ETAPA 02
La plataforma ROS y el robot Takeshi

Instructor: Marco Antonio Negrete Villanueva

Programa Espacial Universitario, UNAM

Consurso 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

Los robots en el espacio

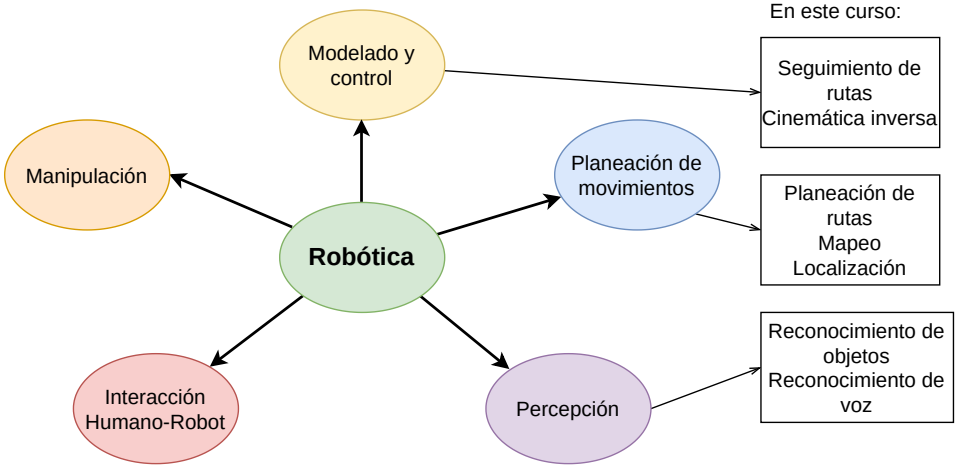
ROS en proyectos aeroespaciales

El robot Toyota HSR

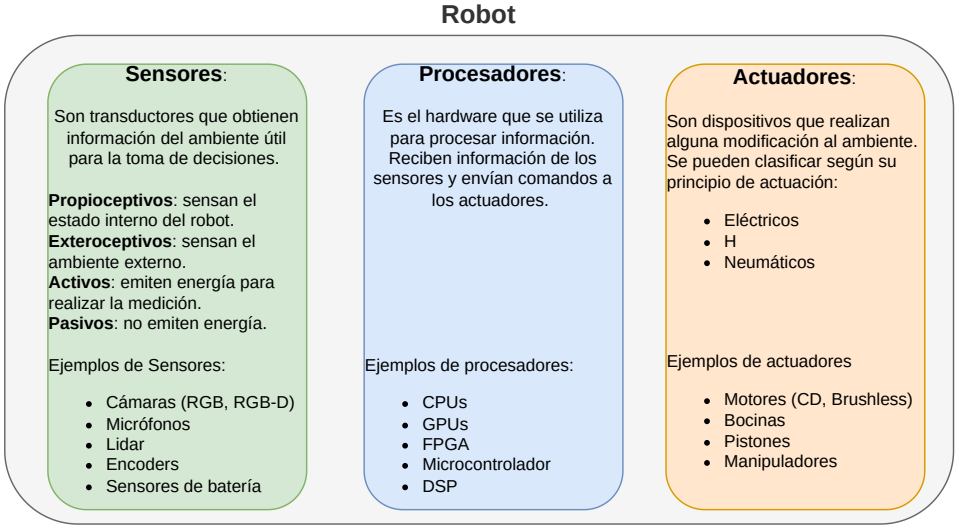
Contexto y definiciones

- La palabra *robot* tiene su origen en la obra *Rossum's Universal Robots* del escritor checo *Karel Čapeck*, 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 [**?**].
- 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 [**?**].
- Robótica es la ciencia que estudia la conexión inteligente entre la percepción y la acción.

Áreas de la Robótica



Componentes de un robot



- **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 espacion de configuraciones. Comunmente 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.
- **Completitud:** un algoritmo es completo cuando garantiza encontrar la solución siempre que ésta exista. Si la solución no exite, indica falla en tiempo finito.
  - Completitud de resolución: la solución existe cuando se tiene una discretización.
  - Completitud probabilística: la probabilidad de encontrar la solución tiende a 1.0 cuando el tiempo tiende a infinito.

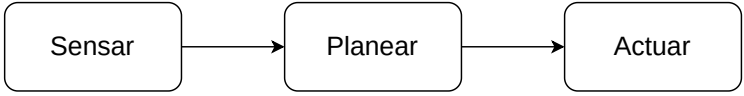
Una explicación más detallada se puede encontrar en el Cap. 3 de [**?**].

Primitivas de la robótica

Las tareas que puede llevar a cabo un robot se pueden clasificar en tres grandes conjuntos conocidos como primitivas de la robótica: sensar, 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 intenra del ambiente.
- **Actuar:** modificación del ambiente con alguno de los dispositivos del robot.

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

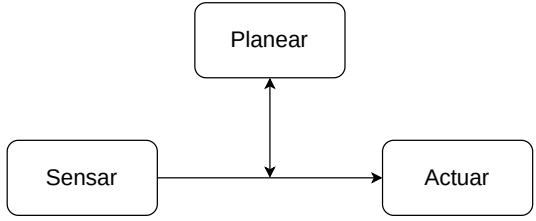


- Fuerte dependencia de una representación interna del ambiente
- Tiempo de respuesta lento comparado con el paradigma reactivo
- Alto costo computacional
- Se pueden resolver tareas con alto nivel cognitivo
- Alta capacidad de predicción

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.

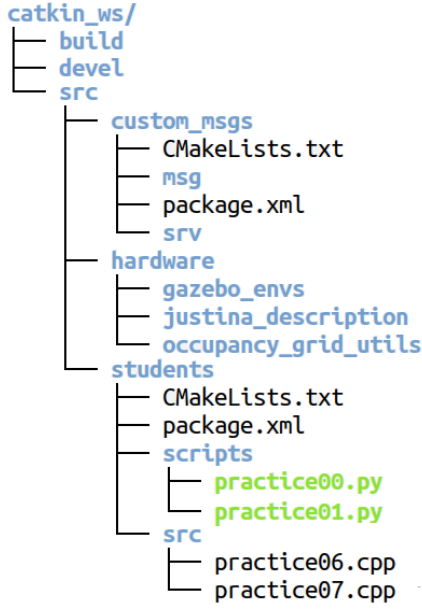
ROS se puede entender en dos grandes niveles conceptuales:

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

Sistema de archivos

Recursos en disco:

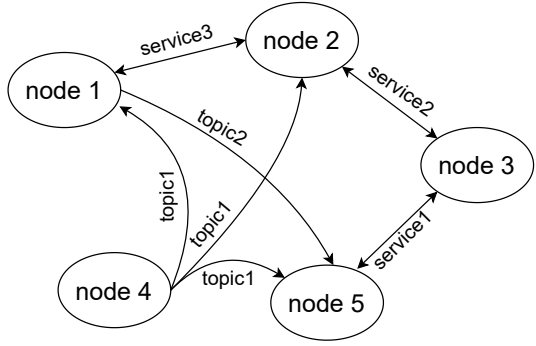
- **Workspace:** carpeta que contiene los paquete desarrollados
- **Paquetes:** Principal unidad de organización del software en ROS (concepto heredado de Linux)
- **Manifiesto:** (`package.xml`) provee metadatos sobre el paquete (dependencias, banderas de compilación, información del desarrollador)
- **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.



Grafo de procesos

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:

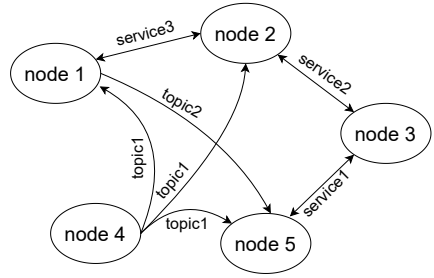
- master
- servidor de parámetros
- nodos
- mensajes
- servicios



Tópicos y 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: `rosrun`, `rostopic list` y `rostopic echo`
4. Abra el archivo `catkin_ws/src/students/scripts/assignment01.py` y agregue el siguiente código en la línea 25:

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

   En el mismo archivo, en la línea 45, agregue el siguiente código:

```
45 msg_cmd_vel = Twist()
46 msg_cmd_vel.linear.x = 0 if obstacle_detected else 0.3
47 pub_cmd_vel.publish(msg_cmd_vel)
48
```

5. Describir qué hace el programa de la tarea 1, de ser posible, agregue un diagrama de flujo o esquema similar.

Gracias

Contacto

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

mnegretev.info
marco.negrete@ingenieria.unam.edu