

# ETAPA 02

## La plataforma ROS y el robot Takeshi

Instructor: Marco Antonio Negrete Villanueva

Programa Espacial Universitario, UNAM

Concurso Iberoamericano de Robótica Espacial 2022

<https://github.com/mnegretev/CIRE2022>

# Objetivos:

**Objetivo General:** Que los participantes se familiaricen con las herramientas de software que se usarán durante el concurso.

## Objetivos Específicos:

- ▶ Revisar la plataforma ROS y el software de control de versiones Git
- ▶ Dar un panorama general del hardware y software con que cuenta el robot Toyota HSR (Takeshi)
- ▶ Aprender a interactuar con los sensores y actuadores del robot mediante ROS:
  - ▶ Base móvil omnidireccional
  - ▶ Sensor Lidar
  - ▶ Cámaras RGB y RGB-D
  - ▶ Manipulador

# Contenido

Motivación

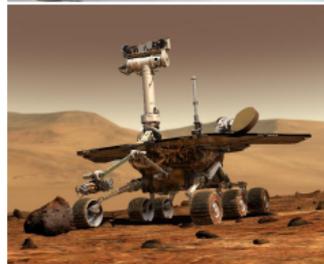
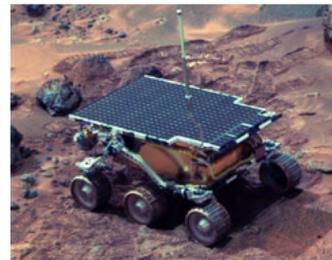
Conceptos Básicos

ROS

Ejercicios

El uso de robots en la exploración del espacio tiene varias décadas[4]:

- ▶ En 1970, la URSS envió el Lunokhod 1 (teleoperado)
- ▶ En 1997 la NASA envió el Mars Pathfinder Sojourner (evasión autónoma de obstáculos)
- ▶ En 2004 aterrizaron en Marte el Spirit y Opportunity (autónomos)
- ▶ En 2012 aterrizó el Curiosity (autónomo)



Además de rovers, también hay desarrollo en robots de servicio para aplicaciones aeroespaciales[2]:

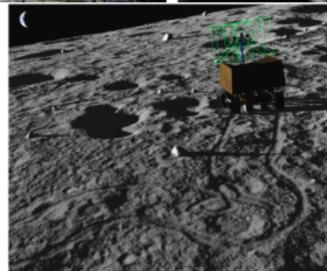


- ▶ Robonaut R2, desarrollado por DARPA/NASA, en la ISS en 2011<sup>1</sup>
- ▶ Rollin' Justin, desarrollado por el DLR<sup>2</sup>

<sup>1</sup><https://www.nasa.gov/robonaut2>

<sup>2</sup><https://www.dlr.de/rm/en/desktopdefault.aspx/tabid-11427/>

- ▶ ROS Space: La plataforma ROS con estándar aeroespacial para software<sup>1</sup>
- ▶ Robonaut 2 y 5 utilizan ROS y tienen sus versiones en Gazebo<sup>2</sup>
- ▶ Astrobees, desarrollado por la NASA utiliza ROS<sup>3</sup>
- ▶ El proyecto VIPER (NASA) utiliza Gazebo para simulaciones<sup>4</sup>



<sup>1</sup><https://vimeo.com/649649866/37198994b5>

<sup>2</sup><https://robots.ros.org/robonaut2/>

<sup>3</sup><https://github.com/nasa/astrobees>

<sup>4</sup><https://www.openrobotics.org/blog/2022/2/2/rosinspace>

El *Human Support Robot* (HSR) de Toyota es una plataforma para investigación en robots de servicio.



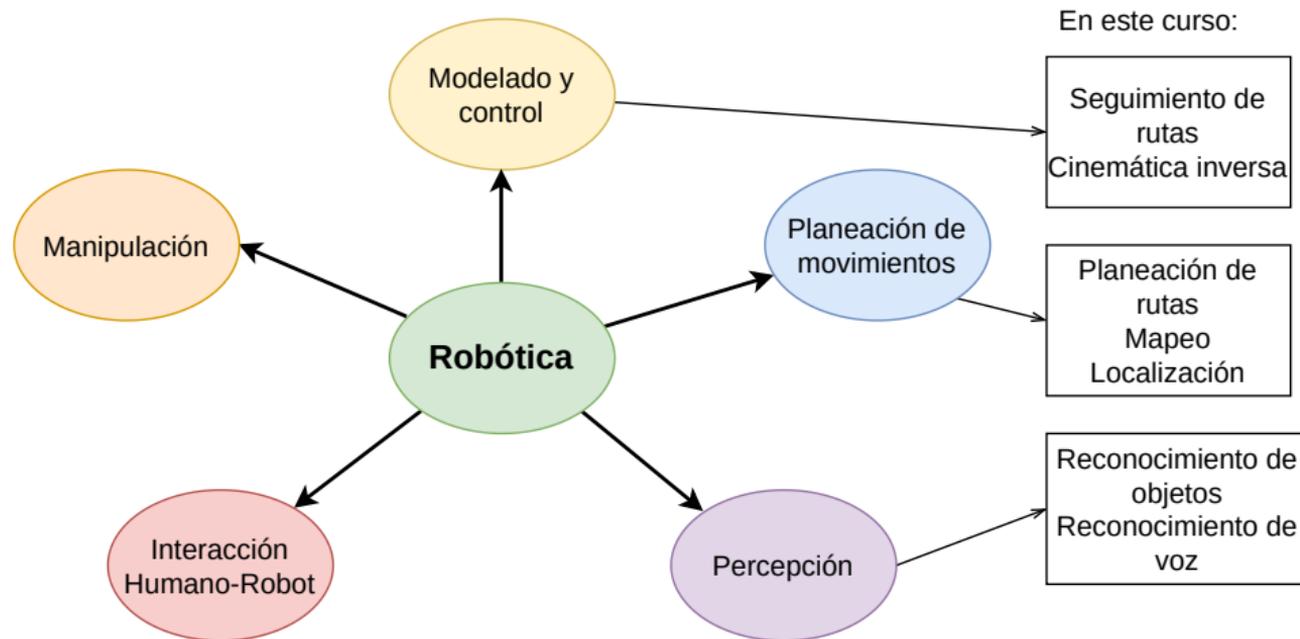
Hardware:

- ▶ Base móvil omnidireccional
- ▶ Manipulador de 5 DoF
- ▶ Cámaras RGB y RGB-D en la cabeza
- ▶ Cámara RGB en el efector final
- ▶ Lidar en la base móvil
- ▶ Bocinas y micrófono

Software: Arquitectura basada en ROS con las siguientes funcionalidades:

- ▶ Control de la base móvil
- ▶ Control del manipulador
- ▶ Acceso a los datos de los sensores mediante tópicos

- ▶ La palabra *robot* tiene su origen en la obra *Rossum's Universal Robots* del escritor checo *Karel Čapek*, publicada en 1921, y su significado es “trabajo duro”.
- ▶ Latombe (1991) define un robot como un dispositivo mecánico versátil equipado con sensores y actuadores bajo el control de un sistema de cómputo [5].
- ▶ Arkin (1998) propone que un robot inteligente es una máquina capaz de extraer información de su ambiente y usar el conocimiento acerca de su mundo para moverse de manera segura y significativa, con un propósito específico [1].
- ▶ Robótica es la ciencia que estudia la conexión inteligente entre la percepción y la acción.



### Robot

#### Sensores:

Son transductores que obtienen información del ambiente útil para la toma de decisiones.

**Propioceptivos:** sienten el estado interno del robot.

**Exteroceptivos:** sienten el ambiente externo.

**Activos:** emiten energía para realizar la medición.

**Pasivos:** no emiten energía.

Ejemplos de Sensores:

- Cámaras (RGB, RGB-D)
- Micrófonos
- Lidar
- Encoders
- Sensores de batería

#### Procesadores:

Es el hardware que se utiliza para procesar información. Reciben información de los sensores y envían comandos a los actuadores.

Ejemplos de procesadores:

- CPUs
- GPUs
- FPGA
- Microcontrolador
- DSP

#### Actuadores:

Son dispositivos que realizan alguna modificación al ambiente. Se pueden clasificar según su principio de actuación:

- Eléctricos
- H
- Neumáticos

Ejemplos de actuadores

- Motores (CD, Brushless)
- Bocinas
- Pistones
- Manipuladores

- ▶ **Configuración:** es la descripción de la posición en el espacio de todos los puntos del robot. Se denota con  $q$ .
- ▶ **Espacio de configuraciones:** es el conjunto  $Q$  de todas las posibles configuraciones.
- ▶ **Grados de libertad:** número mínimo de variables independientes para describir una configuración. En este curso, la base móvil del robot tiene 3 GdL, la cabeza tiene 2 GDL y cada brazo tiene 7 GDL más 1 GdL para el gripper. En total, el robot tiene 21 GdL.

### Propiedades del robot:

- ▶ **Holonómico:** el robot puede moverse instantáneamente en cualquier dirección del espacio de configuraciones. Comúnmente se logra mediante ruedas de tipo *Mecanum* u *Omnidireccionales*.
- ▶ **No holonómico:** existen restricciones de movimiento en velocidad pero no en posición. Son restricciones que solo se pueden expresar en términos de la velocidad pero no pueden integrarse para obtener una restricción en términos de posición. Ejemplo: un coche sólo puede moverse en la dirección que apuntan las llantas delanteras, sin embargo, a través de maniobras puede alcanzar cualquier posición y orientación. El robot de este curso es no holonómico.

### Propiedades de los algoritmos:

- ▶ **Complejidad:** cuánta memoria y cuánto tiempo se requiere para ejecutar un algoritmo, en función del número de datos de entrada (número de grados libertad, número de lecturas de un sensor, entre otros).
- ▶ **Optimalidad:** un algoritmo es óptimo cuándo encuentra una solución que minimiza una función de costo.
- ▶ **Complejitud:** un algoritmo es completo cuando garantiza encontrar la solución siempre que ésta exista. Si la solución no existe, indica falla en tiempo finito.
  - ▶ Complejitud de resolución: la solución existe cuando se tiene una discretización.
  - ▶ Complejitud probabilística: la probabilidad de encontrar la solución tiende a 1.0 cuando el tiempo tiende a infinito.

Una explicación más detallada se puede encontrar en el Cap. 3 de [3].

Las tareas que puede llevar a cabo un robot se pueden clasificar en tres grandes conjuntos conocidos como primitivas de la robótica: sensor, planear y actuar.

- ▶ **Sensar:** extracción de información del ambiente interno o externo del robot.
- ▶ **Planear:** generación de subtareas y toma decisiones a partir de la información de los sensores y/o de alguna Representación interna del ambiente.
- ▶ **Actuar:** modificación del ambiente con alguno de los dispositivos del robot.

**Paradigma jerárquico.** Las tres primitivas se realizan en forma secuencial.



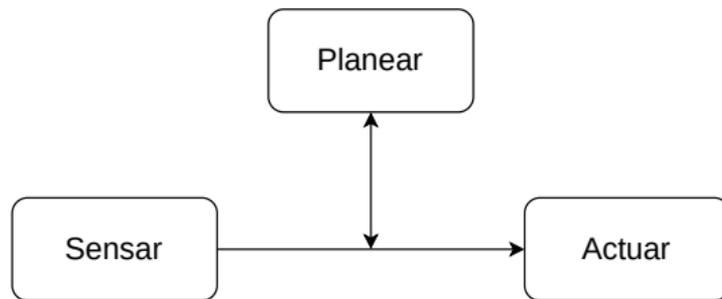
- ▶ Fuerte dependencia de una representación interna del ambiente
- ▶ Tiempo de respuesta lento comparado con el paradigma reactivo
- ▶ Alto costo computacional
- ▶ Se pueden resolver tareas con alto nivel cognitivo
- ▶ Alta capacidad de predicción

**Paradigma reactivo.** El sensado y la actuación se conectan directamente sin que haya de por medio una planeación.



- ▶ No requiere una representación interna del ambiente
- ▶ Tiempo de respuesta rápido comparado con el paradigma jerárquico
- ▶ Bajo costo computacional
- ▶ En general, no se pueden resolver tareas con alto nivel cognitivo
- ▶ Baja capacidad de predicción

**Paradigma híbrido.** Tiene como objetivo utilizar las ventajas de ambos paradigmas, es decir, emplear comportamientos reactivos para que el robot responda rápidamente ante cambios en el ambiente sin perder la alta capacidad cognitiva y de predicción que brinda el paradigma jerárquico





**ROS (Robot Operating System)** es un *middleware* de código abierto para el desarrollo de robots móviles.

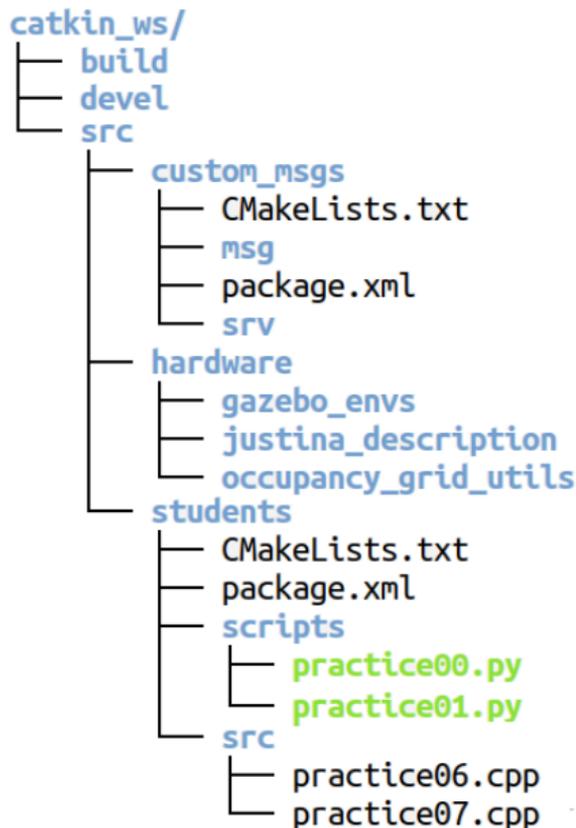
- ▶ Implementa funcionalidades comúnmente usadas en el desarrollo de robots como el paso de mensajes entre procesos y la administración de paquetes.
- ▶ Muchos drivers y algoritmos ya están implementados.
- ▶ Es una plataforma distribuida de procesos (llamados *nodos*).
- ▶ Facilita el reuso de código.
- ▶ Independiente del lenguaje (Python y C++ son los más usados).
- ▶ Facilita el escalamiento para proyectos de gran escala.

ROS se puede entender en dos grandes niveles conceptuales:

- ▶ **Sistema de archivos:** Recursos de ROS en disco
- ▶ **Grafo de procesos:** Una red *peer-to-peer* de procesos (llamados nodos) en tiempo de ejecución.

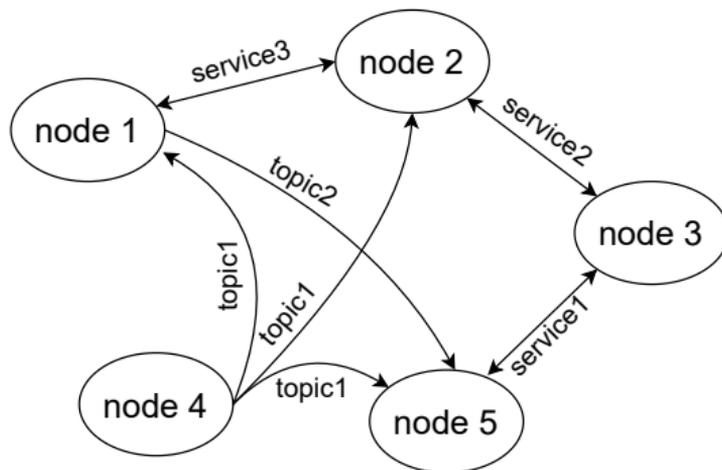
Recursos en disco:

- ▶ **Workspace:** carpeta que contiene los paquete desarrollados
- ▶ **Paquetes:** Principal unidad de organización del software en ROS (concepto heredado de Linux)
- ▶ **Manifiesto:** (`package.xml`) provee metadatos sobre el paquete (dependencias, banderas de compilación, información del desarrollador)
- ▶ **Mensajes (`msg`):** Archivos que definen la estructura de un *mensaje* en ROS.
- ▶ **Servicios (`srv`):** Archivos que definen las estructuras de la petición (*request*) y respuesta (*response*) de un servicio.



El grafo de procesos es una red *peer-to-peer* de programas (nodos) que intercambian información entre sí. Los principales componentes del este grafo son:

- ▶ master
- ▶ servidor de parámetros
- ▶ nodos
- ▶ mensajes
- ▶ servicios



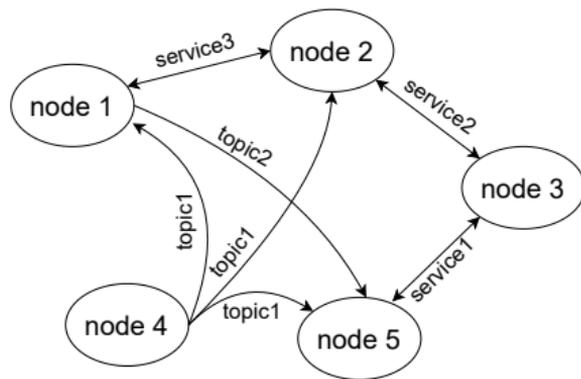
Los nodos (procesos) en ROS intercambian información a través de dos grandes patrones:

### ► Tópicos

- Son un patrón 1 :  $n$  de tipo *publicador/suscriptor*
- Son no bloqueantes
- Utilizan estructuras de datos definidas en archivos \*.msg para el envío de información

### ► Servicios

- Son un patrón 1 : 1 de tipo *petición/respuesta*
- Son bloqueantes
- Utilizan estructuras de datos definidas en archivos \*.srv para el intercambio de información.



Para mayor información:

- Tutoriales <http://wiki.ros.org/ROS/Tutorials>
- Koubâa, A. (Ed.). (2020). Robot Operating System (ROS): The Complete Reference. Springer Nature

1. Investigar para qué sirve el mensaje LaserScan
2. Investigar para qué sirve el mensaje Twist
3. Buscar para qué sirven los comandos: `roslaunch`, `rostopic list` y `rostopic echo`
4. Abra el archivo `catkin_ws/src/exercises/scripts/exercise01.py` y agregue el siguiente código en la línea 27:

```
27 n = int((msg.angle_max - msg.angle_min)/msg.angle_increment/2)
28 obstacle_detected = msg.ranges[n] < 1.0
29
```

En el mismo archivo, en la línea 49, agregue el siguiente código:

```
49 msg_cmd_vel = Twist()
50 msg_cmd_vel.linear.x = 0 if obstacle_detected else 0.3
51 pub_cmd_vel.publish(msg_cmd_vel)
```

5. Corra el programa con los comandos:  

```
$ source ~/CIRE2022/catkin_ws/devel/setup.bash
$ roslaunch exercises exercise01.py
```
6. Describir qué hace el programa y, de ser posible, realice un diagrama de flujo o esquema similar.

-  R. Arkin.  
*Behavior-Based Robotics.*  
MIT Press, 1998.
-  R. Bogue.  
Robots for space exploration.  
*Industrial Robot: An International Journal*, 2012.
-  H. M. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, S. Thrun,  
and R. C. Arkin.  
*Principles of robot motion: theory, algorithms, and implementation.*  
MIT press, 2005.
-  A. Ellery.  
Survey of past rover missions.  
In *Planetary Rovers*, pages 59–69. Springer, 2016.
-  J. Latombe.  
*Robot Motion Planning.*  
Kluwer Academic, 1991.

# Gracias

## Contacto

Dr. Marco Negrete  
Profesor Asociado C  
Departamento de Procesamiento de Señales  
Facultad de Ingeniería, UNAM.

[mnegretev.info](mailto:mnegretev.info)  
[marco.negrete@ingenieria.unam.edu](mailto:marco.negrete@ingenieria.unam.edu)