Guía de Implementación: Delivery de Hamburguesas con ROS 2 Jazzy

Generado por Asistente de IA

November 6, 2025

Contents

1. Visión General

Este documento describe cómo implementar, con **ROS 2 Jazzy**, una celda de entrega de hamburguesas donde un robot manipulador **Kinova Gen3** toma el producto desde una estación de armado y lo deposita en plataformas móviles diferenciales que funcionarán como repartidores terrestres. El mismo ecosistema integra visión artificial basada en AprilTags, nodos de coordinación de pedidos y control distribuido mediante micro-ROS para los robots del piso. El objetivo es contar con una referencia completa que cubra arquitectura de red, nodos requeridos y el uso de **tf2** para mantener coherencia espacial entre el brazo y los móviles.

2. Arquitectura Física y de Red

2.1. Componentes Principales

Componente	Rol	Conectividad	IP/Notas
Router WiFi	Núcleo de red; expone SSID `ros2` / pass `ros12345`	WAN + LAN	`192.168.1.1`
PC Maestro	Corre ROS 2 Jazzy completo, orquestador, micro-ros-agent, MoveIt 2 y Nav2	WiFi	`192.168.1.100`
Robot Kinova Gen3	Manipulación pick & place; ejecuta nodos de bajo nivel y MoveIt servo	Ethernet	`192.168.1.10`
Cámara aérea + PC visión	Detecta AprilTags y estima pose de robots y bandejas	WiFi	DHCP
Robots diferenciales (x3)	Plataformas delivery con ESP32 + micro-ROS	WiFi	DHCP (reservas)
Estación de pedidos	UI que envía órdenes (API REST → ROS Bridge)	WiFi	DHCP

Todos los nodos ROS se mantienen dentro de la misma subred (`192.168.1.0/24`) para permitir descubrimiento DDS sin complicaciones.

2.2. Flujo Físico del Delivery

1. La cocina coloca hamburguesas en bandejas etiquetadas en la estación Kinova.
2. El Kinova toma la bandeja usando una pinza, define el `delivery_slot_i` disponible dentro de `staging_area` y solo deposita la bandeja cuando el robot diferencial se alinea con ese frame.
3. El robot diferencial navega hasta la zona de despacho guiado por Nav2 y datos de visión.
4. Al completar la entrega, el robot regresa a la estación para recibir otro pedido.

2.3. Comunicación ROS 2 Jazzy

- **DDS (FastDDS)** maneja topics de alto ancho de banda (visión, trayectorias, estados del robot).
- **micro-ROS (UDP)** conecta los ESP32 de los diferenciales vía `micro-ros-agent`.
- **ROS Bridge** publica pedidos desde la UI web al topic `/orders/new`.

3. Stack de Software y Nodos

3.1. Nodos en el PC Maestro

Nodo	Paquete	Propósito
`order_manager`	`burger_delivery_msgs`	Convierte órdenes REST → ROS, prioriza pedidos y asigna robots.
`tray_allocator`	`burger_delivery_logic`	Decide qué robot recibe cada bandeja según disponibilidad.

Nodo	Paquete	Propósito
micro_ros_agent	micro_ros_agent	Termina la conexión UDP (ros2 run micro_ros_agent micro_ros_agent udp4 --port 8888).
tf2_ros::static_transform_publisher	tf2_ros	Define frames fijos entre map, staging_area y belt_frame.
nav2_bt_navigator + stack Nav2	nav2_bringup	Genera planes y controla los robots diferenciales a nivel macro.
move_group + kinova_gen3_control	moveit_servo	Resuelve trayectorias del brazo y los pases Kinova → bandeja.

3.2. Nodos en el Robot Kinova

Nodo	Función
kinova_driver_node	Interfaz hardware → ROS 2 (topics /joint_states, /tool_pose).
kinova_task_server	Action server personalizado que recibe PlaceOnRobot y ejecuta pick & place.
tf2 broadcasters	Publican kinova_base_link, kinova_tool_frame, burger_grip_frame.

3.3. Nodos de Visión

- vision_tag_pose (Python + apriltag_ros): calcula pose 2D/3D de bandejas y robots → topics /aruco/pose2d, /aruco/pose3d.
- tray_state_publisher: combina la detección con el estado de pedidos y emite burger_delivery_msgs/TrayState.
- tf2 broadcaster apriltag_map → robot_X/base_link usando los tags colocados en cada móvil.

3.4. Nodos en Robots Diferenciales (ESP32 + micro-ROS)

Nodo	Ubicación	Propósito
motor_controller	Firmware ESP32	Recibe objetivos /move_to_pose/goal, calcula PWM y odometría.
localization_bridge	Firmware ESP32	Fusiona IMU + /aruco/pose2d para publicar /robot_X/odom.
tray_sensor	ESP32	Notifica si la bandeja sigue a bordo (/robot_X/tray_status).

Estos nodos se conectan a micro_ros_agent mediante set_microros_wifi_transports(ssid, password, agent_ip, agent_port);.

4. Flujo de Datos y Topics Clave

Topic	Tipo	Productor → Consumidor
/orders/new	burger_delivery_msgs/Order	UI → order_manager
/tray_assignment	burger_delivery_msgs/Assignment	tray_allocator → kinova_task_server + Nav2
/aruco/pose2d	geometry_msgs/Pose2D	vision_tag_pose → ESP32 (robot_X/localization_bridge)
/move_to_pose/goal	geometry_msgs/PoseStamped	Nav2 → ESP32 motor_controller
/tf / /tf_static	TF frames	Kinova, visión y robots → toda la red
/robot_X/odom	nav_msgs/Odometry	ESP32 → Nav2, RViz

Topic	Tipo	Productor → Consumidor
/kinova/place_result	burger_delivery_msgs/PlaceResult	kinova_task_server → tray_allocator

5. Uso de tf2

tf2 es el pegamento espacial entre la manipulación y la movilidad. El frame `map` se encuentra en el piso (plano de navegación) y es el origen común desde el cual se derivan los frames del Kinova y de cada robot diferencial:

```
map
    staging_area
        delivery_slot_1
    kinova_base_link
        kinova_tool_frame
            burger_grip_frame
    robot_A/base_link
        robot_A/tray_frame
    robot_B/base_link
         robot_B/tray_frame
```

5.1. Fuentes de Transformaciones

- **Visión**: `vision_tag_pose` publica `apriltag_map -> robot_X/base_link` usando los AprilTags colocados en cada robot y en la estación de bandejas.
- **Kinova**: `kinova_driver_node` actualiza `kinova_base_link -> kinova_tool_frame` con cinemática directa. Un `static_transform_publisher` fija la posición del pedestal respecto al mapa (`map -> kinova_base_link`).
- **Robots diferenciales**: cada ESP32 envía su odometría como `robot_X/odom -> robot_X/base_link`. Un nodo en el PC Maestro (`robot_state_publisher` o `tf2_ros::TransformBroadcaster`) enlaza `map -> robot_X/odom`.

5.2. Ejemplo de transformaciones estáticas

```
ros2 run tf2_ros static_transform_publisher \
  1.20 0.30 0.00 0 0 0 \
  map kinova_base_link

ros2 run tf2_ros static_transform_publisher \
  0.80 0.00 0.00 0 0 0 \
  map staging_area

ros2 run tf2_ros static_transform_publisher \
  0.00 0.00 0.15 0 0 0 \
  robot_X/base_link robot_X/tray_frame
```

5.3. Por qué tf2 es crítico

1. MoveIt necesita conocer `map -> burger_grip_frame` para planificar trayectorias libres de colisiones desde la estación hasta la bandeja del robot.
2. Nav2 y RViz dependen de `map -> robot_X/base_link` para ubicar correctamente las metas `/move_to_pose/goal`.
3. Las detecciones de visión (que llegan en frame `apriltag_map`) se transforman vía tf2 hacia el frame del Kinova o del robot objetivo antes de ejecutar acciones.

5.4. Frame `staging_area` y URDF del árbol de transformaciones

El frame `staging_area` se fija en la estructura de la mesa donde se ubican las bandejas, pero su posición se expresa respecto al origen `map`, que está en el piso. De esta forma:

- El Kinova y los robots comparten un plano de referencia común (`map`) que coincide con el suelo donde se desplazan los diferenciales.
- Los robots reciben objetivos expresados en frames derivados del mapa (`delivery_slot_i`) para alinearse exactamente donde el Kinova depositará la hamburguesa en su bandeja.

- El Kinova selecciona un `delivery_slot_i` libre y publica esa selección (por ejemplo en `/tray_assignment`) para que Nav2 envíe al robot a dicho frame antes del pick & place.

Un URDF mínimo que codifica este árbol de transformaciones puede lucir así (también lo encontrarás como archivo independiente en `burger_delivery_frames.urdf` para cargarlo directamente con `robot_state_publisher`):

```
<?xml version="1.0"?>
<robot name="burger_delivery_frames">
  <link name="map"/>
  <link name="staging_area"/>
  <link name="delivery_slot_1"/>
  <link name="kinova_base_link"/>
  <link name="kinova_tool_frame"/>
  <link name="burger_grip_frame"/>
  <link name="robot_a_base_link"/>
  <link name="robot_a_tray_frame"/>
  <link name="robot_b_base_link"/>
  <link name="robot_b_tray_frame"/>

  <joint name="map_to_staging" type="fixed">
    <parent link="map"/>
    <child link="staging_area"/>
    <origin xyz="0.80 0.00 0.00" rpy="0 0 0"/>
  </joint>

  <joint name="map_to_kinova_base" type="fixed">
    <parent link="map"/>
    <child link="kinova_base_link"/>
    <origin xyz="1.20 0.30 0.00" rpy="0 0 0"/>
  </joint>

  <joint name="map_to_robot_a_base" type="fixed">
    <parent link="map"/>
    <child link="robot_a_base_link"/>
    <origin xyz="-0.40 0.60 0.00" rpy="0 0 0"/>
  </joint>

  <joint name="map_to_robot_b_base" type="fixed">
    <parent link="map"/>
    <child link="robot_b_base_link"/>
    <origin xyz="-0.40 -0.60 0.00" rpy="0 0 0"/>
  </joint>

  <joint name="staging_to_delivery_slot_1" type="fixed">
    <parent link="staging_area"/>
    <child link="delivery_slot_1"/>
    <origin xyz="0.00 0.40 0.90" rpy="0 0 0"/>
  </joint>

  <joint name="kinova_base_to_tool" type="floating">
    <parent link="kinova_base_link"/>
    <child link="kinova_tool_frame"/>
  </joint>

  <joint name="tool_to_grip" type="fixed">
    <parent link="kinova_tool_frame"/>
    <child link="burger_grip_frame"/>
    <origin xyz="0 0 0.18" rpy="0 0 0"/>
  </joint>

  <joint name="robot_a_base_to_tray" type="fixed">
```

```
    <parent link="robot_a_base_link"/>
    <child link="robot_a_tray_frame"/>
    <origin xyz="0.00 0.00 0.15" rpy="0 0 0"/>
  </joint>

  <joint name="robot_b_base_to_tray" type="fixed">
    <parent link="robot_b_base_link"/>
    <child link="robot_b_tray_frame"/>
    <origin xyz="0.00 0.00 0.15" rpy="0 0 0"/>
  </joint>
</robot>
```

Este URDF puede cargarse con `robot_state_publisher` para validar el árbol de `tf` en RViz y garantizar que `map -> kinova_base_link -> burger_grip_frame`, los `delivery_slot_i` y los frames de cada robot diferencial se alinean con la disposición física del laboratorio. Kinova usa los slots como objetivos de colocación y los robots se alinean con ellos antes de recibir la bandeja.

6. Secuencia Operativa

1. **Entrada de pedido**: la UI publica un `Order` (ID, destino, prioridad).
2. **Asignación y slot**: `order_manager` consulta disponibilidad de robots (`/robot_X/tray_status`), elige candidato y coordina con `kinova_task_server` qué `delivery_slot_i` (frame hijo de `staging_area`) usará el Kinova para el traspaso. Este frame se comparte vía topic (`/tray_assignment`) para que Nav2 lo use como meta intermedia.
3. **Pick & place**: el robot diferencial se posiciona en el slot indicado (`map -> delivery_slot_i`); `kinova_task_server` valida vía tf2 la coincidencia entre `robot_X/tray_frame` y `delivery_slot_i`, luego MoveIt genera la trayectoria y el Kinova deposita la bandeja.
4. **Despacho**: Nav2 envía un `PoseStamped` a `/move_to_pose/goal` del robot asignado; el firmware micro-ROS sigue el objetivo usando control diferencial.
5. **Monitoreo**: la cámara aérea actualiza `/aruco/pose2d`; el robot fusiona datos y publica `/robot_X/odom`. Nav2 reajusta planes si es necesario.
6. **Retorno**: una vez entregada la hamburguesa, el robot cambia su estado a `AVAILABLE` y solicita una nueva asignación.

7. Consideraciones de Implementación

- **QoS**: para `/aruco/pose2d` se recomienda `Reliability = Best Effort` y `History = Keep Last (5)` para minimizar latencia hacia micro-ROS.
- **Seguridad alimentaria**: incorporar sensores en el `tray_frame` para validar que la bandeja quedó fija antes de autorizar el movimiento.
- **Simulación**: usar `ros2 launch burger_delivery bringup_sim.launch.py` que levanta Gazebo con un modelo del Kinova y robots TurtleBot3 modificados; tf2 permite replicar el mismo árbol de frames.
- **Observabilidad**: RQT y RViz se conectan a `ros2` para visualizar `/tf`, `/orders/new`, y el estado de cada robot.

8. Próximos Pasos

1. Modelar los `burger_delivery_msgs` (Order, Assignment, PlaceResult) y generar los paquetes de interfaces.
2. Configurar `micro-ROS` para los robots diferenciales reutilizando las plantillas de `ros.md`, asegurando credenciales SSID `ros2` / password `ros12345`.
3. Implementar pruebas de integración en ROS 2 Jazzy: primero en simulación (Gazebo + MoveIt), luego en hardware real validando la fidelidad de tf2.