Deusto
Universidad de Deusto

Proyecto Final

Robótica y Automatización Inteligente

Grupo 3
Gonzalo Maldonado Arana
Itsasne Presumido Martínez-Conde
Jesús Sevillano
Gabriel Pulgar
Unai Laconcha
Gorka Elorriaga
Vanessa Elizabeth Mejía Fajardo
Mishelle Quinchiguango

NOTA

Debido a limitaciones de tamaño en Alud, no ha sido posible subir todo el entorno de trabajo. Sin embargo, **en el documento** se incluye el **enlace de acceso al repositorio en Git** donde se encuentra todo el contenido.

Índice

Introducción

En el actual escenario de la automatización robótica, la convergencia de técnicas de inteligencia artificial, visión por computador y sistemas de control distribuido está abriendo nuevas oportunidades para el desarrollo de aplicaciones complejas y adaptativas. Este proyecto se sitúa en ese contexto, proponiendo la creación de un sistema robótico integral que permita la reconstrucción de figuras empleando cubos de colores, operable en modos automático y manual.

La solución se apoya en el uso de ROS (Robot Operating System) como middleware, aprovechando su arquitectura modular para gestionar de forma distribuida las tareas críticas inherentes al procesamiento intensivo de imágenes y a la ejecución en tiempo real de algoritmos de control. En este sistema, ROS actúa como plataforma de integración, coordinando nodos especializados que, a través de protocolos como TCPROS, facilitan la comunicación entre módulos de visión, planificación y control. Aunque ROS 1 no garantiza requisitos estrictos de tiempo real, el diseño modular y la distribución de la carga computacional permiten alcanzar un rendimiento óptimo en escenarios dinámicos.

- **Reinforcement Learning:** Para la generación y ajuste de trayectorias en tiempo real, adaptándose dinámicamente a las variaciones del entorno.
- **Visión por Computador:** Para la detección, clasificación y localización precisa de los cubos, proporcionando la información sensorial necesaria para la toma de decisiones.
- **Robótica y Control mediante ROS:** Para orquestar la comunicación entre nodos en una arquitectura distribuida, facilitando la coordinación, supervisión y control remoto del brazo robótico.

A través de esta integración multidisciplinaria se busca experimentar acerca de cómo distintas áreas de la IA pueden converger para realizar tareas complejas en el ámbito de la robótica moderna, explorando nuevas estrategias que potencien la adaptabilidad y el aprendizaje en tiempo real. De igual manera, a partir de esta integración, es posible optimizar la manipulación del brazo robótico facilitando así su uso y su operatividad en múltiples contextos.

Repositorio y Espacio de Trabajo

GitHub

Para mantener un seguimiento de las versiones del proyecto y unir el trabajo de ambos grupos, se ha creado un repositorio de github con el *workspace* de ROS y las imágenes de docker. En el siguiente enlace se encuentra el repositorio:

https://github.com/gonzalo002/robotica_g3_mucsi

Imagen de Docker Personalizada

Además, en este repositorio se encuentran las dos imágenes de docker personalizadas (*grupo_3_dektop* y *grupo_3_local_gpu*) con las librerías necesarias para la realización del proyecto. Esta imagen de docker se ha desarrollado utilizando como base la imagen aportada en clase.

Las imágenes desarrolladas permiten trabajar tanto de forma local con uso de GPU, ejecutable únicamente en Ubuntu, como a través de la versión desktop, la cual permite su uso a través de WSL de Windows pero sin conexión al robot físico.

Las principales modificaciones realizadas sobre la imagen son la inclusión de las siguientes librerías de Python y paquetes de ROS Noetic.

- **Paquetes de ROS**
  - ros-noetic-camera-calibration
- **Librerías de Python**
  - gymnasium==0.29.1
  - stable-baselines3[extra]==2.2.1
  - numpy --upgrade
  - scipy
  - opencv-contrib-python
  - mediapipe
  - tk

Desarrollo del Proyecto

¿En qué consiste?

El proyecto consiste en el desarrollo de un sistema distribuido para el control de un brazo robótico colaborativo capaz de realizar tareas de detección y reconstrucción de figuras utilizando cubos de colores.

Este sistema combina visión artificial, algoritmos de aprendizaje por refuerzo y control robótico, permitiendo al robot operar en modalidad automática o manual para completar la tarea asignada.

- **Funcionamiento automático:** El robot opera de manera autónoma sin intervención manual directa, procesando y reconstruyendo la figura definida.
- **Funcionamiento manual:** La persona interactúa directamente sobre la posición del robot y el estado de la garra mediante la detección de la mano y sus gestos.

Despliegue del Sistema

El despliegue del proyecto contará con los siguientes elementos:

- **UR3:** Robot colaborativo de 6 ejes de la marca Universal Robots.
- **Cámaras:** Contaremos con un total de 5 cámaras USB, 3 para la detección de la figura y los cubos (*cam_perfil*, *cam_alzado* y *cam_planta*) y otras 2 para la detección de la posición de la mano (*hand_top* y *hand_lateral*).
- **Cubos de colores:** Los utilizaremos para realizar la figura con un límite máximo de 25 cubos por figura.
- **Sistema de control distribuido:** Formado por 3 ordenadores, conectados a través de una topología de árbol, donde existen un maestro (visualización y control) y dos esclavos (captura y procesamiento de imagen, lanzamiento del controlador del robot y ejecución y control de servicios/topics)

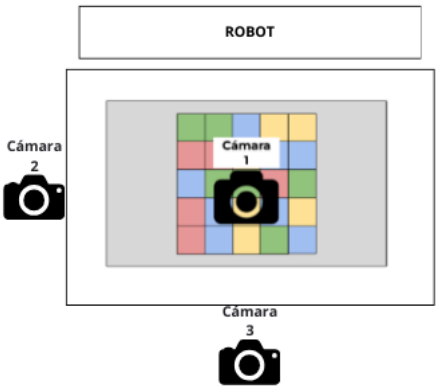


Imagen 1. Representación del primer espacio de trabajo del robot junto con las cámaras y cubos.

Imagen 2. Representación del segundo espacio de trabajo con las cámaras para la detección de la posición de la mano.

Arquitectura Física

La arquitectura física se observa de la siguiente forma:

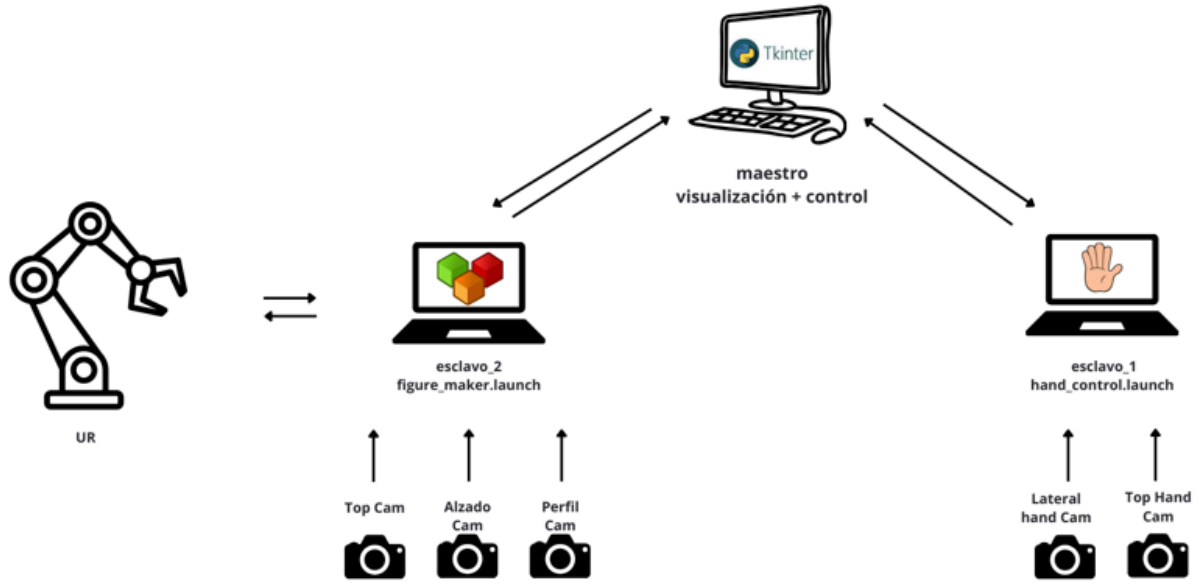


Imagen 3. Diagrama general de la arquitectura.

En la imagen se observa la arquitectura distribuida del sistema, compuesta por tres elementos principales: el nodo maestro y dos ordenadores esclavos (esclavo 1 y esclavo 2). Cada uno cumple funciones específicas que, en conjunto, permiten un procesamiento eficiente y una coordinación óptima del brazo robótico.

- **Esclavo 1 (Procesamiento de la Mano):** Este ordenador procesa las imágenes capturadas por las cámaras 1 y 2, e implementa algoritmos de visión artificial para extraer las coordenadas XYZ y reconocer los gestos de la mano. La información procesada por este nodo se envía a través de un topic para facilitar el control del robot en tiempo real.

- **Esclavo 2 (Procesamiento de Imagen, RL y controlador del Robot)**: Paralelamente, el ordenador esclavo 2 se encarga de recibir y procesar la información visual de las cámaras 3, 4 y 5, para procesar la figura a reconstruir y localizar los cubos esparcidos en el espacio de trabajo del robot; a su vez, también se ejecuta el nodo de aprendizaje por refuerzo, encargado de comprobar la viabilidad de las cogidas de los cubos y el orden de recogida más óptimo. A su vez, este ordenador es el encargado de lanzar el *launcher* del robot, procesando todas las comunicaciones entre el robot y el resto de usuarios de la red.

- **Maestro:** El maestro centraliza la coordinación y supervisión del sistema. Mantiene una comunicación bidireccional con los esclavos mediante direcciones IP asignadas, lo que permite la transferencia de datos y comandos en tiempo real. Este nodo se encarga de la visualización de los procesos en ejecución y del control global de los movimientos del brazo robótico, garantizando una operación remota y segura sin requerir la presencia física en el sitio.

Infraestructura de Conexión y Comunicación:

- Las cámaras (1 a 5) se conectan mediante cableado físico directamente a los ordenadores esclavos, asignándose cada una a la función pertinente de captura de imágenes (por ejemplo, cámaras 1 y 2 para detección de gestos, y cámaras 3, 4 y 5 para el seguimiento y localización de cubos).
- Esclavo_2 está conectado al robot a través de un cable de red, lo que asegura una transmisión de datos rápida y estable para la ejecución de comandos críticos de movimiento.
- La comunicación entre el nodo maestro y los esclavos se establece mediante sus respectivas direcciones IP, utilizando el protocolo de comunicación propio de ROS TCPROS, el cual permite una comunicación fiable y a tiempo real.

La decisión de implementar una arquitectura distribuida se fundamenta en la necesidad de mitigar la carga computacional elevada derivada del procesamiento intensivo de imágenes y del control en tiempo real. Al distribuir las tareas críticas entre varios ordenadores, se mejora significativamente el rendimiento global del sistema, se reduce la latencia en la comunicación y se habilita la operación remota, permitiendo gestionar el brazo robótico desde una ubicación distinta a la del hardware. Esta distribución de recursos es clave para asegurar una respuesta rápida y precisa en aplicaciones de manipulación y ensamblaje que requieren un procesamiento simultáneo de múltiples flujos de datos.

Estructura del Sistema

Para alcanzar nuestro objetivo implementaremos el sistema distribuido aprovechando las capacidades de ROS, que proporciona una arquitectura modular basada en nodos.

- **Captura y Procesamiento de Imágenes**
  - **Nodo “hand_detector_node”:** Recibe y procesa las imágenes captadas por las cámaras para controlar el robot mediante gestos manuales. La cámara superior llamada top_hand_cam identifica la posición de la mano e interpola su ubicación en la imagen para determinar las coordenadas del robot en los ejes X e Y, además de detectar 5 gestos específicos los cuales indican tanto el inicio y el final de la captura de la pose como diferentes acciones con el robot, tales como abrir y cerrar la pinza y volver a posición inicial. Simultáneamente, la cámara lateral llamada lateral_hand_cam localiza la mano y se encarga de definir exclusivamente la posición de la mano en el eje Z. Toda la información obtenida de ambas cámaras se guarda en un mensaje personalizado de ROS el cual se llama hand_data y se transmite mediante un topic al nodo encargado del control del robot.
  - **Nodo “cube_tracker_node”:** Recibe y procesa las imágenes de la cámara superior llamada top_cam del área de trabajo, para la detección de la posición de los cubos en el plano del robot. Posteriormente la figura es desarmada, esparciendo los cubos a través del espacio de trabajo del robot, detectando su posición, color y rotación a por medio de la cámara superior, almacenandose en un mensaje personalizado de ROS. Una vez conocida la localización actual de los cubos, se envían los datos a través de un topic al nodo de *Reinforcement Learning*.

  - **Nodo "figure_maker_node":** Recibe y procesa las imágenes de las cámaras para la detección de cubos y figuras. En este caso, a través de la cámara superior y las frontales, se analiza la figura creada, detectando la posición y el color de los cubos visibles, almacenando esta información en matrices de 5x5.

- **Toma de Decisiones con Aprendizaje por Refuerzo**
  - **Nodo RL:** Implementa un entorno y un agente de aprendizaje por refuerzo el cual tiene como objetivo maximizar la recompensa, en este caso negativa, la cual está asociada al tiempo necesario para la recogida de todos los cubos necesarios para la construcción de la figura descrita. Este nodo recibe a través de un topic los datos de la localización, rotación y color de los cubos, y devuelve tanto el orden de los cubos a coger como las trayectorias calculadas en el proceso.

- **Control y Ejecución del robot**
  - **Nodo Control_robot:** Recibe la consigna de posición de la punta del robot e información acerca del estado de la mano y el gesto que ésta realiza por los nodos previamente mencionados, y en base a esta ejecuta las acciones requeridas para que el robot cumpla con la tarea asignada.

Todos los nodos descritos en el son ejecutados a través de launch files de ROS, los cuales permiten lanzar nodos, habilitar comunicaciones con elementos físicos como las cámaras o ejecutar la escena de planificación del robot. Estos ejecutables se lanzan a través de una única línea de comando a través de terminal, lo que facilita su manejo.

Para este proyecto se han definido los siguientes *launch files*.

- **figure_maker.launch:** Inicializa los topics de las cámaras de alzado, planta y perfil, ejecuta el *launcher* del robot y activa los nodos relacionados a la reconstrucción de la figura, cube_tracker, figure_maker y rl.
  - **Parámetros**:
    - `cam_view bool: Inicializa el nodo figure_maker con o sin las cámaras`
    - `cam_tracker bool: Inicializa el nodo cube_tracker con o sin las cámaras`
    - `rob_sim bool: Lanza el robot real o su simulación.`
- **hand_control.launch:** Inicializa los topics de las camaras hand_top y hand_lateral, así como el nodo asociado a la detección de la posición de la mano y sus gestos.

Para la comunicación entre nodos, se han desarrollado comunicaciones en base a acciones y topics, las cuales adquieren, procesan y envían la información necesaria para la realización del proceso.

Los protocolos definidos son los siguientes:

- **Acción *FigureMaker:***
  - **Request**
    - `order int8`
  - **Result**
    - `figure_3d int8[]`
    - `shape_3d int8[]`
  - **Feedback**

```
- feedback int8
```

- **Funcionamiento:** Adquiere y procesa las imágenes de las cámaras asociadas a la detección de figuras, obteniendo la figura 3d a reconstruir.

- Acción *CubeTracker:*
  - **Request**
    ```
    - order int8
    ```
  - **Result**
    ```
    - idcubos[]
    - cube_position
    - color_counter int8[]
    ```
  - **Feedback**
    ```
    - int8 feedback
    ```
  - **Funcionamiento:** Recibe la información de los cubos necesarios para crear la figura y procesa la imagen para encontrar la posición y el color de los cubos dispersos en el área de trabajo.

- Acción *RL:*
  - **Request**
    ```
    - cubes_position IdCubos[]
    - cubes_order int8[]
    ```
  - **Result**
    ```
    - cubes_correct_order IdCubos[]
    - cubes_trajectories trajectory_msgs/JointTrajectory[]
    ```
  - **Feedback**
    ```
    - feedback int8
    ```
  - **Funcionamiento:** Recibe los cubos localizados y el orden de la figura a realizar, y devuelve los cubos ordenados y las trayectorias hasta los cubos calculados.

- Topic *HandDetector:*
  - **Mensaje (HandData)**
    ```
    - float32 X
    - float32 Y
    - float32 Z
    - bool hand_detected
    - bool is_peace
    - bool is_dino
    - bool is_dislike
    - bool is_open
    ```
  - **Frecuencia de Publicación**
    ```
    - 10Hz
    ```
  - **Funcionamiento:** Adquiere imágenes de las dos cámaras asociadas al proceso, las procesa y devuelve la información necesaria sobre la mano, desde su posición hasta el gesto que está realizando.

A su vez, tal y como se ha visto en algunos de los protocolos anteriores, se han definido mensajes propios para facilitar el manejo de cierta información relevante para el proceso. mensajes definidos son los siguientes:

- **IdCubos:**
  - **Mensaje**
    - `int8 id`
    - `int8 color`
    - `geometry_msgs/Pose pose`

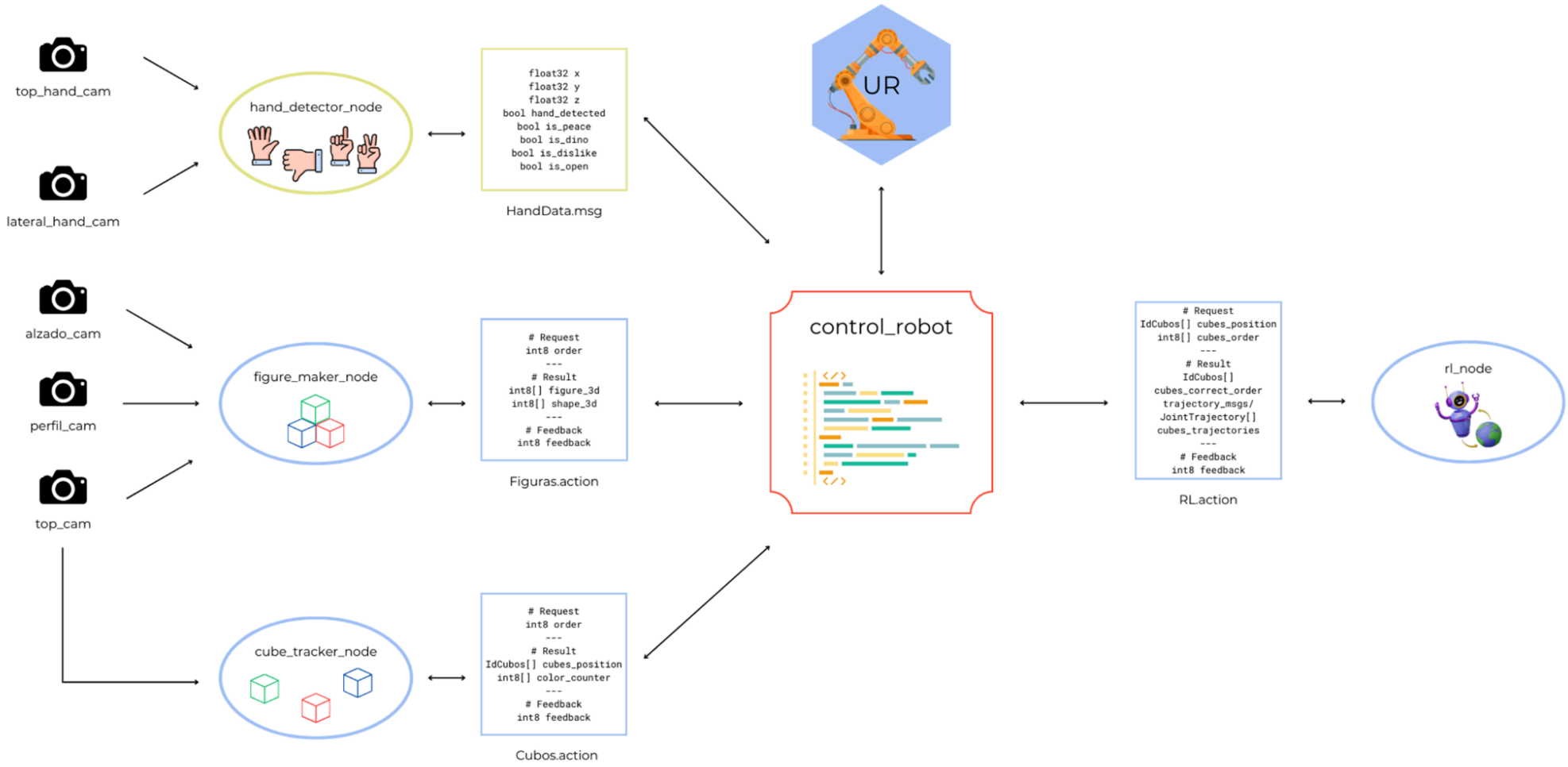


Imagen 4. Diagrama de la arquitectura de nodos de ROS del proyecto

Manejo de la Aplicación

La aplicación desarrollada se divide en dos partes, siendo estas el control manual, donde se mueve el robot en base a la detección de la pose y los gestos de una mano con dos cámaras, y el control automático, donde se ejecuta el proceso completo de procesamiento y reconstrucción de la figura deseada.

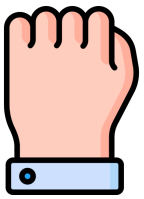
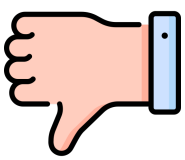
Toda la aplicación es controlada a través de una interfaz gráfica basada en la librería tkinter de python, desde la cual se puede seleccionar el modo de funcionamiento, así como lanzar cualquiera de los procesos mencionados en esta memoria.

- **Control manual**

  En el modo de funcionamiento manual, en el que se replican los movimientos que realiza la mano de un humano, el nodo "hand_detector_node" (que se ejecuta en el esclavo 1) genera las consignas de pose para el TCP. Las cámaras "Lateral hand Cam" y "Top Hand Cam" están conectadas vía cable USB a dicha máquina, permitiendo que el nodo lea directamente las imágenes de las cámaras y realice el procesamiento y análisis pertinentes.

  Una vez calculada la posición y el gesto de la mano, pública en el *topic* "hand_data" al que el nodo "Control_robot" está suscrito. El mensaje, cuyo tipado ya ha quedado reflejado en la tabla de mensajes de los *topics*, comprende una consigna de posición para el robot ({x,y,z}) e información sobre el gesto que el usuario está haciendo ({is_peace,is_dino,is_dislike,is_open}), así como la detección de mano o ausencia de ella (hand_detected).

  Las acciones asociadas a los gestos son las siguientes:

MANO ABIERTA Captura de las posiciones de la mano 	**MANO CERRADA** Dejar de capturar las posiciones de la mano 	**GESTO PAZ** Cierra la pinza 
GESTO DISLIKE Regresa el robot a la pose inicial 	**GESTO DINO** Abre la pinza 	**MANO NO DETECTADA** Espera a detectar una mano 

En modo manual, el nodo "control_robot" calcula el desplazamiento de la mano en relación al espacio de control (en el que el operario tiene la mano) y lo re-escala al espacio del robot. Para cualquiera de las 3 dimensiones del espacio, la operación para calcular el movimiento del robot es la siguiente:

$$\Delta = (\frac{Pos}{Max_{mano}})Max_{robot}0'1$$

Aplicada a cada una de las dimensiones, los cálculos son como sigue. X se calcula en negativo porque la perspectiva del eje está invertida en el espacio de control.

$$\Delta x = -(\frac{x}{max(X_{mano})})max(X_{robot})0'1$$

$$\Delta y = (\frac{y}{max(Y_{mano})})max(Y_{robot})0'1$$

$$\Delta z = (\frac{z}{max(Z_{mano})})max(Z_{robot})0'1$$

Los tramos de movimiento están ponderados con un valor de 0.1, dado que con factores mayores el control del robot a través de la detección de la mano aumenta la complejidad del control. A su vez, el uso de tramos más amplios requiere de mayor tiempo de ejecución por parte del controlador del robot, lo que provoca que el control y el movimiento sea menos fluido, distando cada vez más de un control en tiempo real.

Para efectuar el movimiento, se obtiene la pose actual del robot y se le aplica la diferencia a la dimensión correspondiente y se ordena que el brazo adopte la nueva pose. Puesto que la pinza siempre está orientada hacia abajo, solo la posición de esta puede cambiar.

El método incluye contingencias y avisos en caso de que se exceda el rango de trabajo del robot en alguna dimensión o la pose objetivo sea inalcanzable. Estos rangos se basan en el área de trabajo del robot, y se definen para reducir la posibilidad de aparición de singularidades o posiciones cuya resolución sea compleja.

- **Control automático**

En el control automático el flujo de trabajo comienza desde que se selecciona la opción de automático desde la interfaz del Tkinter, iniciando el robot en un estado de espera. Este proceso se controla en su totalidad a través de la interfaz, guiando la secuencia y visualizando mensajes para facilitar la correcta ejecución del proceso a cualquier usuario.

Para comenzar con la reconstrucción de la figura, el primer paso es capturar sus características a través de la acción asociada a las Figuras, la cual se encarga de almacenar y procesar las imágenes de las 3 cámaras enlazadas a este proceso. El resultado final obtenido es una matriz tridimensional con los datos de la figura.

Una vez obtenida la figura, para la transformación de las coordenadas de los cubos desde la posición en la imagen, hasta su pose respecto al robot, se realiza una calibración manual utilizando un ArUco como referencia entre ambos sistemas de coordenadas. Este

proceso se compone de 5 pasos, todos controlados a través de la interfaz, donde se debe colocar un puntero en la garra del robot, posteriormente llevarlo hasta el origen de coordenadas del Aruco, donde se almacena la pose del robot en ese punto, y finalmente retirar el puntero para permitir la continuación del proceso de reconstrucción de figuras.

Hay que tener en cuenta, que este proceso no va a funcionar si previamente no se ha obtenido la calibración de la lente de la cámara, para así corregir la distorsión de la imagen. En este caso, este proceso se ha realizado a través del paquete propio de ROS, y los datos de la cámara se encuentran almacenados en su correspondiente apartado del proyecto.

Finalmente se esparcen los cubos por el espacio de trabajo del robot, donde a través de la acción *CubeTracker* se localizan las poses de todos los cubos. Con estas posiciones, primeramente se libera el espacio donde se va a construir nuevamente la figura original, dejando los cubos descartados en áreas donde no interfieran con el proceso de reconstrucción.

A su vez, para poder seleccionar un orden de recogida, se ha desarrollado otro nodo basado en algoritmos de aprendizaje por refuerzo el cual selecciona el orden más óptimo para la recolección de cubos, además de asegurar la capacidad del robot de recoger los cubos.

Con la descripción de la figura y el orden de los cubos, se recogen los cubos y se colocan en la posición correspondiente dando forma a la figura original, pero esta vez construida de forma completamente autónoma.

Operatividad

Para la operación del sistema, se deben manipular los 3 ordenadores. Si bien, la gestión principal se realiza mediante el máster, en los otros 2 ordenadores en paralelo deben ser ejecutados los script que habilitan los launch de figure_maker y de *hand_control* tal como se describe detalladamente en el repositorio de github.

Dicho eso, es que a través del ordenador máster, se puede ejecutar el script que inicializa la interfaz del Tkinter que a su vez abre el entorno de visualización RViz el cual permite observar las trayectorias que el brazo estaría planificando y su entorno de trabajo. Esto facilita la integración de la arquitectura.

En la parte superior izquierda se permite distinguir el modo de funcionamiento a través de un botón donde se selecciona el modo deseado. En la misma parte superior, se añade un botón de “STOP” el cual envía un mensaje a control robot para detener el movimiento del brazo robótico. Asimismo, en paralelo al Tkinter se

Al seleccionar el botón del modo automático, se observan 3 botones: calibrar, detectar figura y armar figura, además de dos imágenes a la izquierda y la derecha las cuales entregan una representación de la figura inicial a reconstruir en 3d y una captura de los cubos en posición desordenada.

En un comienzo, sólo está disponible la opción de calibrar, ya que es requisito para las otras dos acciones. Seleccionada esta acción, el robot se calibra a través de estos pasos:

- Detectar ArUco
- Insertar bolígrafo en la pinza del brazo.
- Mover el bolígrafo a la esquina del Aruco.
- Pose alcanzada
- Soltar el bolígrafo

Una vez calibrado, se habilitan las acciones de detectar y armar figura. Tal como mencionan los nombres, mediante detectar figura se ejecuta la captura mediante las 3 cámaras (top_cam, perfil_cam y alzado_cam) y se envía el mensaje con la figura en forma de matriz al nodo figure_maker .

También, está la acción de armar la figura, botón que permite reconstruir la figura identificada inicialmente a través del nodo de para reconstruir la figura a partir de estos pasos:

- Con los cubos desordenados en el mesón, el brazo robótico despeja el área de trabajo y selecciona los cubos que interrumpen el espacio para la figura final y los agrupa en la periferia por color.
- Luego el brazo robótica ejecuta cada una de las trayectorias óptimas recibidas desde el nodo RL_action_server a través del mensaje.
- Una vez terminados los desplazamientos el brazo se mueve a la posición de home.

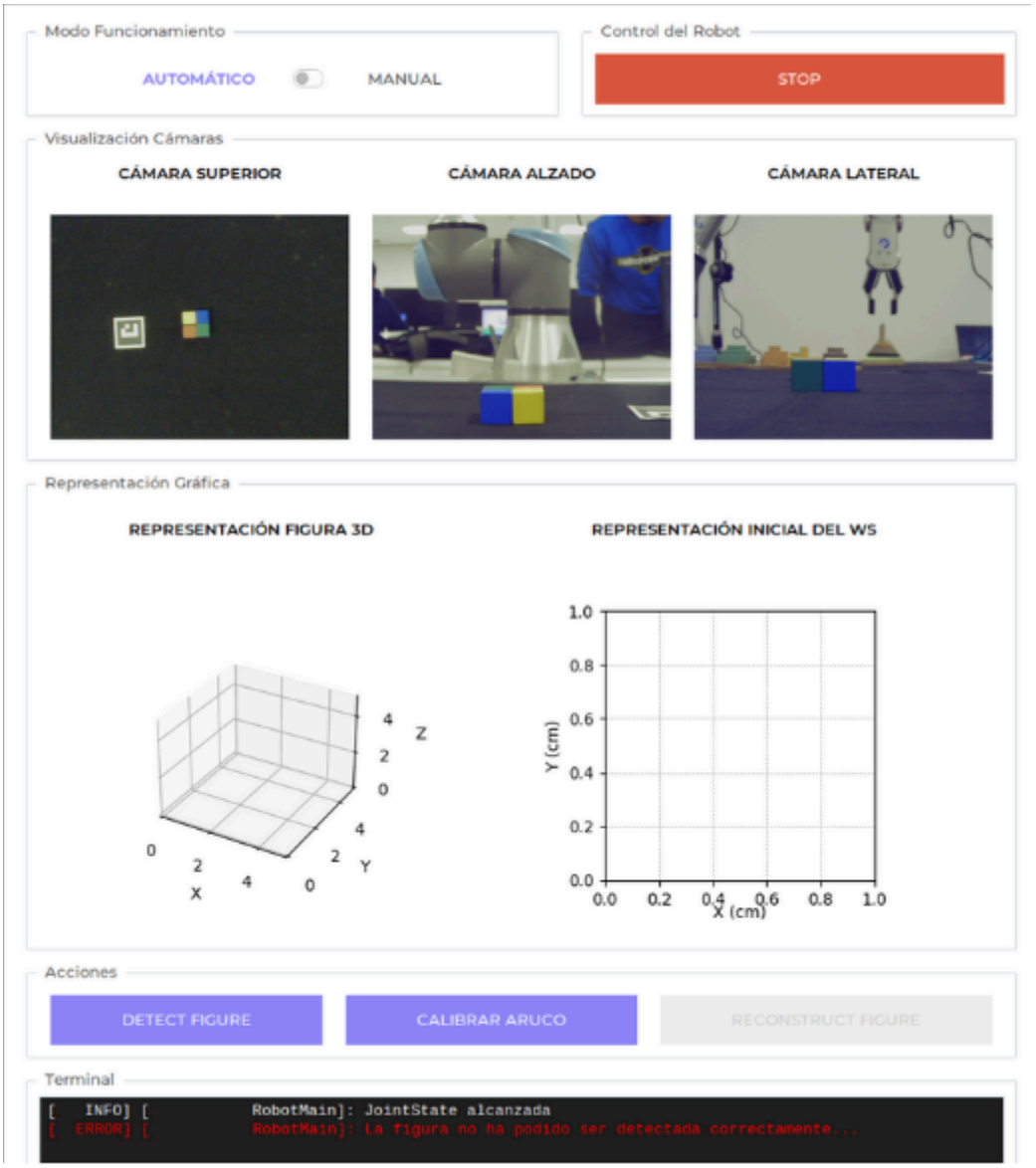


Imagen 5. Modo de funcionamiento automático en Tkinter.

Por otra parte, está el botón manual, el cual una vez seleccionado envía el brazo a una pose inicial definida para este modo.

En cuanto a la interfaz del Tkinter, se observan las capturas de las 3 cámaras, y un espacio 3d con un punto que representa la posición de la mano en las tres dimensiones..

Además, existe un recuadro que devuelve el gesto encontrado a través de una animación del gesto detectado.

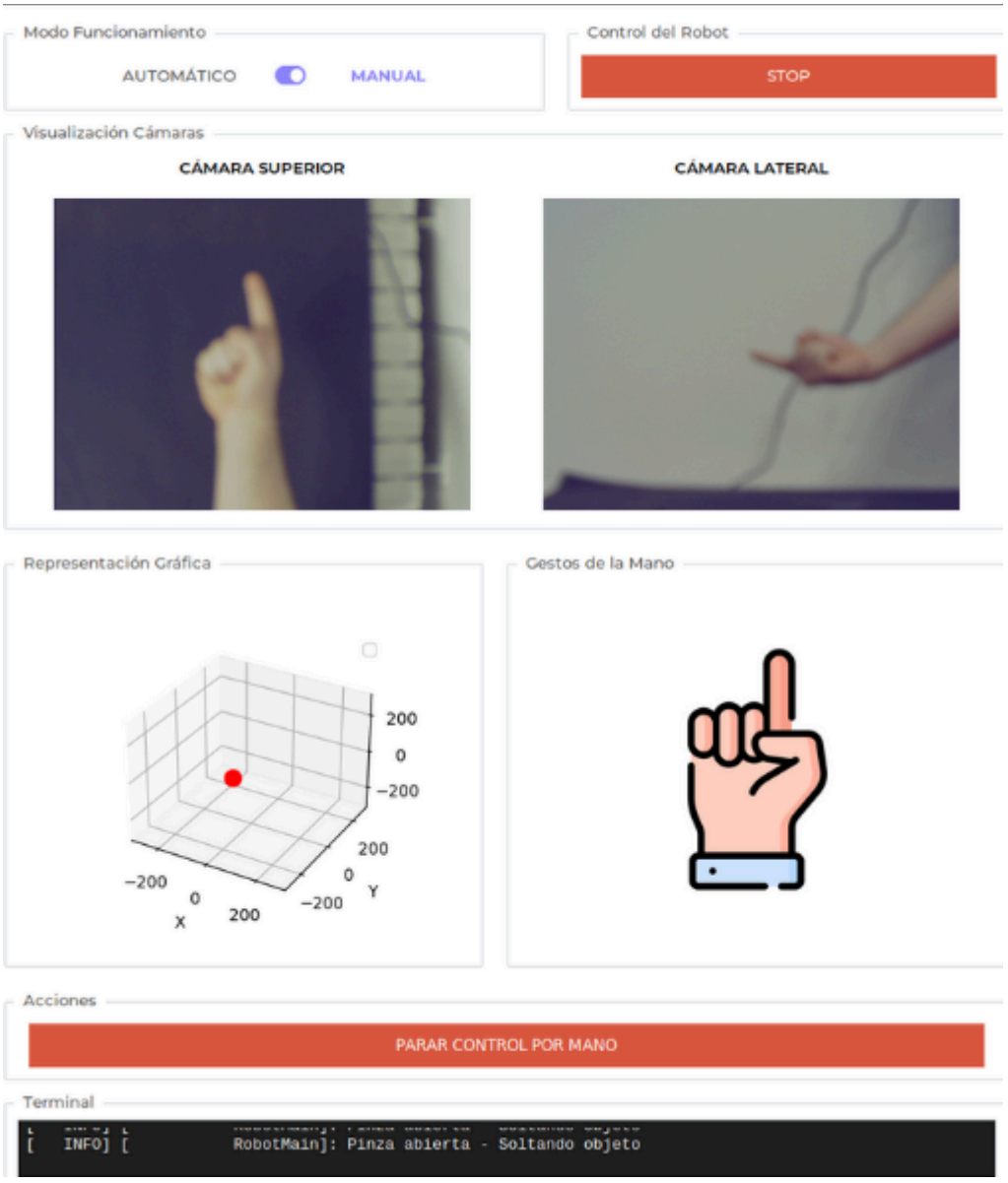


Imagen 6. Modo de funcionamiento manual en Tkinter.

Conclusiones y Trabajo Futuro

El presente trabajo ha alcanzado el objetivo de desarrollar un sistema robusto para la reconstrucción de figuras empleando cubos de colores, operable tanto en modo automático como manual, funcionando de forma continuada con tolerancia a fallos.

Para alcanzar este objetivo, se integraron de forma efectiva tres áreas fundamentales: Aprendizaje por Refuerzo, Visión por Computador y Robótica, lo que permitió generar una solución integral que interpreta el entorno, planifica trayectorias y ejecuta movimientos precisos en tiempo real.

La arquitectura distribuida adoptada se fundamentó en nodos especializados que gestionan tareas críticas de forma paralela, optimizando el uso de recursos y reduciendo la latencia.

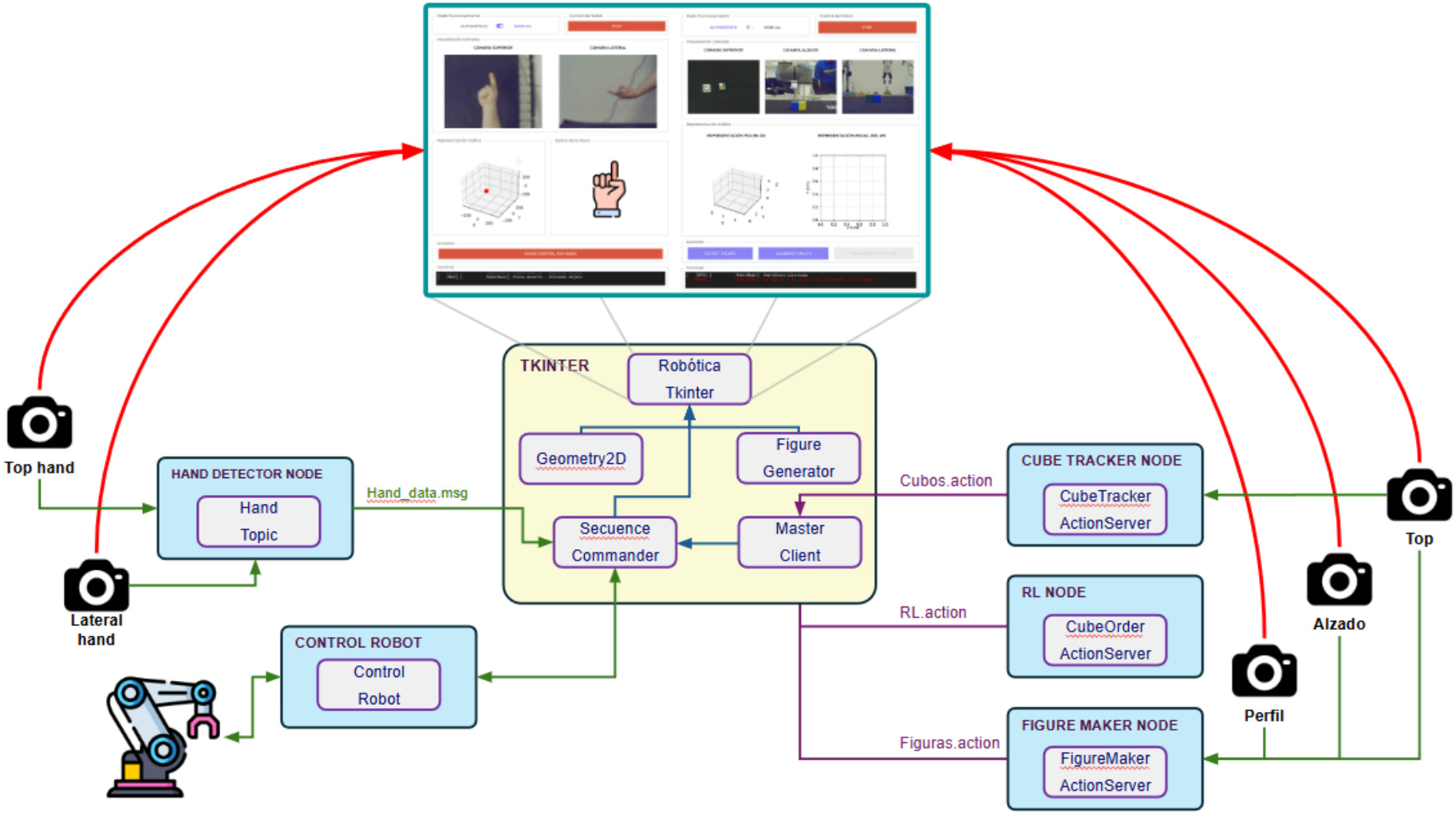
En particular:

- El Módulo de Visión: Se encarga de procesar imágenes capturadas por múltiples cámaras, extrayendo información geométrica y de color para identificar y ubicar con precisión los cubos.
- El Módulo de Planificación y Control: Hace uso de técnicas de aprendizaje por refuerzo para generar y ajustar trayectorias en tiempo real, de modo que el brazo robótico pueda reconstruir las figuras de forma precisa, tanto en el modo automático (donde el sistema toma decisiones en función de la información sensorial) como en el modo manual (en el que se permiten ajustes por parte del operador).
- La Distribución y Sincronización de Tareas: La implementación distribuida entre diferentes nodos no sólo mitiga la alta carga computacional derivada del procesamiento intensivo de imágenes y el control en tiempo real, sino que además habilita el monitoreo y control remoto del sistema, lo que resulta fundamental en entornos operativos exigentes.

Se identificaron además oportunidades para futuros perfeccionamientos:

- Calibración de Ejes XYZ: La mejora en la calibración tanto en modos automáticos como manuales es esencial para alcanzar un posicionamiento aún más preciso del robot durante la reconstrucción de figuras.
- Optimización de la Sincronización y Comunicación: Ajustes en la interacción entre nodos podrían reducir aún más la latencia, aumentando la robustez del sistema en situaciones dinámicas.
- Mejoras de Precisión: El uso de varios Arucos para la generación de una triangulación lineal entre la conversión de pixeles a unidades métricas a lo largo de las imágenes capturada puede reducir enormemente el error producido al convertir la localización de los cubos en referencia a la imagen en poses respecto al robot.

Anexos


TKINTER
Robótica Tkinter
Geometry2D
Figure Generator
Secuence Commander
Master Client
Top hand
Lateral hand
HAND DETECTOR NODE
Hand Topic
Hand_data.msg
CONTROL ROBOT
Control Robot
Cubos.action
CUBE TRACKER NODE
CubeTracker ActionServer
RL.action
RL NODE
CubeOrder ActionServer
Figuras.action
FIGURE MAKER NODE
FigureMaker ActionServer
Top
Alzado
Perfil