REVISTA ELECTRO, Vol. 45 pp. 196-201, Oct 2023, Chihuahua, Chih. México
https://itchihuahua.mx/revista_electro
ISSN 1405-2172

# INTEGRACIÓN DEL ROBOT UR3 Y TURTLEBOT 4 MEDIANTE EL SISTEMA OPERATIVO ROS 2

*Ramírez Valenzuela Julio César, Vázquez Cuevas Ignacio Javier, Hernández Barajas Diana Belinda, Fabian Fabian Joana Inés*
Tecnológico Nacional de México / I.T. de Nogales,
Ingeniería Eléctrica y Electrónica
Av. Tecnológico 911, Nogales, Sonora, México
+52(631) 311 1881
julio.rv@nogales.tecnm.mx

**RESUMEN.**
**El propósito de este trabajo, es la integración de los robots UR3 y Turtlebot 4 siguiendo la ideología de la Industria 4.0. Para ello se hizo uso del mega sistema operativo ROS 2. Además, se hizo uso de la cámara Robotiq instalada previamente en el UR3. Para lograr los objetivos de integración, se hizo uso de las herramientas del ROS 2, como son: SLAM y Rviz2. Para lograr la comunicación entre los robots, se hizo uso de la tecnología de sockets. También, se usaron los lenguajes de programación Python y URScript. Se tienen algunos productos resultantes de este trabajo; a destacar, un artículo publicado en ELECTRO 2022 [16].**
**Palabras Clave: ROS 2, Industria 4.0, robot móvil, robot homomórfico, integración de robots**

**ABSTRACT.**
**The purpose of this work is the integration of UR3 and Turtlebot 4 robots from the perspective of Industry 4.0. For this, the mega operating system ROS 2 was used. In addition, the Robotiq camera previously integrated with the UR3 was used. To achieve the integration objectives, the ROS 2 tools were used, such as: SLAM and Rviz2. To achieve communication between the robots, socket technology was used. Also, the programming languages Python and URScript were used. There are some products resulting from this work; To highlight, an article published in ELECTRO 2022 [16].**
**Keywords: ROS 2, Industry 4.0, mobile robot, homomorphic robot, robot integration**

## 1. INTRODUCCIÓN

En la actualidad el paradigma en la industria es la industria 4.0 [1,2,4]. Este término fue acuñado en Alemania y ha tomado mucha relevancia. Este trabajo trata sobre la integración de dos robots desde la perspectiva de la Industria 4.0 [4]. Esto es, integrar un robot homomórfico UR3 con un robot móvil Turtlebot 4 en una celda de producción. Para ello, se propone usar el Sistema Operativo para Robots: ROS 2 (*Robot Operating System* 2) [5].

Este proyecto, es la continuación de otro proyecto llamado "Jóvenes Especialistas", convocatoria de un concurso que lanzó la secretaria de economía del estado de Sonora en el año 2019. La carrera de Ingeniería Electrónica del Tecnológico Nacional de México Campus Nogales, en su participación en este concurso, ganó un Robot UR3 fabricado por *Universal Robots*. En esta siguiente fase, se recibió financiamiento por parte de PRODEP con lo que se adquirió una cámara *Robotiq* para el robot UR3 y dos Robots móviles un Turtlebot 3 y un Turlebot 4.

En la actualidad algunas universidades [4,6,7] han desarrollado proyectos de integración de robots similares al propuesto en este enfoque. Con la diferencia de que ellos usan el sistema operativo ROS 1, que ha sido sustituido por el ROS 2. También usan el robot móvil Turtlebot 2 que actualmente ha dejado de fabricarse y en su lugar se fabrica el Turtlebot 4. En la sección 2 se estudia la integración del Turttlebot 4 con el UR3. Como primera subsección, se tiene el mapeo del laboratorio por el uso de SLAM y Rviz2, que son herramientas del ROS 2. Luego, se estudian los requerimientos necesarios para la integración de los robots. Después, se presenta el programa que controla al Turtlebot 4. Finalmente, el programa que controla al UR3. En la sección 3, se tiene el análisis de resultados, destacando los problemas que se resolvieron para lograr el éxito de este proyecto. Por último, se tienen las conclusiones finales y trabajos futuros.

## 2. INTEGRACIÓN DEL LOS ROBOTS

La intención de este diseño, es desarrollar tareas de recogido y colocado (*pick and place*) del UR3 sobre el Turlebot 4. Una vez cargado el Turtlebot 4 (ver Fig. 3), llevará las piezas a la estación de descarga del UR3. Esta es una tarea ejemplo que se podría desarrollar.

### 2.1. Mapeo del laboratorio.

Lo primero que se necesita es habilitar el Turtlebot 4 para su movilidad. El robot puede ser conducido a través del teclado de la computadora del usuario o por medio de un controlador de palanca (*joystick*). Para construir un mapa de un área, como un laboratorio, el robot usa una herramienta llamada SLAM (*Simultaneous localization and mapping*) [8,9]. Esta herramienta usa varios algoritmos como: filtro de partículas, filtro Kalman extendido, FastSLAM, intersección de covarianza y SLAM basado en gráficos. El paquete (librería) gmapping proporciona la herramienta SLAM basado en láser, como un nodo ROS 2 llamado slam gmapping. Para poder ejecutar el paquete SLAM, se debe de instalar en la computadora de control una herramienta para navegación del robot con el siguiente comando:

$ sudo apt install ros-galactic-turtlebot4-navigation

Después de esto, se ejecuta la herramienta SLAM. Se recomienda ejecutar SLAM síncrono en la computadora del usuario para obtener un mapa de mayor resolución, como se muestra en el siguiente código:

$ ros2 launch turtlebot4_ navigation slam_sync.launch.py

Para visualizar el mapa, lanzar el Rviz2 [10] con el archivo de inicio view_robot. Para ello ejecutar el siguiente código:

$ ros2 launch turtlebot4_viz view_robot.launch.py

En la Fig. 1 se muestara el Rviz2 con el mapa generado del laboratorio. Rviz2 es un puerto de Rviz a ROS 2. Proporciona una interfaz gráfica para que los usuarios vean su robot, datos de sensores, mapas y más. Para producir un mapa como el de la figura, es necesario hacer un recorrido con el robot a través de todo el laboratorio. A medida que el robot se mueve se va formando el mapa que se puede visualizar en el Rviz2. Una vez que el mapa se ha generado apropiadamente, es buena idea guardarlo para su posterior uso.

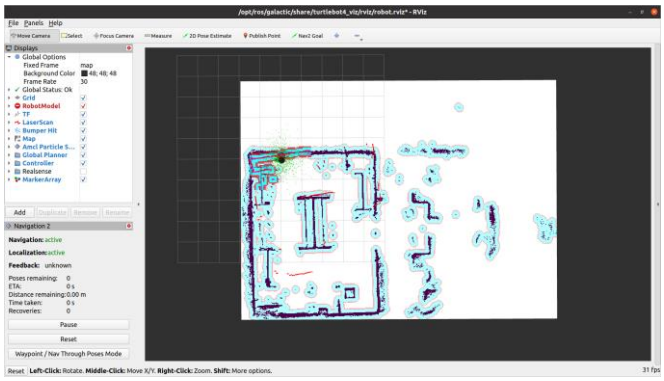

Figura 1. Mapa del laboratorio generado por Rviz2.

En la Fig. 2 se muestra un ejemplo del uso del mapa del laboratorio. La herramienta *Nav2 Goal* del Rviz2 permite establecer una pose de meta para el robot. Primero, se establece una pose inicial del robot antes de establecer una pose objetivo (*Nav2 Goal*). Luego de establecida la meta, el algoritmo de navegación calcula automáticamente una trayectoria, que evita los obstáculos al momento que el robot se va moviendo.

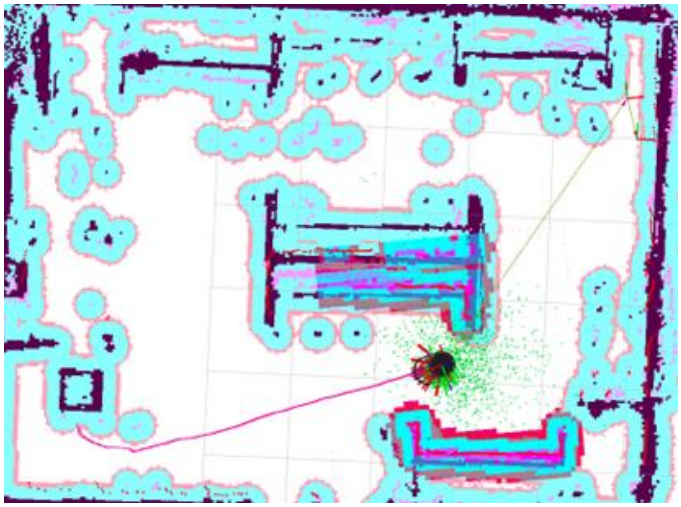

Figura 2. Ejemplo de trayectoria a recorrer.

### 2.2. Integración de los robots.

Con el propósito de mostrar el proceso de integración, se desarrollará una tarea de recogido y colocado de piezas. La tarea consistirá en el transporte de dos piezas. Una pieza será un cilindro y la otra un cubo. El Turtlebot 4, saldrá de su estación de carga con las piezas sobre la base de servicio que se fabricó para esta tarea (ver Fig. 3). El Turtlebot 4, llevará las piezas en una trayectoria específica. Se le programarán las coordenadas donde se encuentra el UR3 (estación de descarga). El Turtlebot 4, navegará en forma autónoma, evadiendo obstáculos y escogiendo las trayectorias más cortas hacia sus metas. Al llegar a la estación de descarga, el UR3 reconocerá también en forma autónoma la pieza que se le ha pedido recoger. Una vez recogida la pieza, el Turtlebot 4 regresará a su estación de carga.

Figura 3. Turtlebot 4 con las piezas a transportar.

### 2.3. Programación del Turtlebot 4.

En la Fig. 4, se muestra el diagrama de flujo del programa que se ejecuta en la computadora de control, en el sistema operativo ROS 2. El código del bloque de inicio se muestra en el Listado 1.

Listado 1. Código de bloque de inicio.

```
1  from geometry_msgs.msg import PoseStamped
2  import rclpy
3  from practicas_turtle.robot_navigator import BasicNavigator,
   NavigationResult
4  import socket
5  mi_socket=socket.socket(socket.AF_INET,
   socket.SOCK_STREAM)
6  mi_socket.bind(('192.168.88.61',6400))
7  mi_socket.listen(5)
8  rclpy.init()
9  conexion, addr =mi_socket.accept()
```

En la línea 1 y 2, se cargan en memoria las librerías básicas. La línea 4 carga un paquete llamado *socket*, que se encargará de la comunicación entre el Turtlebot 4 y el UR3 a través de la red Wi-Fi. De la línea 5 a la 6, se configura el *socket*. Con el fin de conectar los robots, en la línea 6 se proporciona la IP de la computadora de control. En la línea 9, se establece la conexión por medio del *socket*.

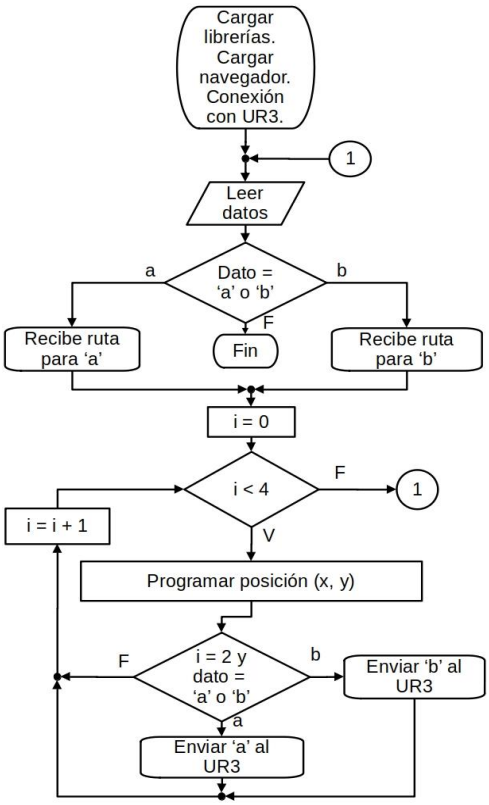


Figura 4. Diagrama de flujo del programa del Turtlebot 4.

El Listado 2, es código que se encuentra en un bucle infinito. En la línea 1, se lee la ruta que se quiere recorrer para llegar al UR3. De la línea 2 a la 9, se encuentra el primer bloque de decisión del diagrama de flujo de la Fig. 4. La ruta "a", corresponde al cubo. Esto es, el UR3 tiene que diferenciar entre un cuadrado y círculo de forma autónoma. Para el traslado del cubo, se escogieron dos paradas antes de llegar a su meta.

Listado 2. Definición de ruta del robot móvil.

```
1  data=input("ruta1=a,ruta2=b,else salir:")
2  if data =="a":
3     x=-0.7719,-5.25,-6.6456,0
4     y=-0.4279,-0.14,-3.608,0
5  elif data =="b":
6     x=-0.7719,-1.93,-6.6456,0
7     y=-0.4279,-3.17,-3.6078,0
8  else:
9     break
```

Estas paradas, se definen con las duplas (*x*, *y*). Por ejemplo, en la ruta "a", la primera parada sucede en la coordenada (-0.7719, -0.4279). Estas paradas, suponen la carga de las piezas. La última coordenada, se refiere al origen del Turtlebot 4 (*Home*). La ruta "b", corresponde al cilindro. También, para esta ruta se escogieron dos paradas, que son diferentes a las de la ruta "a", lo que supone la carga de las piezas. Si no se escoge ninguna de estas dos rutas, entonces el programa termina.

En la línea 1 del Listado 3, se muestra el inicio del ciclo *for*. Este se corresponde con el bloque donde "i = 0" en el diagrama de flujo de la Fig. 4. Las demás líneas del Listado 3, especifican la posición y orientación del robot móvil al llegar a la meta programada.

Listado 3. Programación de las posiciones a recorrer del Turtlebot 4.

```
1  for i in range(0, 4):
2     X=x[i]
3     Y=y[i]
4     goal_pose = PoseStamped()
5     goal_pose.header.frame_id = 'map'
6     goal_pose.header.stamp =
      navigator.get_clock().now().to_msg()
7     goal_pose.pose.position.x = float(X)
8     goal_pose.pose.position.y = float(Y)
9     goal_pose.pose.position.z = 0.0
10    goal_pose.pose.orientation.x = 1.0
11    goal_pose.pose.orientation.y = 0.0
12    goal_pose.pose.orientation.z = 0.0
13    goal_pose.pose.orientation.w = 1.0
14    navigator.goToPose(goal_pose)
15    while not navigator.isNavComplete():
```

El Listado 4, corresponde a la tercera iteración. Para esta instancia, el robot móvil ya ha efectuado dos paradas siendo la actual la más importante, donde se comunicará con el UR3. En esta tercera iteración, "i == 2" por lo que se envía el mensaje a a computadora de control, de que el robot móvil ha llegado a la estación de descarga del UR3; el mensaje es enviado por medio de *socket*.

Listado 4. Llegada del Turtlebot 4 a la estación del UR3.

```
1     if i==2:
2       print("El Turtlebot ha llegado")
3       if data =="a":
4         conexion.send("a".encode('utf8'))
5         jj=conexion.recv(2048)
6         print(jj.decode('utf8'))
7       else:
8         conexion.send("b".encode('utf8'))
9         jj=conexion.recv(2048)
10        print(jj.decode('utf8'))
```

En la Fig. 5 se muestra como se recoge el cilindro; para este momento el cubo ya ha sido recogido y clasificado. El cilindro se colocará en una posición diferente al cubo.

### 2.4. Programación del UR3.

La programación en los robots homomórficos UR3 es del tipo árbol, como se puede apreciar en la Fig. 6. La interfaz de programación es una pantalla táctil, pero también se puede programar a través de una computadora vía un enlace Wi-Fi. El lenguaje de programación del UR3 es el *URScript* y su interfaz gráfica de usuario se le llama *PolyScope* [11]. Se programó el robot UR3 como servidor de la comunicación *socket.*

Figura 5. Recogida del cilindro.

Inicialmente establece comunicación con el Maestro (Turtlebot 4 en Python). Después de establecida la conexión, el UR3 siempre estará monitoreando el puerto, esperando mensaje del Turtlebot 4. Si el mensaje fue una "a", busca la pieza cuadrada y la clasifica; si el mensaje fue una "b", busca la pieza circular y la clasifica. En la Fig. 7, se puede observar el diagrama de flujo de este programa.

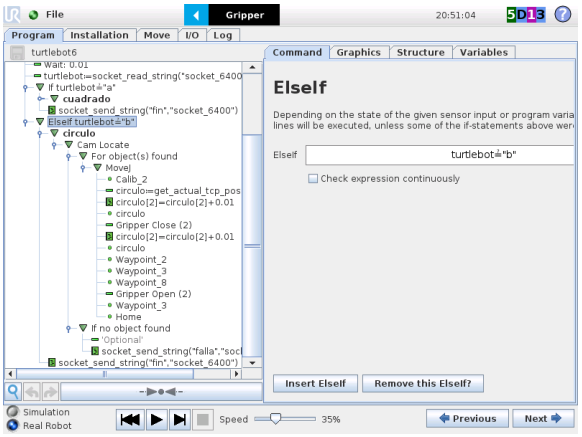


Figura 6. Programación tipo árbol del UR3.

En el Listado 5, en la línea 5, se establece comunicación con el Turtlebot 4 por medio de *sockets*. Para este propósito, se usa el comando *socket_open*(). La IP que se incluye como parámetro en este comando, se corresponde con la IP de la computadora de control del Turtlebot 4. De la línea 6 a la 11, se da inicio al programa de recogido y colocado de piezas.
Inicialmente se mueve en la posición de inicio esperando un mensaje por *socket* como se observa en la línea 6, el comando *socket_read_string()*, lee el mensaje enviado por el Turtlebot 4.

Listado 5. Inicio de programa de UR3: preparación para esperar la pieza.

```
1   Program
2     BeforeStart
3       MoveJ
4         Home
5         socket_open("192.168.88.61",6400,"
          socket_6400")
6     Robot Program
7       'Turtle'
8       MoveJ
9         Waypoint_1
10        Wait: 0.01
11        Turtlebot socket_read_string("socket_
          6400")
12        If turtlebot≟"a"
13          cuadrado
14            MoveJ
15              Waypoint_4
16            Cam Locate
```

En la línea 16, se ejecuta el comando *Cam Locate*, que inicia la localización de objetos por medio de la cámara *Robotiq*. La cámara *Robotiq*, es una cámara inteligente que tiene muchas prestaciones: como el aprendizaje de patrones de imagen.

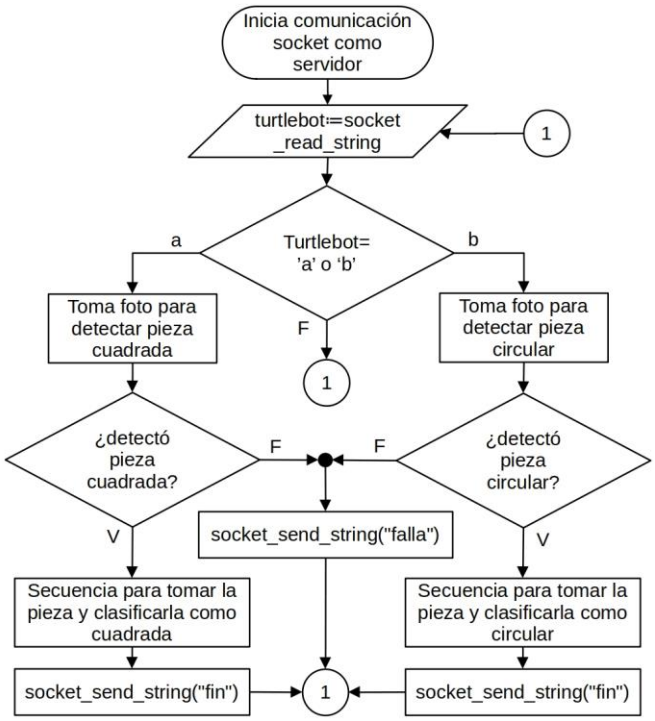


Figura 7. Diagrama de flujo del programa del UR3.

En el Listado 6, en la línea 1, se inicia el ciclo *for* para la búsqueda del cuadrado. En la línea 2, se ejecuta un movimiento hacia la posición de calibración Calib_ 2 de la línea 3.

Listado 6. Reconocimiento del cubo.

```
1        For object(s) found
2          MoveJ
3            Calib_2
4            pieza:=get_actual_tcp_pose()
5            pieza[2]=pieza[2]-0.01
6            pieza
7            Gripper Close (2)
8            pieza[2]=pieza[2]+0.01
9            pieza
10           Waypoint_5
11           Waypoint_6
12           Waypoint_7
13           Gripper Open (2)
14           Waypoint_6
15       If no object found
16         'Optional'
17         socket_send_string("falla","socket
           _6400")
18     socket_send_string("fin","socket_6400")
```

En la Fig. 8, se muestra la posición Calib_ 2, con la que se hizo el procedimiento de entrenamiento del reconocimiento y aprendizaje de patrones de imágenes. Para esta calibración se usa un programa asistente (*wizard*), que toma alrededor de treinta fotografías de cada pieza en diferentes poses en la Fig. 9, se puede observar el entrenamiento del reconocimiento de las piezas. Después de esta calibración, la cámara es capaz de reconocer en forma autónoma cada pieza.

Figura 8. Posición de calibración para el aprendizaje de piezas Calib_ 2.

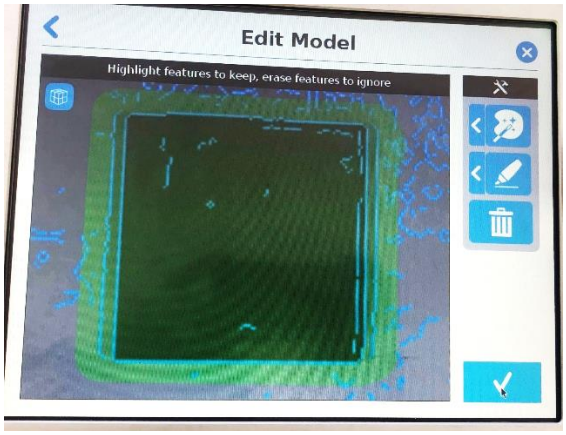


Figura 9. Entrenamiento de reconocimiento de piezas.

En este caso, detecta al cuadrado y carga la posición donde se localizó la pieza en el comando *get_actual_tcp_pose*(), que se encuentra en la línea 4. De la línea 4 a la 18 se muestra el procedimiento de recogida y colocado del cubo. Para la recogida y colocado del cilindro, el programa es similar al del cubo, sólo se agrega la inferencia *ElseIf turtlebot*≟"b" en el árbol del programa del UR3, con un procedimiento similar al del cubo.

## 3. ANÁLISIS DE RESULTADOS

Se ha podido desarrollar prácticas de recogido y colocado de piezas del UR3 sobre el Turtlebot 4. Un ejemplo de los recorridos realizados por el Turtlebot 4 es la trayectoria que se muestra en la Fig. 2. Se ha podido desarrollar y publicar un artículo que involucara la cámara Robotiq y al UR3 [16]. Se ha integrado con éxito el Turtlebot 4 con el UR3 por el uso de *sockets*. Aunque la curva de aprendizaje para la configuración del Turtlebot 3 y Turtlebot 4 es algo prolongada, los procedimientos de configuración son confiables. Unos de los

problemas encontrados en el Turtlebot 4, es la precisión en los recorridos. Esto es, que cuando se le indica viajar a una meta en coordenadas (X, Y), la precisión puede varias alrededor de +/- 3 cm. La solución consistió, en ajustar la orientación del Turtlebot 4 cuando este llega a la estación de descarga del UR3. También ayudó en la solución de este problema, el hecho de que la cámara *Robotiq* del UR3 puede localizar autónomamente la pieza; siempre y cuando el Turtlebot 4 se encuentre en el rango de visión del UR3.

Se pudo probar empíricamente (esto es, por funcionalidad), que el ROS 2 es una tecnología de software más conveniente que el ROS 1 [12,15]. Tiene mayor seguridad, entre otras muchas prestaciones [13]. Entre las diferencias más significativas entre ROS 1 y ROS 2 están: el lenguaje, los sistemas operativos, la creación de nodos, la capa de transporte y la compilación, que se describen en la Tabla 1.

Tabla 1. Diferencias significativas entre ROS 1 y ROS 2.

| *Diferencias* | **ROS 1** | **ROS 2** |
|---|---|---|
| *Lenguaje* | Trabaja con Python 2.7 y C++11. | Se puede crear código en Python 3.5 y C++14 y 17. |
| *Sistemas operativos* | Se puede instalar en Linux y también en el MacOS. | Se puede instalar en Linux, MacOS y también en Windows 10. |
| *Creación de nodos* | "Master", crea los nodos que definen al sistema de una manera centralizada. | Prescinde del Master, es un sistema distribuido. |
| *Capa de transporte* | Hace uso de TCPROS, el cual se basa en TCP/IP para el envío de mensajes y la comunicación con servicios. | Se utiliza DDS (*Data Distribution Service*). El DDS es una herramienta necesaria para aplicaciones en tiempo real, es una estructura de arquitectura publicador/subscriptor. |
| *Compilación* | Usa un comando llamado *Catkin* | Usa la herramienta llamada *Colcon*. |

Las diferencias entre el Turtlebot 2 y el Turtlebot 4 son: que el Turtlebot 2 está equipado con una base para robot Kobuki, una netbook de doble núcleo, un sensor Orbbec Astra Pro y un giroscopio. En cambio, el Turtlebot 4 no está equipado con una laptop, sino con una computadora de una sola placa, como es el Raspberry Pi 4. El software que se ejecuta en esta pequeña pero poderosa computadora, es el ROS 2. El modelo que se usó para integrar en este laboratorio es el Turtlebot 4 estándar (Ver Fig. 3) que viene equipado con una base móvil iRobot® Create3, una cámara estéreo espacial OAK-D, un sensor Laser 2D LiDAR y algunos sensores más. Los Turtlebot son robots móviles asequibles (menos de tres mil *us dlls*.) [14], comparados con los robots móviles industriales (que suelen ser mucho más caros).

## 4. CONCLUSIONES

Se logró el objetivo principal de este proyecto, que es la integración de los robots UR3 y Turtlebot 4 utilizando sockets y cámara robotiq siguiendo la ideología de la Industria 4.0 similar al presentado por Tosello *et. al.* [4]. El diseño de este lproyecto implicó el aprendizaje del mega sistema operativo ROS 2, con algunas de sus herramientas como el SLAM y el Rviz2. También, el uso de *sockets* para la comunicación e integración con los robots. Como trabajos futuros se propone el diseño de robots móviles o articulados aplicando el ROS 2 para su control.

### 4.1. Referencias.

[1] Oztemel, Ercan, and Samet Gursev. "Literature review of Industry 4.0 and related technologies." *Journal of intelligent manufacturing* 31 (2020): 127-182.
[2] Javaid, Mohd, et al. "Substantial capabilities of robotics in enhancing industry 4.0 implementation." Cognitive Robotics 1 (2021): 58-75.
[3] Vaisi, Bahareh. "A review of optimization models and applications in robotic manufacturing systems: Industry 4.0 and beyond." Decision Analytics Journal (2022): 100031.
[4] Tosello, Elisa, Nicola Castaman, and Emanuele Menegatti. "Using robotics to train students for Industry 4.0." IFAC-PapersOnLine 52.9 (2019): 153-158.
[5] Macenski, Steven, et al. "Robot Operating System 2: Design, architecture, and uses in the wild." Science Robotics 7.66 (2022): eabm6074.
[6] Thale, Sumegh Pramod, et al. "ROS based SLAM implementation for Autonomous navigation using Turtlebot." ITM Web of conferences. Vol. 32. EDP Sciences, 2020.
[7] Hou, Yew Cheong, et al. "Development of collision avoidance system for multiple autonomous mobile robots." International Journal of Advanced Robotic Systems 17.4 (2020): 1729881420923967.
[8] Macenski, Steve, and Ivona Jambrecic. "SLAM Toolbox: SLAM for the dynamic world." Journal of Open Source Software 6.61 (2021): 2783N.R. Vela, Titulo, Ciudad, Editorial, año, páginas
[9] Merzlyakov, Alexey, and Steve Macenski. "A comparison of modern general-purpose visual SLAM approaches." 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2021.
[10] Rviz2, Guía de usuario. Enlace: http://docs.ros.org/en/jade/api/rviz/html/user_guide/index.html
[11] *PolyScope*, Universal Robots User Manual, UR3/CB3. Universal Robot, 2021. Enlace: https://www.universal-robots.com/download/?query=
[12] Wong, Ching-Chang, et al. "Generic Development of Bin Pick-and-Place System Based on Robot Operating System." IEEE Access 10 (2022): 65257-65270.
[13] Goerke, Niklas, David Timmermann, and Ingmar Baumgart. "Who controls your robot? An evaluation of ROS security mechanisms." 2021 7th International conference on automation, robotics and applications (ICARA). IEEE, 2021.
[14] Niu, Haoyu, Tiebiao Zhao, and YangQuan Chen. "Intelligent bugs mapping and wiping (iBMW): An affordable robot-driven robot for farmers." 2019 IEEE International Conference on Mechatronics and Automation (ICMA). IEEE, 2019.
[15] Fischer, Tobias, et al. "A RoboStack Tutorial: Using the Robot Operating System Alongside the Conda and Jupyter Data Science Ecosystems." IEEE Robotics & Automation Magazine 29.2 (2021): 65-74.
[16] Ramírez, Julio, et al. "Método para determinar el campo FOV de una cámara Robotiq y una cámara Cognex en condiciones de diseño no específicas." Revista ELECTRO, Vol. 44 pp. 213-218, Oct 2022, Chihuahua, Chih. México. http:// electro.itchihuahua.edu.mx/revista. ISSN 1405-2172