## 1. Crear el espacio de trabajo

Primero, creamos el espacio de trabajo para ROS 2. Nos aseguramos de estar en nuestro directorio home:

```
C/C++
cd ~
mkdir -p prueba/src
cd prueba
colcon build
source install/setup.bash
```

---

## 2. Crear el paquete de interfaces (Servicio)

Este paquete define el **servicio** (`DesiredPosition.srv`) que usaremos para enviar y recibir la posición deseada.

**Ubicación:** **`~/prueba/src`**
Ejecutamos:

```
C/C++
cd ~/prueba/src
ros2 pkg create cinematica_inversa_pkg --build-type ament_cmake
cd cinematica_inversa_pkg
mkdir srv
cd srv
touch DesiredPosition.srv
```

📄 **Editar** **`DesiredPosition.srv`**
Editamos el archivo con:

```
C/C++
code .
```

Agregamos la siguiente definición:

```
C/C++
# Solicitud (Request): Posición deseada (x, y)
float64 x
float64 y
---
# Respuesta (Response): Angulos obtenidos (o valores de error si no es
posible)
float64 theta1
float64 theta2
```

**Ubicación: `~/prueba/src/cinematica_inversa_pkg`**
Ahora editamos `CMakeLists.txt` y agregamos:

Editamos el archivo con:

```
C/C++
code .
```

Añadimos lo siguiente:

```
C/C++
find_package(rosidl_default_generators REQUIRED)
rosidl_generate_interfaces(${PROJECT_NAME}
  srv/DesiredPosition.srv
)
```

**Ubicación: `~/prueba/src/cinematica_inversa_pkg`**
Editamos `package.xml` para incluir dependencias:

Editamos el archivo con:

```
C/C++
code .
```

Agregamos:

```
C/C++
<build_depend>rosidl_default_generators</build_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

**Ubicación:** `~/prueba`
Compilamos y verificamos:

```
C/C++
cd ~/prueba
colcon build --packages-select cinematica_inversa_pkg
source install/setup.bash
ros2 interface show cinematica_inversa_pkg/srv/DesiredPosition
```

Debe mostrar:

```
C/C++
float64 x
float64 y
---
float64 theta1
float64 theta2
```

---

## 3. Crear el paquete de nodos

**Ubicación:** `~/prueba/src`
Ahora creamos el paquete que contendrá nuestros nodos:

```
C/C++
cd ~/prueba/src
ros2 pkg create cinematica_inversa --build-type ament_python --dependencies
rclpy cinematica_inversa_pkg
cd cinematica_inversa
```

---

## 4. Crear el Nodo de Servicio (`nodo_servicio.py`)

**Ubicación:** `~/prueba/src/cinematica_inversa/cinematica_inversa`
Creamos el archivo:

```
C/C++
touch nodo_servicio.py
chmod +x nodo_servicio.py
nano nodo_servicio.py
```

**Código de `nodo_servicio.py`**

Editamos el archivo con:

```
C/C++
code .
```

Agregamos:

```
Python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from cinematica_inversa_pkg.srv import DesiredPosition

#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from cinematica_inversa_pkg.srv import DesiredPosition

class ServicioPosicion(Node):
    def __init__(self):
        super().__init__('nodo_servicio')
        self.cli = self.create_client(DesiredPosition,
'desired_position')

        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Esperando el servicio de cinemática
inversa...')

    def send_request(self, x, y):
        request = DesiredPosition.Request()
        request.x = x
        request.y = y
```

```
        future = self.cli.call_async(request)
        rclpy.spin_until_future_complete(self, future)
        return future.result()

def main():
    rclpy.init()
    node = ServicioPosicion()

    while True:
        try:
            x = float(input("Ingrese x: "))
            y = float(input("Ingrese y: "))

            response = node.send_request(x, y)

            if response.theta1 != -1 and response.theta2 != -1:
                node.get_logger().info(f"Ángulos obtenidos:
theta1={response.theta1}, theta2={response.theta2}")
            else:
                node.get_logger().info("Posición NO alcanzable. Intente
nuevamente.")

        except ValueError:
            print("Ingrese valores numéricos.")

    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

---

## 5. Crear el Nodo de Cinemática Inversa (`nodo_cinematica.py`)

**Ubicación:** `~/prueba/src/cinematica_inversa/cinematica_inversa`
Creamos el archivo:

```
C/C++
touch nodo_cinematica.py
chmod +x nodo_cinematica.py
```

```
nano nodo_cinematica.py
```

Editamos el archivo con:

```
C/C++
code .
```

### Código de `nodo_cinematica.py`

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from cinematica_inversa_pkg.srv import DesiredPosition
from std_msgs.msg import Float64MultiArray
import math

class CinematicaInversa(Node):
    def __init__(self):
        super().__init__('nodo_cinematica')

        # Longitudes de los eslabones
        self.L1 = 7.5  # Longitud del primer eslabón
        self.L2 = 9.5  # Longitud del segundo eslabón

        # Crear el servicio
        self.srv = self.create_service(DesiredPosition,
'desired_position', self.ik_callback)

        #Crear el publicador
        self.publisher = self.create_publisher(Float64MultiArray,
'/joint_angles', 10)

        self.get_logger().info("Nodo de Cinemática Inversa Listo.")

    def ik_callback(self, request, response):
        x, y = request.x, request.y

        # Calcular cinemática inversa usando el modelo algebraico
        theta1, theta2 = self.calculate_inverse_kinematics(x, y)

#Restricciones
#Restricción1: Comprobar si la posición está en el rango de los eslabones
```

```
        distancia = math.sqrt(x**2 + y**2)
        if distancia < abs(self.L1 - self.L2) or distancia > (self.L1 +
self.L2):
            self.get_logger().info(f"Posición ({x}, {y}) fuera del rango
alcanzable.")
            response.theta1 = -1
            response.theta2 = -1
            return response

#Cálculo de los ángulos theta1 y theta2 (cinemática inversa de un brazo
de 2DOF)
        cos_theta2 = (x**2 + y**2 - self.L1**2 - self.L2**2) / (2 *
self.L1 * self.L2)

        if abs(cos_theta2) > 1:
            self.get_logger().info(f"Posición ({x}, {y}) no es alcanzable
debido a valores de cos_theta2 fuera de rango.")
            response.theta1 = -1
            response.theta2 = -1
            return response

        theta2 = math.degrees(math.acos(cos_theta2))
        theta1 = math.degrees(math.atan2(y, x) - math.atan2(self.L2 *
math.sin(math.radians(theta2)), self.L1 + self.L2 *
math.cos(math.radians(theta2))))

#Restricción 2: Si theta1 o theta2 están fuera del rango [-90, 90]
        if not (-90 <= theta1 <= 90) or not (-90 <= theta2 <= 90):
            self.get_logger().info(f"Ángulos fuera de rango:
theta1={theta1}, theta2={theta2}")
            response.theta1 = -1
            response.theta2 = -1
            return response

#Restricción 3: Si ambos ángulos son ±90° y del mismo signo, no es
alcanzable
        if abs(theta1) == 90 and abs(theta2) == 90 and (theta1 * theta2 >
0):
            self.get_logger().info(f"Configuración inválida:
theta1={theta1}, theta2={theta2} con mismo signo.")
            response.theta1 = float(-1)
            response.theta2 = float(-1)
            return response

#Si todas las restricciones se cumplen, la posición es alcanzable
        response.theta1 = theta1
        response.theta2 = theta2
```

```
        # Publicar en el tópico
        msg = Float64MultiArray()
        msg.data = [theta1, theta2]
        self.publisher.publish(msg)
        self.get_logger().info(f"Publicado en /joint_angles: {msg.data}")

        return response

#--------------------------------------
    def calculate_inverse_kinematics(self, x, y):
        # Calcular theta2 usando el modelo algebraico
        cos_theta2 = (x**2 + y**2 - self.L1**2 - self.L2**2) / (2 *
self.L1 * self.L2)
        cos_theta2 = max(min(cos_theta2, 1), -1)  # Asegurar que esté en
el rango [-1, 1]
        theta2 = math.acos(cos_theta2)  # Solución positiva
(configuración "arriba")
        # theta2 = -math.acos(cos_theta2)  # Solución negativa
(configuración "abajo")

        # Calcular theta1 usando el modelo algebraico
        alpha = math.atan2(y, x)
        beta = math.atan2(self.L2 * math.sin(theta2), self.L1 + self.L2 *
math.cos(theta2))
        theta1 = alpha - beta

        return theta1, theta2

def main():
    rclpy.init()
    node = CinematicaInversa()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

---

## 6. Configurar `setup.py`

**Ubicación:** `~/prueba/src/cinematica_inversa`
Editamos `setup.py`:

Editamos el archivo con:

```
C/C++
code .
```

Agregamos:

```
Python
entry_points={
    'console_scripts': [
        'cinematica_inversa_server =
cinematica_inversa.nodo_cinematica:main',
        'cinematica_inversa_client =
cinematica_inversa.nodo_servicio:main',
    ],
},
```

---

## 7. Compilación y Ejecución

**Ubicación:** `~/prueba`
Compilamos y cargamos la configuración:

```
C/C++
cd ~/prueba
colcon build --packages-select cinematica_inversa
source install/setup.bash
```

Ejecutamos el **servidor**:

```
C/C++
ros2 run cinematica_inversa cinematica_inversa_server
```

Ejecutamos el **cliente**:

```
C/C++
ros2 run cinematica_inversa cinematica_inversa_client
```

Prueba ingresando valores como `(10,10)`, `(12,8)`, `(8,12)`, `(0,17)`, **y revisa si se publican los ángulos.**