

La investigación que ha conducido a estos resultados ha recibido financiación del proyecto iRoboCity2030-CM, Robótica inteligente para ciudades sostenibles (TEC-2024/TEC-62), financiado por Programas de Actividades I+D en tecnologías de la Comunidad de Madrid, y del proyecto ADAPTA, con referencia PLEC2023- 010218, financiado por MICIU /AEI /10.13039/501100011033.

Referencias

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., et al., 2022. Do as i can, not as i say: Grounding language in robotic affordances. arXiv preprint arXiv:2204.01691.
DOI: 10.48550/arXiv.2204.01691

Hwang, Y., Sato, A. J., Praveena, P., White, N. T., Mutlu, B., 2024. Understanding generative ai in robot logic parametrization. arXiv preprint arXiv:2411.04273.
DOI: 10.48550/arXiv.2411.04273

Menendez, E., Martínez, S., Balaguer, C., 2024a. Selección y agarre robótico de objetos basada en el seguimiento de la mirada. In: Actas del Simposio de Robótica, Bioingeniería y Visión por Computador: Badajoz, 29 a 31 de mayo de 2024. Servicio de Publicaciones, pp. 127–132.

Menendez, E., Martínez, S., Díaz-de María, F., Balaguer, C., 2024b. Integrating egocentric and robotic vision for object identification using siamese networks and superquadric estimations in partial occlusion scenarios. Biomimetics 9 (2), 100.
DOI: 10.3390/biomimetics9020100

Miller, A. T., Allen, P. K., 2004. Graspit! a versatile simulator for robotic grasping. IEEE Robotics & Automation Magazine 11 (4), 110–122.
DOI: 10.1109/MRA.2004.1371616

Mon-Williams, R., Li, G., Long, R., Du, W., Lucas, C. G., 2025. Embodied large language models enable robots to complete complex tasks in unpredictable environments. Nature Machine Intelligence, 1–10.
DOI: 10.1038/s42256-025-01005-x

Morrison, D., Corke, P., Leitner, J., 2020. Learning robust, real-time, reactive robotic grasping. The International journal of robotics research 39 (2-3), 183–201.
DOI: 10.1177/0278364919859066

Pages, J., Marchionni, L., Ferro, F., 2016. Tiago: the modular robot that adapts to different research needs. In: International workshop on robot modularity, IROS. Vol. 290.

Pekarek Rosin, T., Hassouna, V., Sun, X., Krohm, L., Kordt, H.-L., Beetz, M., Wermter, S., 2024. A framework for adapting human-robot interaction to diverse user groups. In: International Conference on Social Robotics. Springer, pp. 24–38.
DOI: 10.1007/978-981-96-3525-2_3

Tanneberg, D., Ocker, F., Hasler, S., Deigmoeller, J., Belardinelli, A., Wang, C., Wersing, H., Sendhoff, B., Gienger, M., 2024. To help or not to help: Llm-based attentive support for human-robot group interactions. In: 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 9130–9137.
DOI: 10.1109/IROS58592.2024.10801517

Vemprala, S., Bonatti, R., Bucker, A., Kapoor, A., 2023. Chatgpt for robotics: Design principles and model abilities. 2023. Published by Microsoft.
DOI: 10.1109/ACCESS.2024.3387941

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y., 2023. React: Synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR).

Tabla 1: Resumen de herramientas adaptadas, argumentos y respuestas

Herramienta	Descripción	Argumentos	Respuestas posibles
Herramientas de expresión			
speak	El robot comunica un mensaje al usuario.	**person_name**: Nombre de la persona. **text**: Texto a comunicar.	"Dijiste a [person_name]: [text]"
look_at	El robot gira el cuello para mirar un objeto.	**object_name**: Nombre del objeto.	"Has mirado a [object_name]." o "No pudiste mirar a [object_name]."
Herramientas de acción			
hand_object_over_to_person	Entrega un objeto a una persona.	**object_name**: Nombre del objeto. **person_name**: Nombre de la persona.	"No pudiste agarrar el objeto [object_name]." / "Pudiste alcanzar el objeto [object_name], pero no al usuario [person_name]." / "Has entregado [object_name] a [person_name]."
move_object_to_person	Coloca un objeto junto a una persona.	**object_name**: Nombre del objeto. **person_name**: Nombre de la persona.	"No pudiste agarrar el objeto [object_name]." / "Pudiste alcanzar el objeto [object_name], pero no al usuario [person_name]." / "Has colocado [object_name] junto a [person_name]."
pour_into	Vierte el contenido de un recipiente en otro.	**source_container_name**: Nombre del contenedor de origen. **target_container_name**: Nombre del contenedor de destino	"No pudiste agarrar el contenedor [source_container_name]." / "Pudiste alcanzar el contenedor [source_container_name], pero no al objetivo [target_container_name]." / "Has vertido [source_container_name] en [target_container_name]."
push_object_closer_to_person	Empuja un objeto hacia una persona.	**object_name**: Nombre del objeto. **person_name**: Nombre de la persona.	"No pudiste empujar el objeto [object_name]." / "Has empujado [object_name] hacia [person_name]."