ROS 2 - Foxy

Introducción

La robótica es un campo en desarrollo, durante los próximos años se espera un crecimiento significativo en el número de robots empleados en diferentes áreas de la economía mundial, no sólo en la industria, sino también en áreas más cercanas como el comercio o los medios de transporte.

Uno de los principales retos a los que se enfrenta la robótica es la estandarización de los sistemas de programación de los robots. Con la finalidad de obtener un estándar surge el ecosistema ROS (Robot Operating System).

ROS es un sistema de código abierto que nació en 2007 en el laboratorio de investigación Willow Garage. Su filosofía es la de compartir código, por lo que actualmente el ecosistema ROS está compuesto por decenas de miles de desarrolladores de todo el mundo, desde aficionados hasta programadores de grandes industrias. En este artículo podéis ver algunas de las companías que están apostando por ROS para programar sus robots.

Las características principales de ROS son:

- Está compuesto por un conjunto de librerías y utilidades que permiten implementar aplicaciones para diferentes plataformas robóticas
- El sistema operativo nos proporciona una capa que nos permite abstraernos del hardware
- Los procesos a realizar se programan en módulos pequeños que denomina nodos (programación modular)
- Los nodos utilizan la técnica del intercambio de mensajes para comunicarse entre ellos, para ello hacen uso del protocolo de comunicaciones TCP/IP
- Es necesario un nodo maestro que dirige la comunicación entre el resto de los nodos (roscore)
- Las funcionalidades básicas necesarias para operar con un robot ya están implementadas (navegación, mapeado de un entorno, percepción, visión, manipulación...) (Programación reutilizable)
- Soporta varios lenguajes de programación (C++, Python, Java, MATLAB...)

A partir del año 2017 aparece una nueva versión de ROS, denominada ROS2, que introduce cambios significativos en el modo de trabajar de este sistema operativo.

El principal cambio radica en que, mientras que ROS utilizaba un sistema de intercambio de mensajes basado en el protocolo de comunicaciones TCP/IP, ROS2 hace uso de un sistema distribuido (DDS- Data Distribution Service).

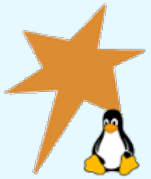
Se trata de un framework de arquitectura publicador/subscriptor que aporta mucha flexibilidad y eficiencia al permitir la comunicación directa entre nodos, sin necesidad de que exista un nodo maestro. Además, DDS ofrece gran capacidad de configuración gracias al QoS (Quality of Service). Se utiliza en aplicaciones que necesitan respuesta en tiempo real, tales como sistemas aeroespaciales, sistemas de vuelo etc.

La siguiente tabla muestra las principales diferencias entre ROS y ROS2.

	ROS	ROS2
Cuando aparece	2007	2017
Lenguajes de programación	python 2.7 C ++ 03 C ++ 11	python 3.5 C++ 11 C ++ 14 C ++ 17
Plataformas	Ubuntu OS X	Ubuntu Xenial Windows 10 OS X
OS	Linux Mac	Linux Windows MAC RTOs
Trabaja con múltiples robots en un mismo proceso	NO	SI
Protocolo utilizado para el intercambio de mensajes	TCP/IP	DDS
Compilador utilizado	catkin	colcon

ROS 2- Conceptos y Arquitectura

- Instalación ROS2 en PC
- Acceso a Polilab Elegid **EPSG Linux**
- **Practica Polilab**



EPSG Linux

- Documentación ROS2
- Estructura de paquetes ROS2
- Nodos
- Introducción al interface de comunicaciones
- Topics
  - Publicando en un Topic
  - Subscribiéndonos a un Topic
- Mensajes de usuario
- Parámetros
- Servicios
- Acciones
- Herramientas de depuración:
  - Mensajes de depuración
  - Visualizador RVIZ
  - Visualizador transformadas

Enlace a ROS2 Cheats Sheet

Enlace a web de referencia API ROS2-python (rclpy)

API ROS2 Mensajes

Teoría transformación de coordenadas

En ROS se denominará *frame* a los sistemas de coordenadas.

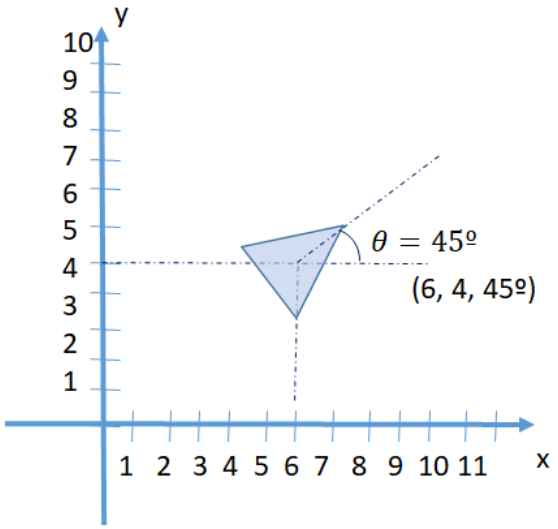
La representación de un objeto en un plano viene dada por una variable llamada Pose que está compuesta por la posición (x, y, z) más la orientación. La orientación se presenta en modo de cuaterniones.

Representación de coordenadas en 2D

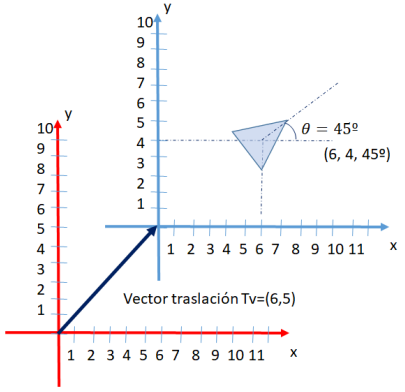
Representación de coordenadas en 2D

Para representar la posición y orientación de un objeto en un plano se utilizan las coordenadas x, y más el ángulo θ que forma el eje del objeto con el plano x.

En el caso del ejemplo de la figura siguiente la posición y orientación del objeto sería x=6, y=4, θ=45º.



Si tuviésemos dos sistemas de coordenas (el W1 mostrado con los ejes azules en la figura y el W2 mostrado con los ejes rojos), necesitamos conocer en qué posición se encuentra un sistema respecto al otro. Si ambos sistemas tienen la misma orientación y simplemente se encuentran desplazados (como es el caso de la siguiente figura), sólo necesitaremos conocer el vector traslación (Tv). En este caso el vector es Tv = (6, 5).



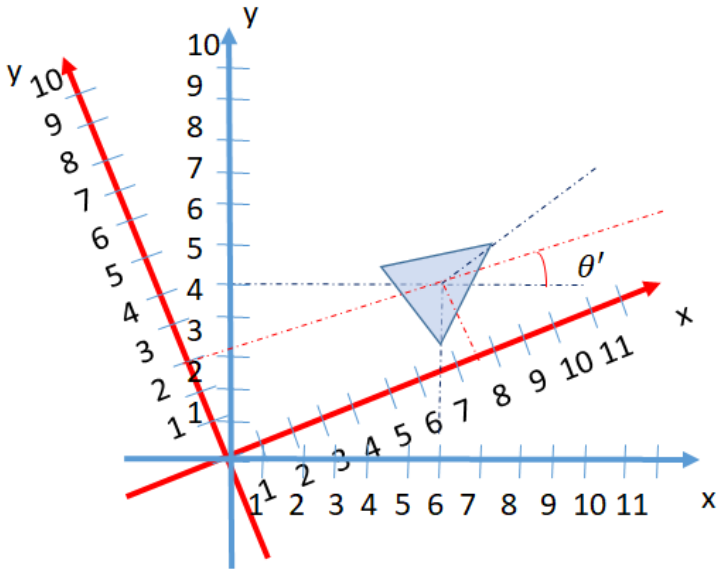
$$\begin{bmatrix} X^{W2} \\ Y^{W2} \end{bmatrix} = \begin{bmatrix} X^{W1} \\ Y^{W1} \end{bmatrix} + \begin{bmatrix} T_{Vx} \\ T_{Vy} \end{bmatrix}$$

$$\begin{bmatrix} X^{W2} \\ Y^{W2} \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix} + \begin{bmatrix} 6 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} X^{W2} \\ Y^{W2} \end{bmatrix} = \begin{bmatrix} 12 \\ 9 \end{bmatrix}$$

Por lo tanto, en el sistema de coordenadas W2 la posición del objeto es (12,9, 45º).

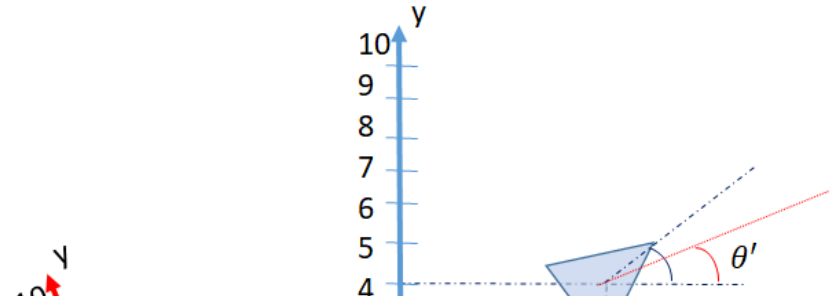
Si se da el caso de utilizar dos sistemas de coordenadas que se hallan girados uno respecto a otro, como en la siguiente figura, deberemos aplicar trigonometría para obtener la matriz de rotación R.

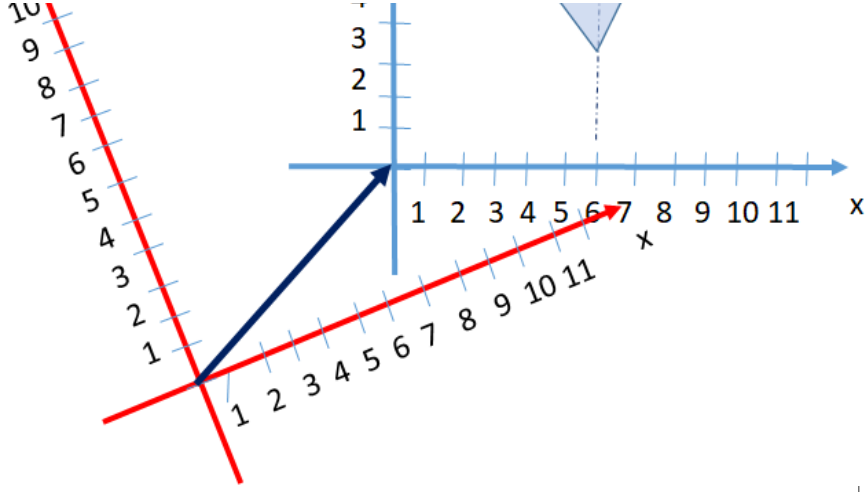


$$\begin{bmatrix} X^{W1} \\ Y^{W1} \end{bmatrix} = \begin{bmatrix} cos\,(\theta') & -sin\,(\theta') \\ sin\,(\theta') & cos\,(\theta') \end{bmatrix} \cdot \begin{bmatrix} X^{W2} \\ Y^{W2} \end{bmatrix}$$

$$R = \begin{bmatrix} cos\,(\theta') & -sin\,(\theta') \\ sin\,(\theta') & cos\,(\theta') \end{bmatrix}$$

En el caso de que entre los dos sistemas de coordenadas haya una traslación y una rotación deberemos tener en cuenta el vector traslación y la matriz de rotación.





$$\begin{bmatrix} X^{W1} \\ Y^{W1} \end{bmatrix} = \begin{bmatrix} cos\,(\theta') & -sin\,(\theta') \\ sin\,(\theta') & cos\,(\theta') \end{bmatrix} \cdot \begin{bmatrix} X^{W2} \\ Y^{W2} \end{bmatrix} - \begin{bmatrix} T_{Vx} \\ T_{Vy} \end{bmatrix}$$

Representación de coordenadas en 3D

Representación de coordenadas en 3D

Al representar coordenadas en un espacio tridimensional se complican los cálculos ya que las rotaciones se pueden producir en cualquiera de los tres ejes. Podemos utilizar diferentes métodos para representar coordenadas en 3 dimensiones. A continuación presentamos los más utilizados.

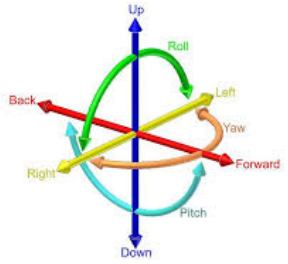
Sistema de representación de 3-ángulos Euler

Se define la rotación en cada uno de los 3 ejes (x, y, z).

Este sistema utiliza la regla de la mano derecha para definir el sentido de rotación positivo.

La nomenclatura a utilizar para indicar las rotaciones alrededor de los ejes es la siguiente:

- Se llama Roll (balanceo) a la rotación alrededor del eje x. El ángulo de rotación será α.
- Se llama Pitch (inclinación) a la rotación alrededor del eje y. El ángulo de rotación será β.
- Se llama Yaw (viraje) a la rotación alrededor del eje z. El ángulo de rotación será γ.



En este caso tendremos una matriz de rotación por cada uno de los ejes. Estas matrices serán

$$Rx = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,(\alpha) & -sin\,(\alpha) \\ 0 & sin\,(\alpha) & cos\,(\alpha) \end{bmatrix}$$

$$Ry = \begin{bmatrix} cos\,(\beta) & 0 & sin\,(\beta) \\ 0 & 1 & 0 \\ -sin\,(\beta) & 0 & cos\,(\beta) \end{bmatrix}$$

$$Rz = \begin{bmatrix} cos\,(\gamma) & -sin\,(\gamma) & 0 \\ sin\,(\gamma) & cos\,(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La matriz de rotación completa será la multiplicación de las tres matrices:

$$R = Rx \cdot Ry \cdot Rz = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

El punto en el nuevo sistema de coordenadas será:

$$\begin{bmatrix} X^{W1} \\ Y^{W1} \\ Z^{W1} \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_{Vx} \\ r_{21} & r_{22} & r_{23} & T_{Vy} \\ r_{31} & r_{32} & r_{33} & T_{Vz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X^{W2} \\ Y^{W2} \\ Z^{W2} \\ 1 \end{bmatrix}$$

Siendo T_{Vx}, T_{Vy} y T_{Vz} los tres puntos del vector de traslación.

De todo lo anterior de puede concluir que para poder realizar las transformaciones de un sistema de coordenadas a otro necesitamos conocer:

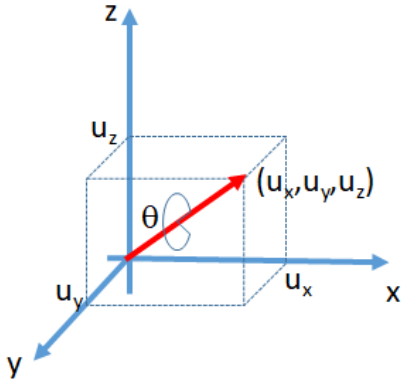
- El vector de traslación
- La matriz de rotación

Afortunadamente ROS nos permite trabajar con varios sistemas de coordenadas y nos proporciona tanto el vector de traslación y la matriz de rotación para poder realizar las transformaciones oportunas.

Eje-Rotación

También se podría utilizar otro sistema similar llamado **eje-rotación.** Se basa en la definición de un eje alrededor del cual se realiza la rotación. El eje se define con tres puntos (u_x,u_y,u_z) y tendrá su origen en el punto (0,0,0).

Al ángulo de rotación alrededor del eje lo denominaremos θ.



En este caso la matriz de rotación será:

$$R = \begin{bmatrix} cos\,(\theta) + u_x^2 \cdot (1 - cos\,(\theta)) & u_x \cdot u_y \cdot (1 - cos\,(\theta)) - u_z \cdot sin\,(\theta) & u_x \cdot u_z \cdot (1 - \\ u_y \cdot u_x \cdot (1 - cos\,(\theta)) + u_z \cdot sin\,(\theta) & cos\,(\theta) + u_y^2 \cdot (1 - cos\,(\theta)) & u_y \cdot u_z \cdot (1 - \\ u_z \cdot u_x \cdot (1 - cos\,(\theta)) - u_y \cdot sin\,(\theta) & u_z \cdot u_y \cdot (1 - cos\,(\theta)) + u_x \cdot sin\,(\theta) & cos\,(\theta) + \end{bmatrix}$$

Cuaternión

ROS representa la orientación de un objeto a través de un **cuaternión**. Se trata de un sistema de números

(parecido a los números complejos) que utiliza cuatro dimensiones.

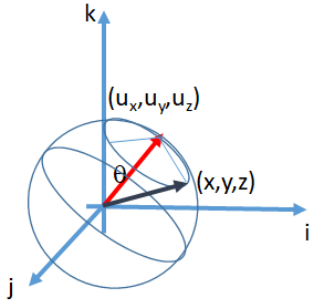
Cualquier objeto tendrá asociado cuatro números escalares: uno real y tres de ellos imaginarios que representaremos de la siguiente manera:

$q = q_0 + q_1 \cdot \hat{i} + q_2 \cdot \hat{j} + q_3 \cdot \hat{k}$, donde $\hat{i}$, $\hat{j}$ y $\hat{k}$ son los ejes imaginarios.

Desde el punto de vista de la representación de eje-rotación vista en el apartado anterior, podemos representar un cuaternión como se muestra a continuación:

$$q = cos\left(\frac{\theta}{2}\right) + \hat{i} \cdot u_x \cdot sin\left(\frac{\theta}{2}\right) + \hat{j} \cdot u_y \cdot sin\left(\frac{\theta}{2}\right) + \hat{k} \cdot u_z \cdot sin\left(\frac{\theta}{2}\right)$$

donde θ es el ángulo de rotación sobre el eje (u_x, u_y, u_z).



La matriz de rotación será:

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2 \cdot (q_1 \cdot q_2 - q_0 \cdot q_3) & 2 \cdot (q_0 \cdot q_2 + q_1 \cdot q_3) \\ 2 \cdot (q_1 \cdot q_2 + q_0 \cdot q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2 \cdot (q_2 \cdot q_3 - q_0 \cdot q_1) \\ 2 \cdot (q_1 \cdot q_3 - q_0 \cdot q_2) & 2 \cdot (q_0 \cdot q_1 + q_2 \cdot q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

Para realizar la rotación utilizando cuaterniones necesitamos conocer:

- el vector inicial a rotar (x, y, z)
- el vector eje de rotación (u_x, u_y, u_z)
- el ángulo a rotar (θ)

Conociendo todos los parámetros anteriores podemos calcular las componentes del cuaternión:

$q_0 = cos(\theta)$

$q_1 = u_x \cdot sin(\theta)$

$q_2 = u_y \cdot sin(\theta)$

$q_3 = u_z \cdot sin(\theta)$

La rotación se realizará multiplicando la matriz de rotación por el quaternión del vector a rotar.

$P_1 = R \cdot P_0$

siendo P_0 el cuaternión del objeto a rotar, R la matriz de rotación y P_1 el cuaternión final.

ROS nos ofrece la información de la posición utilizando quaterniones, la nomenclatura utilizada es la siguiente:

$w = q_0 = cos(\theta)$

$x = q_1 = u_x \cdot sin(\theta)$

$y = q_2 = u_y \cdot sin(\theta)$

$z = q_3 = u_z \cdot sin(\theta)$

ROS 2 - Navegación

Para que nuestro robot sea capaz de navegar de manera autónoma vamos a necesitar realizar las siguientes tareas:

- Escanear el entorno en el que va a trabajar nuestro robot
- Cargar el mapa escaneado
- Localizar el robot en el mapa
- Planificar la ruta a seguir para alcanzar el destino
- Controlar el seguimiento de la ruta

Todas estas acciones las haremos utilizando los paquetes

- Cartographer_ros: para escanear el mapa

- Navigation2 para controlar el servidor de mapas y todas las acciones de navegación

El paquete de navegación navigation2 implementa varios algoritmos que nos van a permitir:

- mover un robot de un punto a otro
- seguir objetos dinámicos

Para ello el paquete incluye:

- la elaboración de trayectorias dinámicas
- cómputo de la velocidad a aplicar a las ruedas del robot
- evita obstáculos
- algoritmo de recuperación en el caso de que el robot se quede atascado

El paquete posee herramientas para:

- Cargar el mapa (map_server)
- Localizar el robot en el mapa (amcl)
- Planificar una trayectoria evitando los obstáculos (nav2planner)
- Controlar el robot mientras sigue una trayectoria (nav2controller)
- Convertir la información que recibimos de los sensores en una mapa (Nav2 costmap 2D) donde se representan en diferentes colores las zonas ocupadas de las libres
- Implementar un árbol de comportamiento para decidir qué va a hacer el robot dependiendo de las circunstancias en las que se encuentre (Nav2 Behavior Tree)
- Calcular el algoritmo de recuperación en caso de fallo (Nav2 Recovery)
- Seguir un punto móvil (Nav2 Waypoint Follower)
- Administrar el ciclo de vida y el watchdog de los servidores (Nav2 Lifecycle Manager)
- Permite implementar Plugins para hacer nuestros propios algoritmos (Nav2 Core)

Para instalar el paquete de navegación debéis ejecutar el siguiente código:

```
sudo apt install ros-foxy-navigation2 ros-foxy-nav2-bringup '~ros-foxy-turtle
```

Simultaneous Localization and Mapping (SLAM) es el nombre que recibe el paquete que se encarga de crear un mapa y localizar el robot en en mismo. Se trata de la primera tarea que deberá realizar nuestro robot antes de comenzar a utilizar un entorno de trabajo nuevo.

Existen varios paquetes que podemos utilizar para realizar esta tarea, nosotros vamos a utilizar el paquete Cartographer, creado por Google para realizar tareas de localización y mapeado en 2D y 3D, utilizando diferentes plataformas. Este paquete permite configurarse para usarse con distintos sensores.

A través de los siguientes colabs vamos a aprender a mapear un entorno, guardar el mapa y volver a cargarlo cuando lo necesitemos.

SLAM en simulación

El servidor de mapas

ROS 2 - Testeo

ROS2 - Máquinas de Estado