

# Módulo 1: Introducción a ROS y Manipuladores Robóticos

Docente:  
Ing. Wilber Rojas Fernández



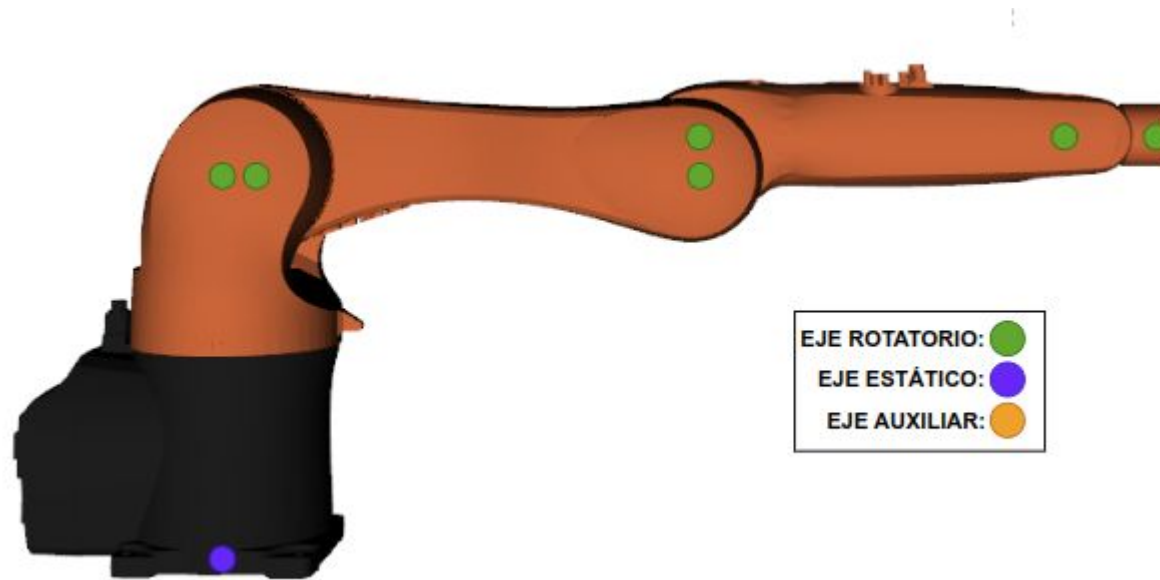
Postgrado

# 6. Programación de brazos Robóticos

## Un poco de teoria...

Para manipular brazos roboticos existen dos metodos que se pueden implementar:

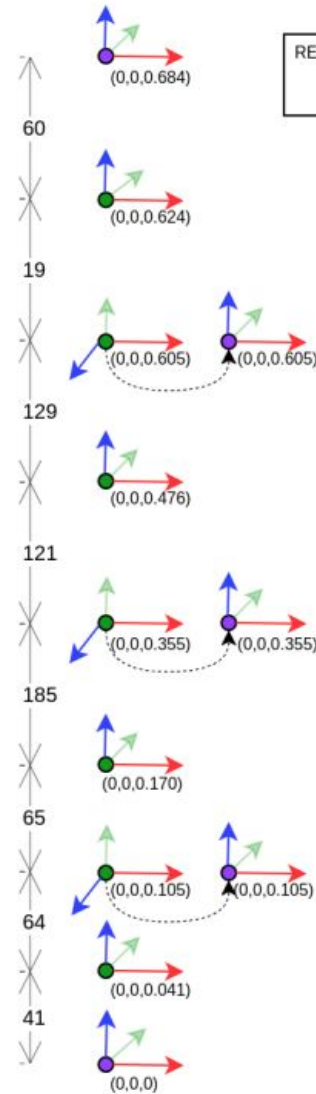
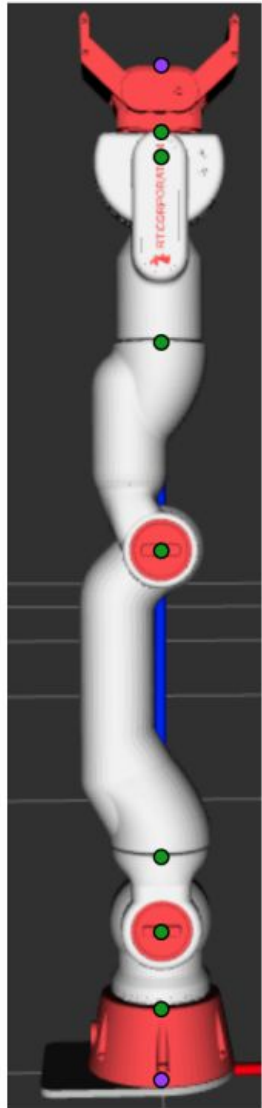
- Cinemática Directa: Saber donde esta la herramienta del robot (X, Y, Z) en base a los ángulos de los motores (J1,J2,...).
- Cinemática Inversa: Saber qué ángulos deberían tener los motores (J1, J2, ...) en base a una posicion (X, Y, Z)



## Técnicas Matemáticas

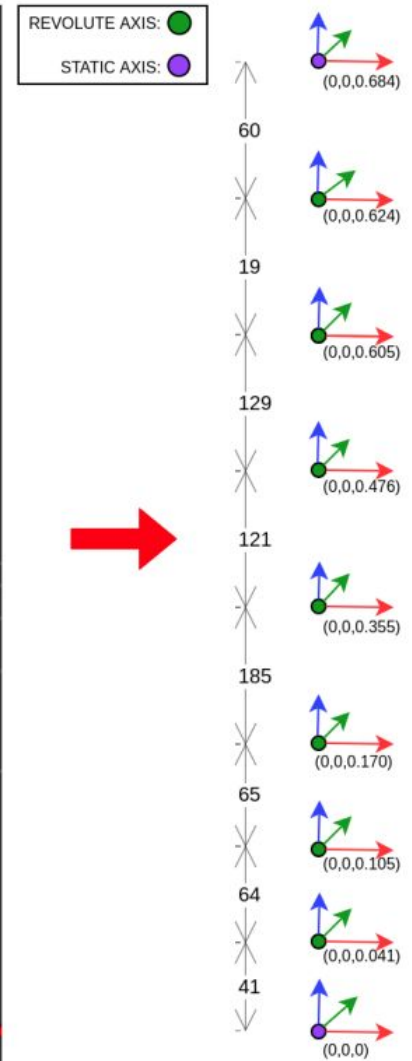
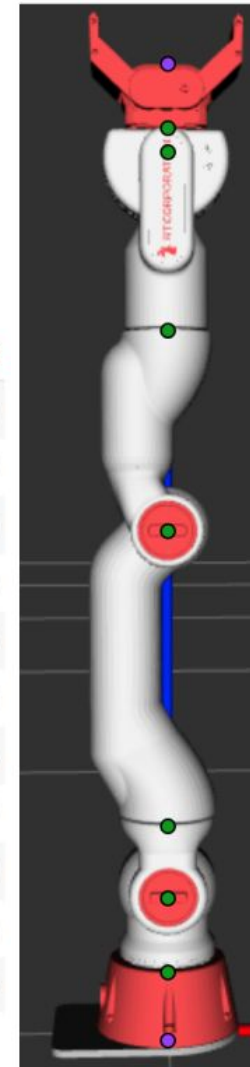
Por ejemplo, el método de Denavit-Hartenberg (DH) y la fórmula de Rodrigues son dos técnicas matemáticas utilizadas en robótica para calcular la cinemática directa.

## Método de Denavit-Hartenberg (DH)



	$\theta$	$d$	$a$	$\alpha$
T01	0	41	0	0
T12	T1	64	0	90
T23	T2	0	0	-90
T34	0	65	0	0
T45	T3	185	0	90
T56	T4	0	0	-90
T67	0	121	0	0
T78	T5	129	0	90
T89	T6	0	0	-90
T910	0	19	0	0
T1011	T7	60	0	0

## Fórmula de Rodrigues



Para Rodríguez se arman matrices de transformación:

$$T'_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,005 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T'_6 = \begin{bmatrix} \cos(\theta_6) & 0 & -\sin(\theta_6) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta_6) & 0 & \cos(\theta_6) & 0,129 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T'_1 = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0,036 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T'_7 = \begin{bmatrix} \cos(\theta_7) & -\sin(\theta_7) & 0 & 0 \\ \sin(\theta_7) & \cos(\theta_7) & 0 & 0 \\ 0 & 0 & 1 & 0,019 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T'_2 = \begin{bmatrix} \cos(\theta_2) & 0 & -\sin(\theta_2) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta_2) & 0 & \cos(\theta_2) & 0,064 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T'_8 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,060 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T'_3 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & 0 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0,065 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T'_4 = \begin{bmatrix} \cos(\theta_4) & 0 & -\sin(\theta_4) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta_4) & 0 & \cos(\theta_4) & 0,185 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Luego se multiplican todas estas matrices

$$T'_{09} = T'_0 \cdot T'_1 \cdot T'_2 \cdot T'_3 \cdot T'_4 \cdot T'_5 \cdot T'_6 \cdot T'_7 \cdot T'_8 \cdot T'_{n-1,n} \cdot \dots$$

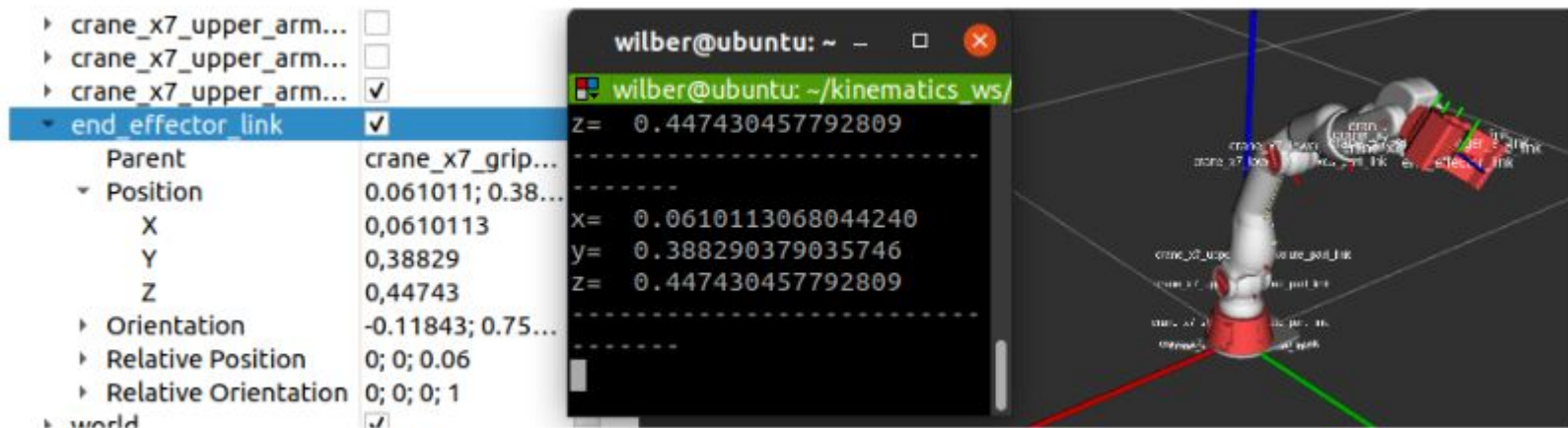
En donde, del resultante se obtienen ecuaciones para x,y,z.

$$\begin{bmatrix} T'_{09_{03}} \\ T'_{09_{13}} \\ T'_{09_{23}} \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Por ejemplo, para x:

$$\begin{aligned} x = & -0,079 * (((-sin(j1) * sin(j3) + cos(j1) * cos(j2) * cos(j3)) * cos(j4) \\ & - sin(j2) * sin(j4) * cos(j1)) * cos(j5) + (-sin(j1) * cos(j3) \\ & - sin(j3) * cos(j1) * cos(j2)) * sin(j5)) * sin(j6) \\ & + 0,079 * (-(-sin(j1) * sin(j3) + cos(j1) * cos(j2) * cos(j3)) * sin(j4) \\ & - sin(j2) * cos(j1) * cos(j4)) * cos(j6) - 0,25 * (-sin(j1) * sin(j3) \\ & + cos(j1) * cos(j2) * cos(j3)) * sin(j4) - 0,25 * sin(j2) * cos(j1) * cos(j4) \\ & - 0,25 * sin(j2) * cos(j1) \end{aligned}$$

Cuando se implementan esas ecuaciones en un código python se obtiene una predicción de la posición:



The image shows a screenshot of a ROS environment. On the left, a tree view displays the robot's structure, with 'end\_effector\_link' selected. In the center, a terminal window shows the output of a Python script, displaying the predicted position (X, Y, Z) and orientation of the end effector. On the right, a 3D visualization of the robotic arm is shown, with the end effector highlighted in red.

Link	Parent	Position (X, Y, Z)	Orientation (Roll, Pitch, Yaw)	Relative Position	Relative Orientation
crane_x7_upper_arm...					
crane_x7_upper_arm...					
crane_x7_upper_arm...					
end_effector_link	crane_x7_grip...	0.061011; 0.38...	-0.11843; 0.75...	0; 0; 0.06	0; 0; 0; 1

```
wilber@ubuntu: ~  
wilber@ubuntu: ~/kinematics_ws/  
z= 0.447430457792809  
-----  
x= 0.0610113068044240  
y= 0.388290379035746  
z= 0.447430457792809  
-----
```

Esto solo para hacer cinemática directa...

Para cinemática inversa son más fórmulas (Jacobianos)



## Problemas de estos métodos matemáticos

- Es más tardado de implementar
- Es más difícil de generar secuencias
- Cálculos relativamente complicados
- Para generar cinemática inversa se requiere optimización de ecuaciones.

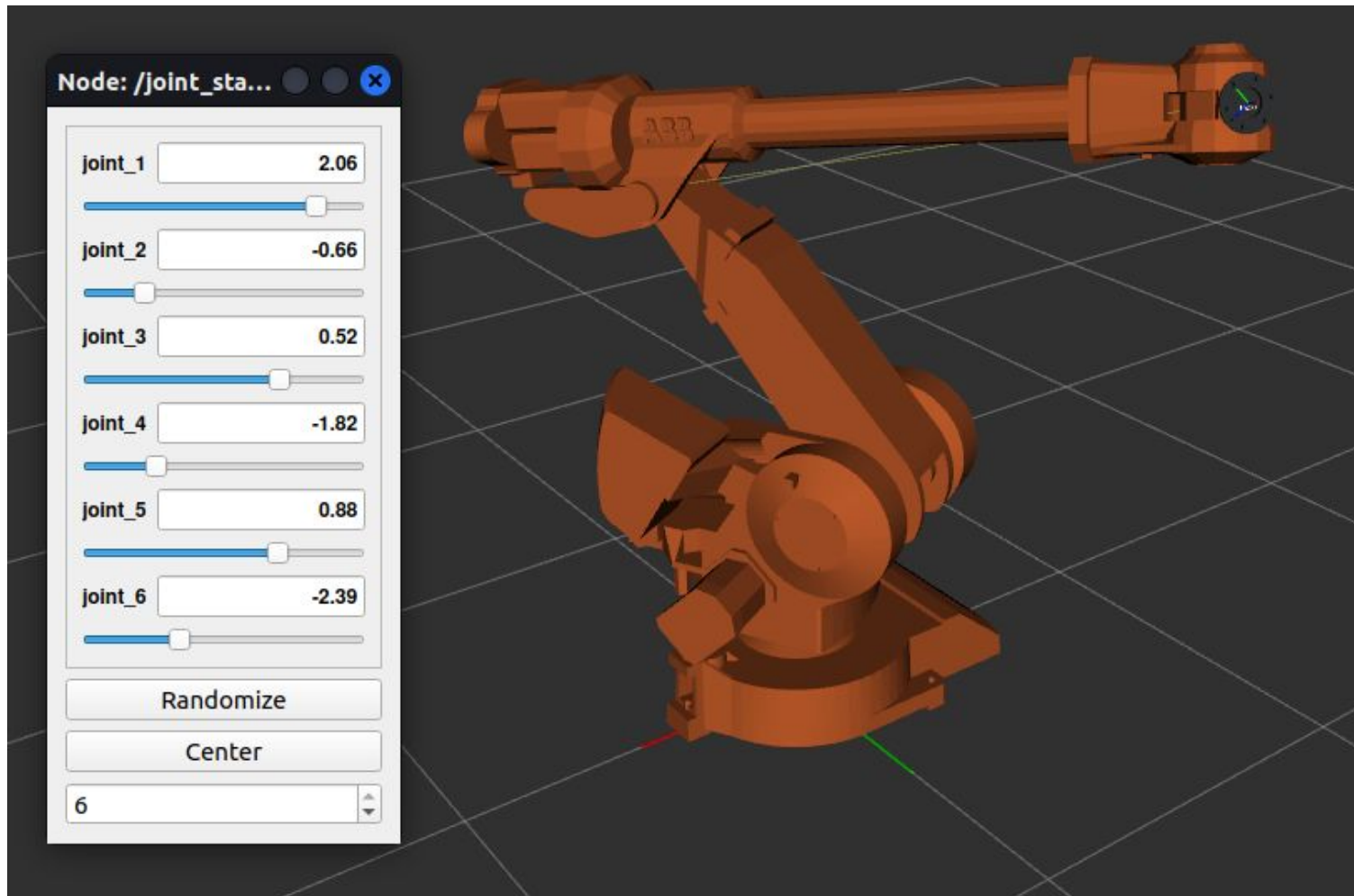
## ¿Como obtenemos un Robot para ROS?

1. Buscar un github, ej:  
ros github kuka
2. Buscar carpeta “support” o “Description”
3. Clonar/Descargar el github

## Para Generar Robot desde Solidworks

- Crear piezas
- Unirlas en un ensamblaje
- Usar una extensión (Add-ons) llamada “URDF2RVIZ”

# Simulación Básica



## MOVEIT

Moveit es un conjunto de packages desarrollados para programar de manera facil y rapida brazos robóticos.

Instalarlo:

```
sudo apt install ros-noetic-moveit
```

Se ejecuta Moveit Assistant:

```
roslaunch moveit_setup_assistant setup_assistant.launch
```

Para editar un Package de Moveit (cambiar nombre al pkg):

```
roslaunch irb4400_moveit setup_assistant.launch
```

## Se carga el URDF o XACRO:

**Movelt Setup Assistant**

Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

### Movelt Setup Assistant

These tools will assist you in creating a Semantic Robot Description Format (SRDF) file, various yaml configuration and many roslaunch files for utilizing all aspects of Movelt functionality.

Create new or edit existing?

Create New Movelt Configuration Package

Edit Existing Movelt Configuration Package

Load a URDF or COLLADA Robot Model

Specify the location of an existing Universal Robot Description Format or COLLADA file for your robot

arms/ws\_brazosRoboticos/src/abb\_irb4400\_support/urdf/irb4400l\_30\_243.xacro

Browse

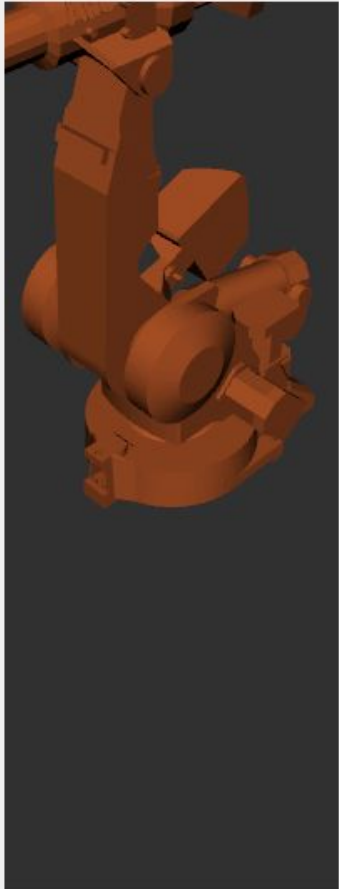
optional xacro arguments:

Success! Use the left navigation pane to continue.

100%

Load Files

☒ visual ☐ collision



# Se genera la matriz de colisión

Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

## Optimize Self-Collision Checking

This searches for pairs of robot links that can safely be disabled from collision checking, decreasing motion planning time. These pairs are disabled when they are always in collision, never in collision, in collision in the robot's default position, or when the links are adjacent to each other on the kinematic chain. Sampling density

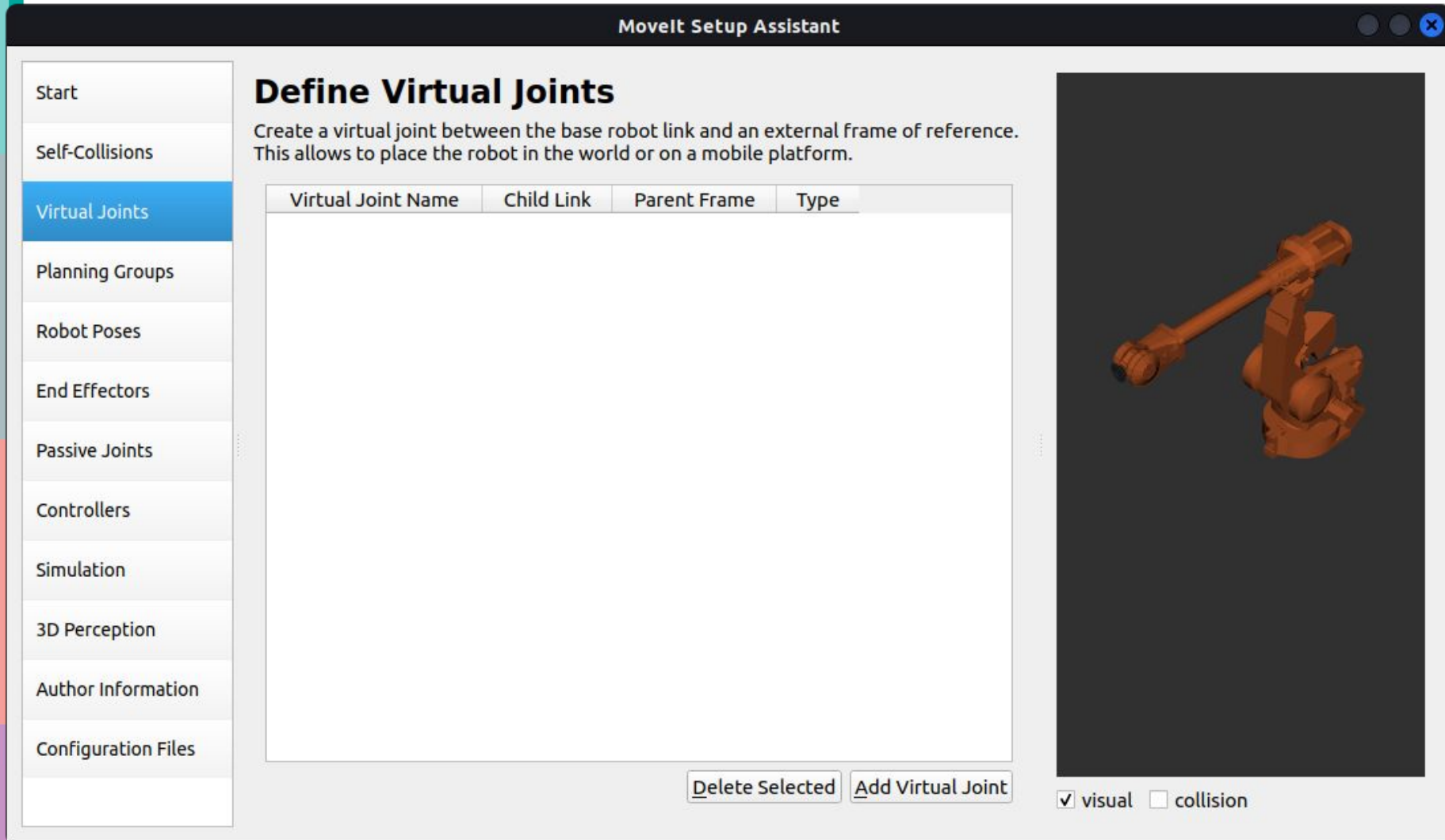
Sampling Density: Low  High 10000

Min. collisions for "always"-colliding pairs: 95%

	base_link	link_1	link_2	link_3	link_4	link_5	link_6
base_link		✓	✓	✓			
link_1	✓		✓		✓	✓	✓
link_2	✓	✓		✓	✓	✓	✓
link_3	✓		✓		✓	✓	✓
link_4		✓	✓	✓		✓	✓
link_5		✓	✓	✓	✓		✓
link_6		✓	✓	✓	✓	✓	

link name filter  ☐ linear view ☒ matrix view  ☒ visual ☐ collision

## Se “ancla el robot al piso”





Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

## Define Virtual Joints

Create a virtual joint between the base robot link and an external frame of reference. This allows to place the robot in the world or on a mobile platform.

Virtual Joint Name:

irb4400

Child Link:

base\_link

Parent Frame Name:

world

Joint Type:

fixed

Save

Cancel

☒ visual ☐ collision

# Crea un Planning Group

## MoveIt Setup Assistant

Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

## Define Planning Groups

Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.

### Create New Planning Group

#### Kinematics

Group Name:	<input type="text" value="irb4400"/>
Kinematic Solver:	<input type="text" value="kdl_kinematics_plugin/KDLKinematicsPlugin"/>
Kin. Search Resolution:	<input type="text" value="0.005"/>
Kin. Search Timeout (sec):	<input type="text" value="0.005"/>
Goal Joint Tolerance (m rad):	<input type="text" value="0.0001"/>
Goal Position Tolerance (m):	<input type="text" value="0.0001"/>
Goal Orientation Tolerance (rad):	<input type="text" value="0.0010"/>
Kin. parameters file:	<input type="text"/> ...

#### OMPL Planning

Group Default Planner:	<input type="text" value="None"/>
------------------------	-----------------------------------

### Next, Add Components To Group:

Recommended:

Add Kin. Chain

Add Joints

Advanced Options:

Add Subgroups

Add Links

Cancel

☒ visual ☐ collision



Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

## Define Planning Groups

Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.

### Edit 'irb4400' Kinematic Chain

Robot Links

- ▼ base\_link
  - base
  - ▼ link\_1
    - ▼ link\_2
      - ▼ link\_3
        - ▼ link\_4
          - ▼ link\_5
            - ▼ link\_6
              - tool0

Base Link Tip Link [Expand All](#) [Collapse All](#)☒ visual ☐ collision

# Se asigna poses del robot (opcional)

Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

## Define Robot Poses

Create poses for the robot. Poses are defined as sets of joint values for particular planning groups. This is useful for things like *home position*. The *first* listed pose will be the robot's initial pose in simulation.

	Pose Name	Group Name
1	init_pose	irb4400

Show Default Pose


Movelt

Edit Selected

Delete Selected

Add Pose

☒ visual ☐ collision



MoveIt Setup Assistant

Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

## Define Robot Poses

Create poses for the robot. Poses are defined as sets of joint values for particular planning groups. This is useful for things like *home position*. The *first* listed pose will be the robot's initial pose in simulation.

Pose Name:  
init\_pose

Planning Group:  
irb4400

joint\_1

1.5707

joint\_2

0.0000

joint\_3

0.0000

joint\_4

0.0000

joint\_5

0.0000


joint\_6

0.0000

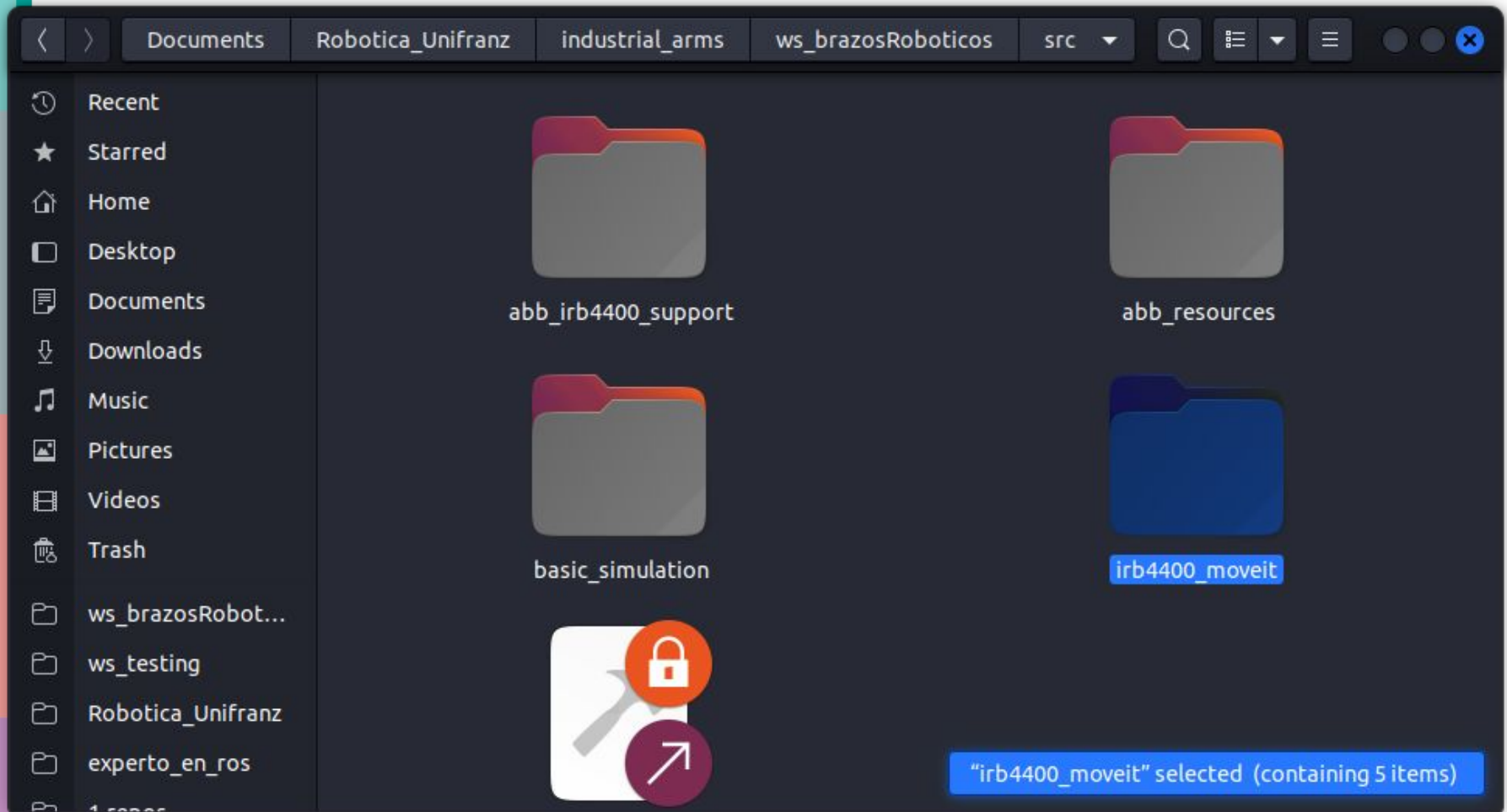
Save

Cancel

☒ visual ☐ collision



## Crear una carpeta para guardar el package





# Seleccionar la carpeta en “browse” y generar el package

Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

## Generate Configuration Files

Create or update the configuration files package needed to run your robot with MoveIt. Uncheck files to disable them from being generated - this is useful if you have made custom changes to them. Files in orange have been automatically detected as changed.

### Configuration Package Save Path

Specify the desired directory for the MoveIt configuration package to be generated. Overwriting an existing configuration package directory is acceptable. Example: `/u/robot/ros/panda_moveit_config`

ents/Robotica\_Unifranz/industrial\_arms/ws\_brazosRoboticos/src/irb4400\_moveit

Browse

Check files you want to be generated:

☒ package.xml

☒ CMakeLists.txt

☒ config/

☒ config/abb\_irb4400l\_30\_243.srdf

☒ config/ompl\_planning.yaml

☒ config/chomp\_planning.yaml

☒ config/stomp\_planning.yaml

☒ config/kinematics.yaml

☒ config/joint\_limits.yaml

☒ config/cartesian\_limits.yaml

☒ config/fake\_controllers.yaml

☒ config/simple\_moveit\_controllers.yaml

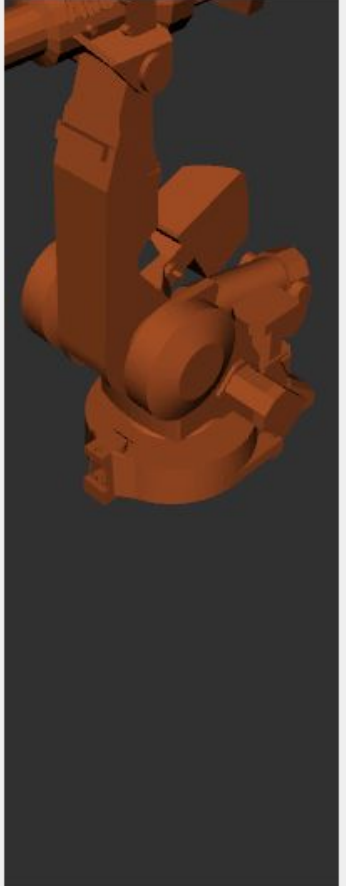
☒ config/qazabo\_controllers.yaml

Defines a ROS package

Generate Package

Exit Setup Assistant

☒ visual ☐ collision



## Práctica:

1. Realizar una simulación simple con un Robot Industrial
2. Realizar una simulación con Moveit de un Robot Industrial
  - Planear y ejecutar mediante RVIZ un movimiento con el Robot

Sugerencias: El robot puede ser de KUKA, ABB, Fanuc, Universal Robots.