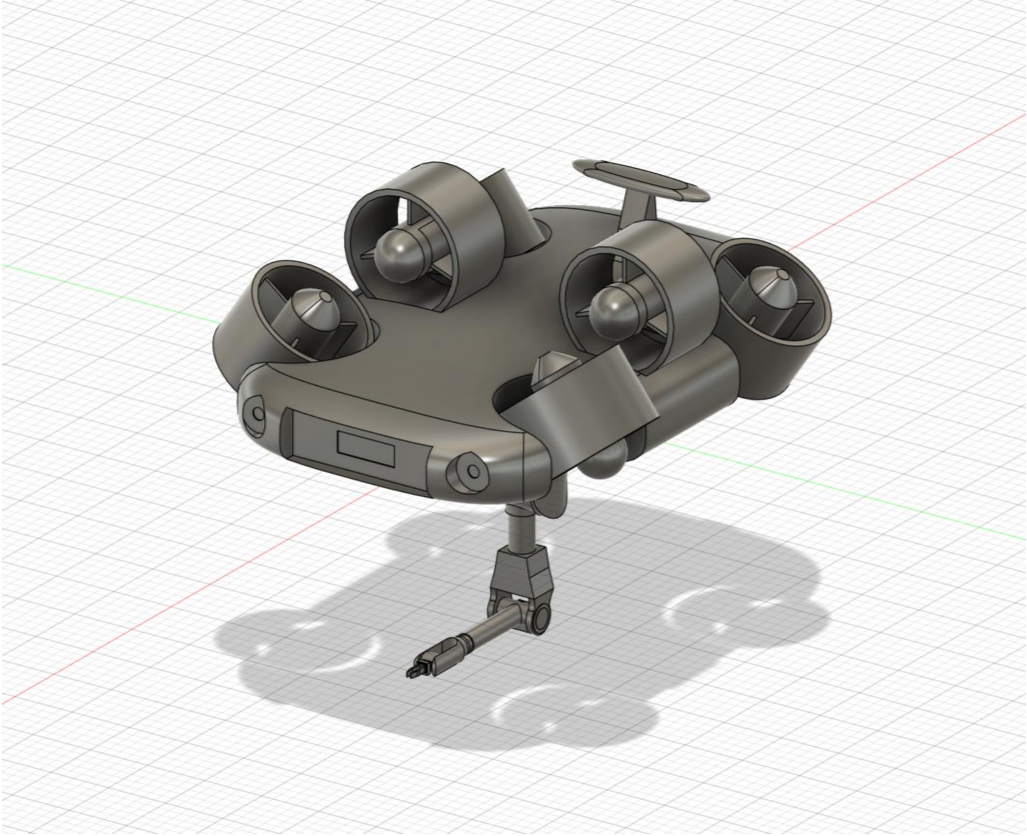
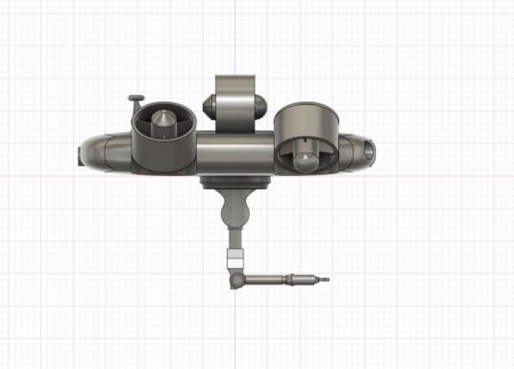
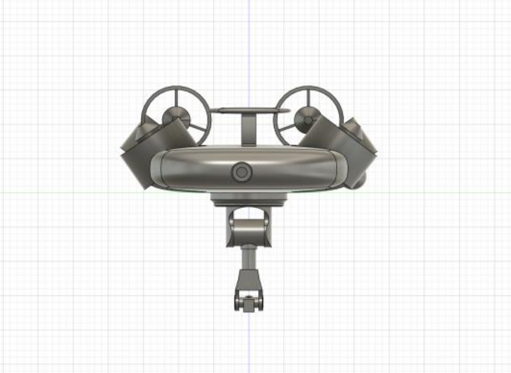
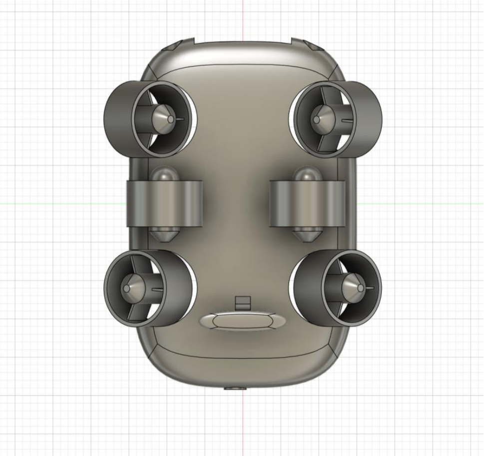
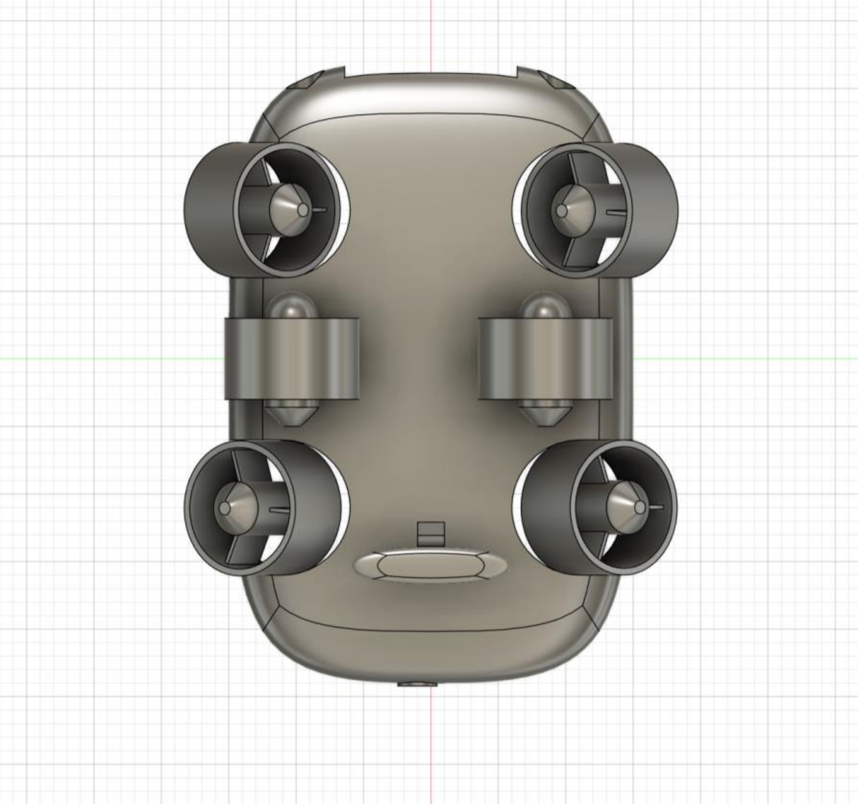
Dron acuático
NEM·O

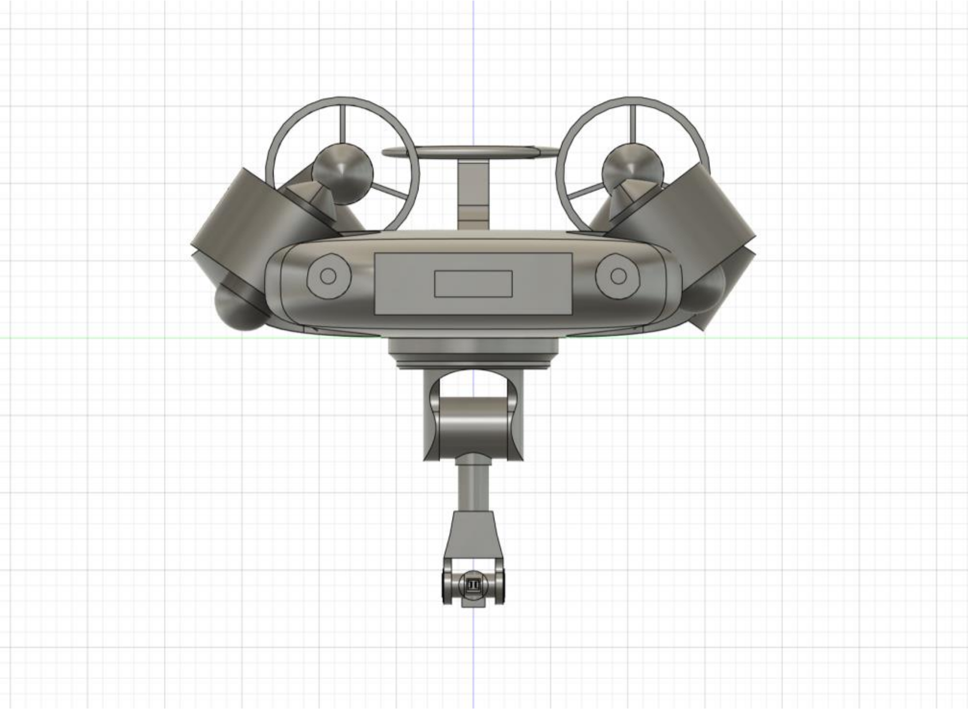
Hugo García   Héctor Fontán   Sofia Fernández

Índice

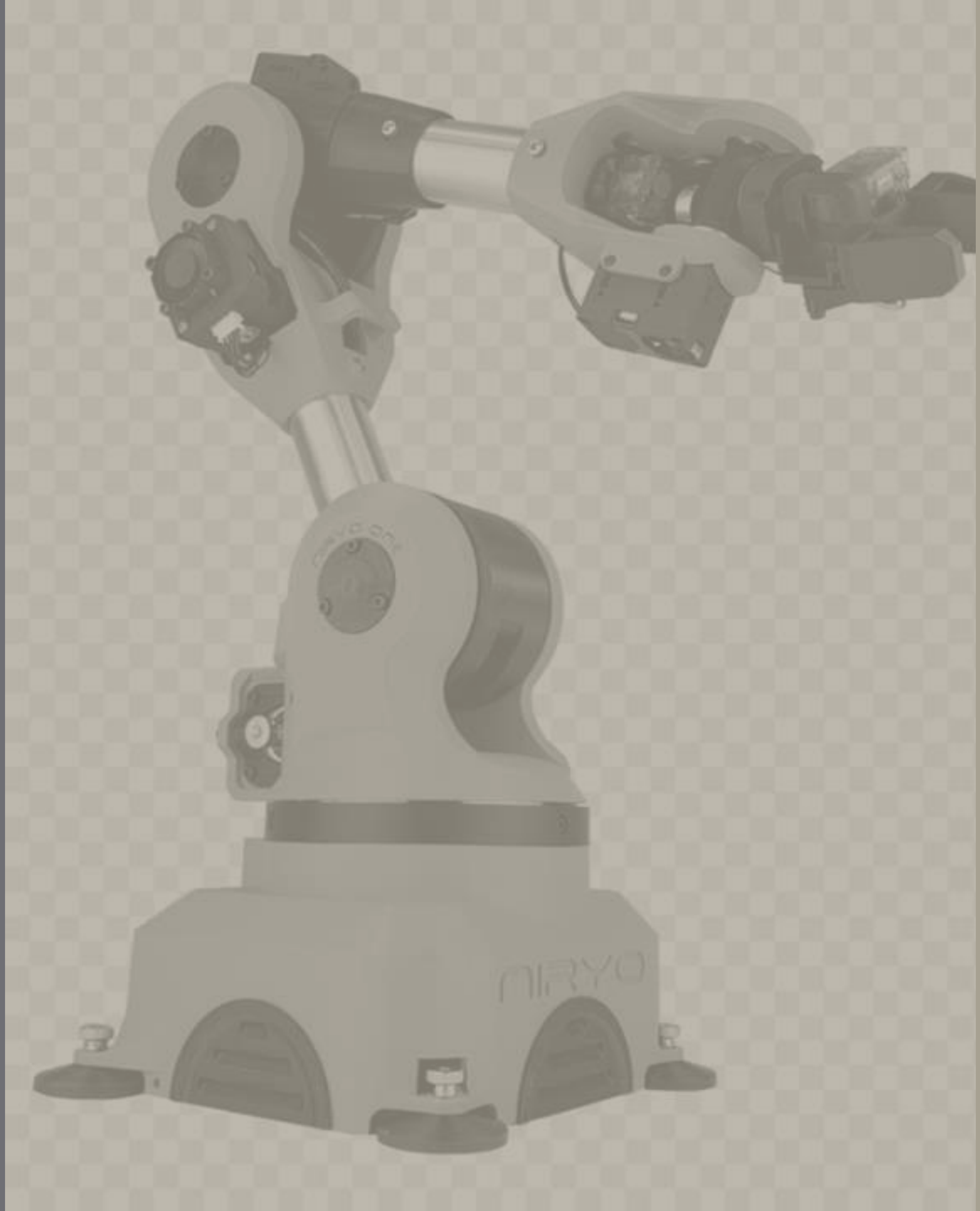


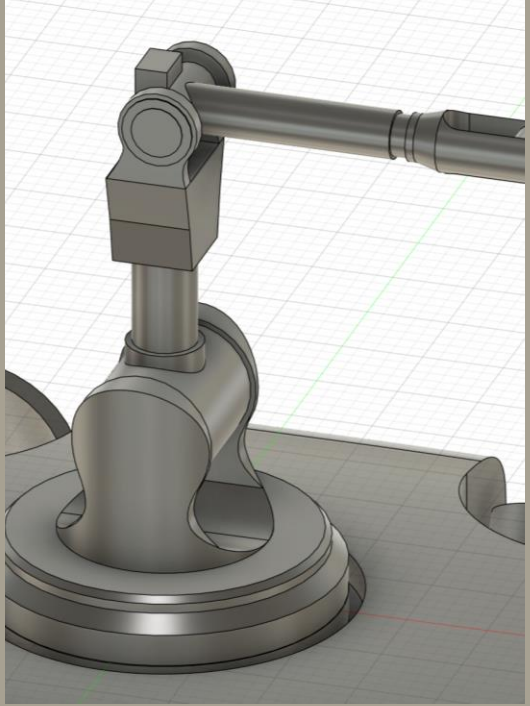


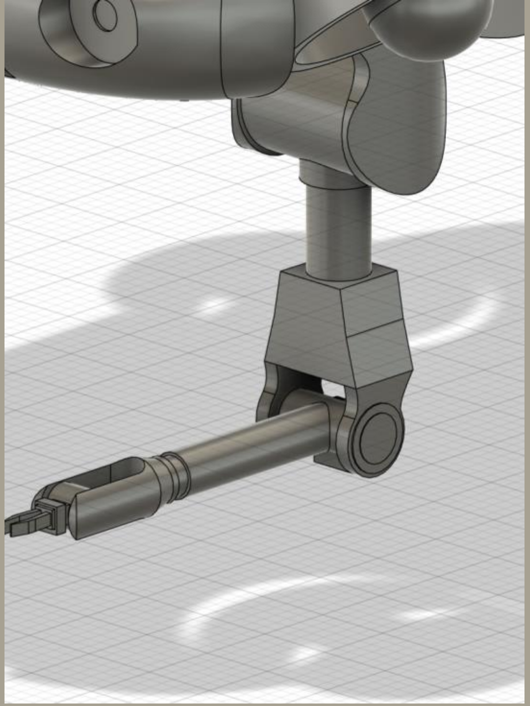
6 hélices distribuidas en 3 posiciones estratégicas:

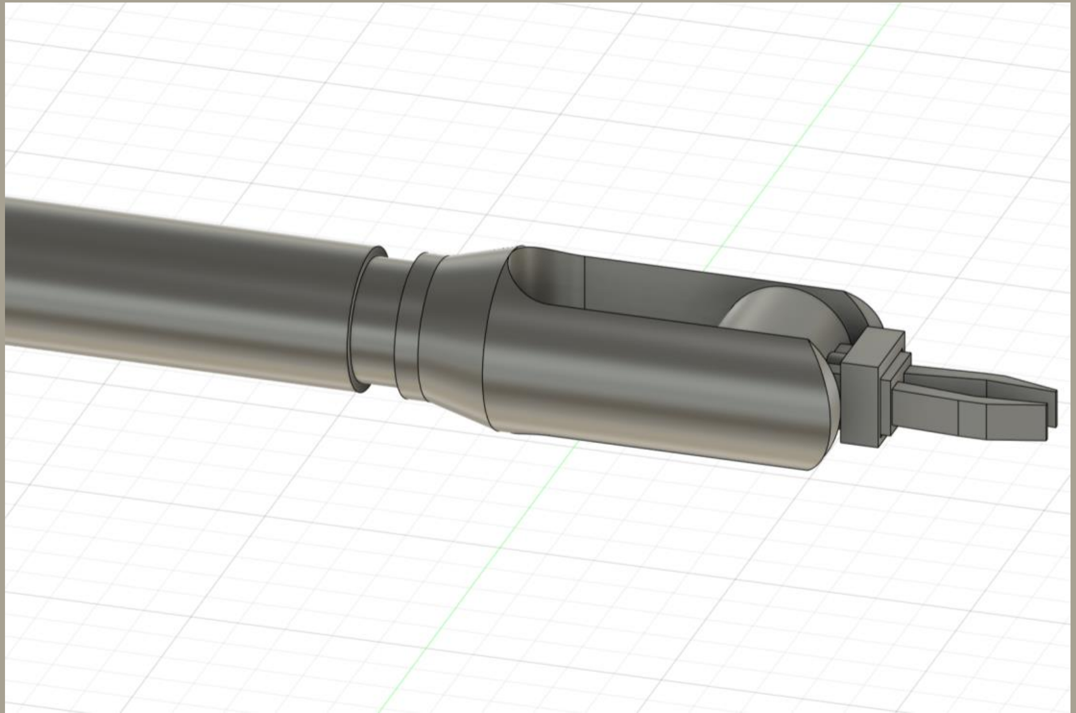
- 2 horizontales
- 2 verticales
- 2 inclinadas

- **Dos luces LED** para mejorar la visibilidad subacuática
- **Hueco para cámara**, para observación en tiempo real


NIRYO







Aplicaciones Prácticas:

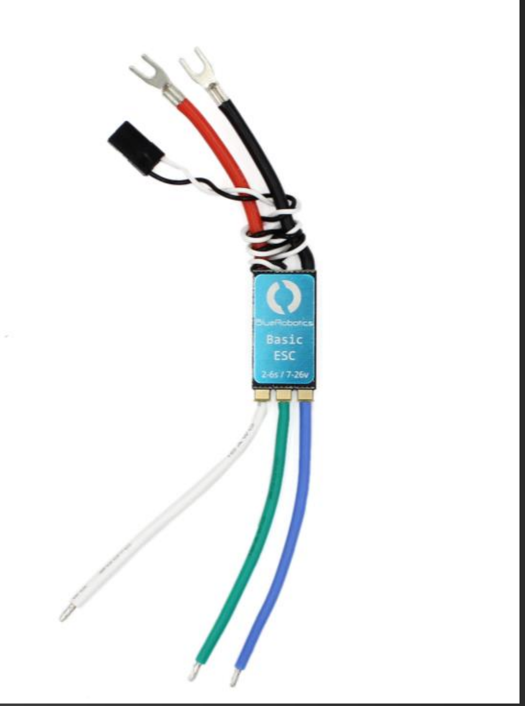
- Exploración submarina
- Recolección de objetos
- Recuperación de objetos pesados
- Monitoreo ambiental

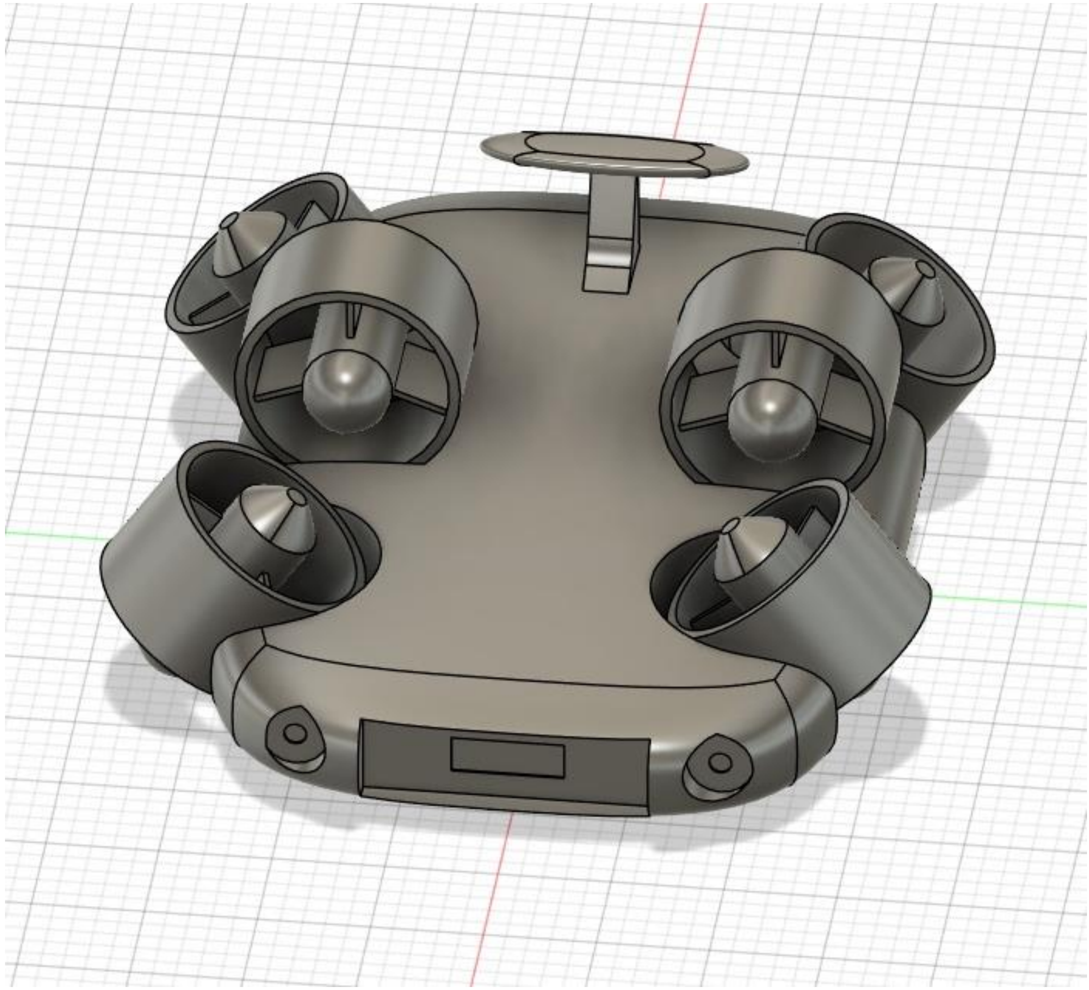
Ventajas:

- Disminución del riesgo
- Mayor precisión y eficiencia
- Posibilidad de operación con mayor duración bajo el agua
- Reducción de costes operativos a largo plazo

Estructura y materiales

- 6 motores T200 thruster
- 2 focos
- 1 camara
- Carcasa: plástico ABS
- Conexion: cable ethernet umbilical
- Servos sumergibles
- Baterias de 2500mah





Movimiento

Control

Archivo del robot

```python
class Robot:
    """ Clase que representa un robot con eslabones y sus propiedades."""
    def __init__(self, name: str):
        """ Inicializa una nueva instancia de la clase Robot. """
        self.name = name
        self.links = []
        self.ejes_helicoidales = "Los ejes helicoidales se calcularán al crear el robot."
        self.limits_dict = None # Inicializa limits_dict como None, se puede establecer más tarde
        print(f"\033[92\tmRobot '{self.name}' creado.\033[0m")
```

```python
class Link:
    """
    Clase que representa un eslabón de un robot manipulador.
    Un eslabón se define por su identificador único, longitud, tipo de articulación (revoluta o prismática),
    orientación, coordenadas de la articulación y el eje de la articulación.  Esta clase proporciona
    métodos para acceder a las propiedades del eslabón y calcular su eje helicoidal, que es fundamental
    para la cinemática exponencial.
    Attributes:
        id (str): Identificador único del eslabón.
        length (float): Longitud del eslabón.
        tipo (str): Tipo de articulación asociada al eslabón ("revolute" o "prismatic").
        orientation (numpy.ndarray): Orientación del eslabón representada como un vector.
        joint_coords (numpy.ndarray): Coordenadas de la articulación del eslabón en el espacio.
        joint_axis (numpy.ndarray): Eje de la articulación del eslabón.
    """
    def __init__(self, id, length, tipo, orientation, joint_coords, joint_axis, joint_limits):
        """ Inicializa una nueva instancia de la clase Link. """
        self.id = id
        self.length = length
        self.tipo = tipo
        self.orientation = np.array(orientation)
        self.joint_coords = np.array(joint_coords)
        self.joint_axis = np.array(joint_axis)
        self.joint_limits = joint_limits
```

```yaml
name: brazon_dron
# Función para imponer limites
# margen = np.deg2rad(5) # This line is a comment and not valid YAML for variable assignment.
                         # The string values from limits_dict will be moved as is.

links:
  # Base (J1)
  - id: Base
    length: 0.103
    type: revolute
    link_orientation: [0, 0, 1]
    joint_coords: [0, 0, 0.103]
    joint_axis: [0, 0, 1]
    joint_limits: (-3.054, 3.054)   # t0

  # Hombro (J2)
  - id: Hombro
    length: 0.080
    type: revolute
    link_orientation: [0, 1, 0]
    joint_coords: [0, 0, 0.080]
    joint_axis: [0, -1, 0]
    joint_limits: (-1.571, 0.6405)  # t1

  # Brazo (J3)
  - id: Brazo
    length: 0.210
```

```yaml
  # Codo (J4) - Corrección de eje
  - id: Codo
    length: 0.030
    type: revolute
    link_orientation: [0, 1, 0]
    joint_coords: [0.0415, 0, 0.030]
    joint_axis: [1, 0, 0]  # Eje X
    joint_limits: (-3.054, 3.054)   # t3

  # Antebrazo (J5) - Corrección de eje
  - id: Antebrazo
    length: 0.0415
    type: revolute
    link_orientation: [1, 0, 0]
    joint_coords: [0.180, 0, 0]
    joint_axis: [0, -1, 0]  # Eje -Y
    joint_limits: (-1.745, 1.745)   # t4

  # Muñeca (J6) - Corrección de eje y coordenadas
  - id: Muneca
    length: 0.180
    type: revolute
    link_orientation: [1, 0, 0]
    joint_coords: [0.0237, 0, -0.0055]  # Posición corregida
    joint_axis: [1, 0, 0]  # Eje X
    joint_limits: (-2.574, 2.574)   # t5
```

Ejemplo de uso

```python
class Robot:
    """ Clase que representa un robot con eslabones y sus propiedades."""
    def __init__(self, name: str):
        """ Inicializa una nueva instancia de la clase Robot. """
        self.name = name
        self.links = []
        self.ejes_helicoidales = "Los ejes helicoidales se calcularán al crear el robot."
        self.limits_dict = None # Inicializa limits_dict como None, se puede establecer más tarde
```

```yaml
name: brazon_dron
# Función para imponer limites
# margen = np.deg2rad(5) # This line is a comment and not valid YAML for variable assignment.
                        # The string values from limits_dict will be moved as is.

links:
  # Base (J1)
  - id: Base
    length: 0.103
    type: revolute
    link_orientation: [0, 0, 1]
    joint_coords: [0, 0, 0.103]
    joint_axis: [0, 0, 1]
    joint_limits: (-3.054, 3.054)   # t0

  # Hombro (J2)
  - id: Hombro
    length: 0.080
    type: revolute
    link_orientation: [0, 1, 0]
    joint_coords: [0, 0, 0.080]
    joint_axis: [0, -1, 0]
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

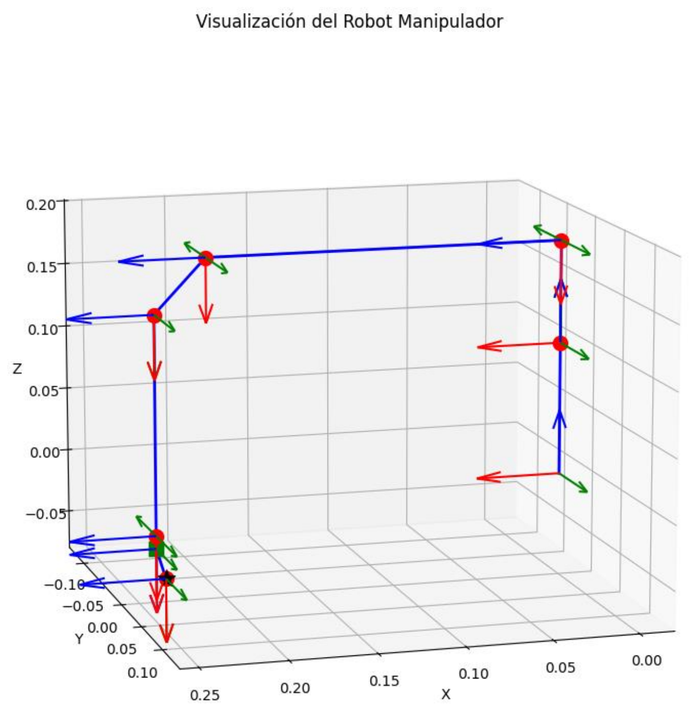
```
sh-5.2$ /usr/bin/python /home/heaven/regit/physics-II/class_robot_structure.py
        Robot 'brazo_dron' creado.
        Tiempo de ejecución de get_ejes_helicoidales: 0.0003 segundos


Robot 'brazo_dron' con 7 eslabones.
        El Eslabón 'Base' (revolute), coordenadas: [0.    0.    0.103], eje: [0 0 1], longitud: 0.103 y límites: (-3.054, 3.054)
        El Eslabón 'Hombro' (revolute), coordenadas: [0.   0.   0.08], eje: [ 0 -1  0], longitud: 0.08 y límites: (-1.571, 0.6405)
        El Eslabón 'Brazo' (revolute), coordenadas: [0.   0.   0.21], eje: [ 0 -1  0], longitud: 0.21 y límites: (-1.396, 1.571)
        El Eslabón 'Codo' (revolute), coordenadas: [0.0415 0.     0.03  ], eje: [1 0 0], longitud: 0.03 y límites: (-3.054, 3.054)
        El Eslabón 'Antebrazo' (revolute), coordenadas: [0.17 0.   0.  ], eje: [ 0 -1  0], longitud: 0.0415 y límites: (-1.745, 1.745)
        El Eslabón 'Extensor' (prismatic), coordenadas: [0.01 0.   0.  ], eje: [1 0 0], longitud: 0.1 y límites: (0.0, 0.15)
        El Eslabón 'Muneca' (revolute), coordenadas: [ 0.0237  0.     -0.0055], eje: [1 0 0], longitud: 0.18 y límites: (-2.574, 2.574)

Ejes helicoidales del robot:
        [    0.0000,     0.0000,     1.0000,    -0.0000,    -0.0000,    -0.0000]
        [    0.0000,    -1.0000,     0.0000,     0.1830,    -0.0000,    -0.0000]
        [    0.0000,    -1.0000,     0.0000,     0.3930,    -0.0000,    -0.0000]
        [    1.0000,     0.0000,     0.0000,    -0.0000,     0.4230,    -0.0000]
        [    0.0000,    -1.0000,     0.0000,     0.4230,    -0.0000,    -0.2115]
        [    0.0000,     0.0000,     0.0000,     1.0000,     0.0000,     0.0000]
        [    1.0000,     0.0000,     0.0000,    -0.0000,     0.4175,    -0.0000]

Límites de las articulaciones: {'joint_1': (-3.054, 3.054), 'joint_2': (-1.571, 0.6405), 'joint_3': (-1.396, 1.571), 'joint_4': (-3.054, 3.054), 'joint_5': (-1.745, 1.745), 'joint_6': (0.0, 0.15), 'joint_7': (-2.574, 2.574)}

Obtener_eje_de_giro
        Eslabón Base:       Eje de giro: +Z     ( sentido positivo →  == horario ↻ )
        Eslabón Hombro:     Eje de giro: -Y     ( sentido negativo ←  == antihorario ↺ )
        Eslabón Brazo:      Eje de giro: -Y     ( sentido negativo ←  == antihorario ↺ )
        Eslabón Codo:       Eje de giro: +X     ( sentido positivo →  == horario ↻ )
        Eslabón Antebrazo:  Eje de giro: -Y     ( sentido negativo ←  == antihorario ↺ )
        Eslabón Extensor:   Eje de giro: +X     ( sentido positivo →  == horario ↻ )
        Eslabón Muneca:     Eje de giro: +X     ( sentido positivo →  == horario ↻ )
sh-5.2$
```

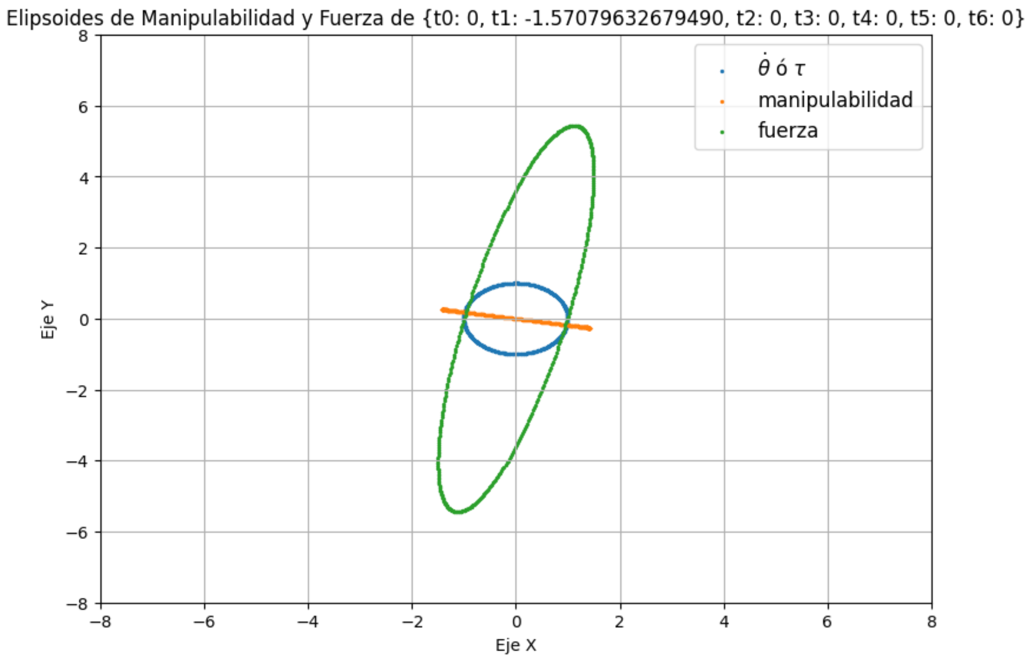
Jacobiana, Singularidades y Ellipsoides

```
PS C:\Users\Huxsby\Documents\repgit\physics-II> & C:/Users/Huxsby/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Huxsby/Docu
so_gen.py
        Robot 'niryo_one' creado.
        Tiempo de ejecución de get_ejes_helicoidales: 0.0000 segundos

Matriz de transformación homogénea inical T_d:
 [[1.  0.  0.  0.1]
 [0.  1.  0.  0.1]
 [0.  0.  1.  0.1]
 [0.  0.  0.  1. ]]

Vectores oritentation y p_xyz (distancia al objetivo):
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[0.1 0.1 0.1]

Extrayendo dastos del robot:
Ejes helicoidales del robot: [array([ 0.,  0.,  1., -0., -0., -0.]), array([ 0.   , -1.   ,  0.   ,  0.183, -0.   , -0.   ]), array([ 0.
ay([ 1.    ,  0.    ,  0.    , -0.    ,  0.423, -0.    ]), array([ 0.     , -1.     ,  0.     ,  0.423 , -0.     , -0.2215]), array([ 1.    ,  0.

Matriz Jacobiana del robot:
        Tiempo de cálculo de la Jacobiana del robot niryo_one: 0.0692 segundos

⎡0      sin(t0)                  sin(t0)  -sin(t1)*sin(t2)*cos...  -(-sin(t1)*cos(t0)*c...  ((-sin(t1)*cos(t0)*c...⎤
⎢0     -cos(t0)                 -cos(t0)  -sin(t0)*sin(t1)*sin...  -(-sin(t0)*sin(t1)*c...  ((-sin(t0)*sin(t1)*c...⎥
⎢1            0                        0  sin(t1)*cos(t2) + si...  -(-sin(t1)*sin(t2) +...  (-sin(t1)*sin(t2) + ...⎥
⎢0  0.183*cos(t0)  (0.21*cos(t1) + 0.18...  (sin(t1)*cos(t2) + s...  -(-(-sin(t0)*sin(t1)...  -(((-sin(t0)*sin(t1)...⎥
⎢0  0.183*sin(t0)  (0.21*cos(t1) + 0.18...  -(sin(t1)*cos(t2) + ...  (-(-sin(t1)*cos(t0)*...  (((-sin(t1)*cos(t0)*...⎥
⎣0            0  0.21*sin(t0)**2*sin(...  (-sin(t0)*sin(t1)*si...  (-(-sin(t0)*sin(t1)*...  (((-sin(t0)*sin(t1)*...⎦
```

heaven@cyborg-e16: ~/regit/physics-8

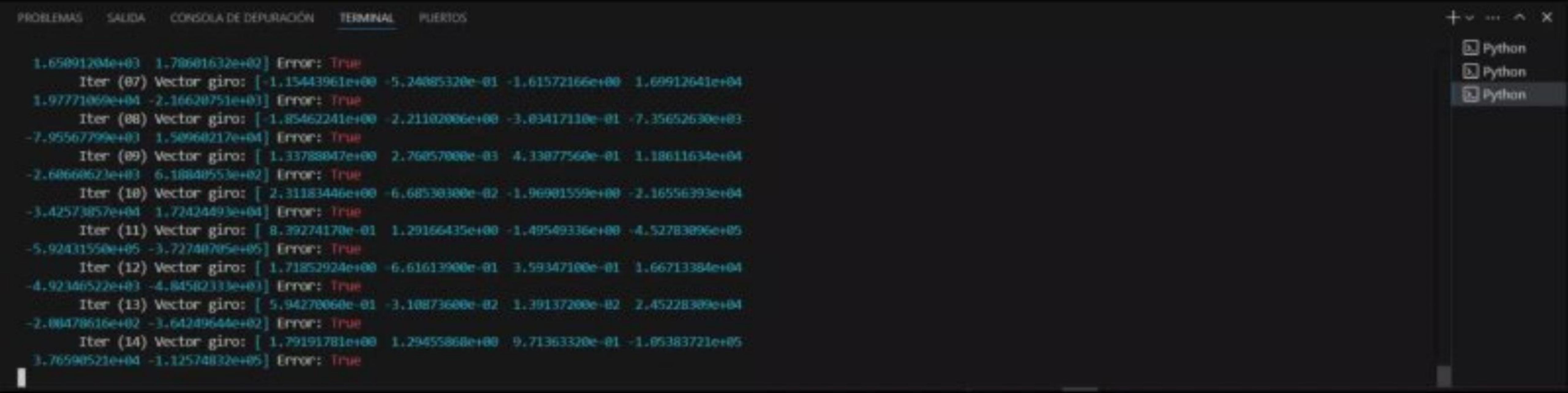
```
heaven@cyborg-e16:~/regit/physics-8$
```





Configuración Singular t1: -pi/2

Cinemática Inversa con prismática



PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

Python
Python
Python

```
 1.65091204e+03  1.78601632e+02] Error: True
      Iter (07) Vector giro: [-1.15443961e+00 -5.24085320e-01 -1.61572166e+00  1.69912641e+04
 1.97771069e+04 -2.16620751e+03] Error: True
      Iter (08) Vector giro: [-1.85462241e+00 -2.21102006e+00 -3.03417110e-01 -7.35652630e+03
-7.95567799e+03  1.50960217e+04] Error: True
      Iter (09) Vector giro: [ 1.33788847e+00  2.76057000e-03  4.33077560e-01  1.18611634e+04
-2.60660623e+03  6.18840553e+02] Error: True
      Iter (10) Vector giro: [ 2.31183446e+00 -6.68530300e-02 -1.96901559e+00 -2.16556393e+04
-3.42573857e+04  1.72424493e+04] Error: True
      Iter (11) Vector giro: [ 8.39274170e-01  1.29166435e+00 -1.49549336e+00 -4.52783096e+05
-5.92431550e+05 -3.72740705e+05] Error: True
      Iter (12) Vector giro: [ 1.71852924e+00 -6.61613900e-01  3.59347100e-01  1.66713384e+04
-4.92346522e+03 -4.84582333e+03] Error: True
      Iter (13) Vector giro: [ 5.94270060e-01 -3.10873600e-02  1.39137200e-02  2.45228309e+04
-2.08478616e+02 -3.64249644e+02] Error: True
      Iter (14) Vector giro: [ 1.79191781e+00  1.29455868e+00  9.71363320e-01 -1.05383721e+05
 3.76590521e+04 -1.12574832e+05] Error: True
```

Cinemática Inversa sin prismática

```
PS C:\Users\Huxsby\Documents\repgit\physics-II> & C:/Users/Huxsby/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Huxsby/Docu
so_gen.py
        Robot 'niryo_one' creado.
        Tiempo de ejecución de get_ejes_helicoidales: 0.0000 segundos

Matriz de transformación homogénea inical Tsd:
 [[1.  0.  0.  0.1]
 [0.  1.  0.  0.1]
 [0.  0.  1.  0.1]
 [0.  0.  0.  1. ]]

Vectores oritentation y p_xyz (distancia al objetivo):
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[0.1 0.1 0.1]

Extrayendo dastos del robot:
Ejes helicoidales del robot: [array([ 0., 0., 1., -0., -0., -0.]), array([ 0.   , -1.   , 0.   , 0.183, -0.   , -0.   ]), array([ 0.
ay([ 1.    ,                                           0.    , 0.423 , -0.    , -0.2215]), array([ 1.    ,  0.

Matriz Jacobiana del robot:
        Tiempo de cálculo de la Jacobiana del robot niryo_one: 0.0692 segundos

⎡0      sin(t0)              sin(t0)  -sin(t1)*sin(t2)*cos...  -(-sin(t1)*cos(t0)*c...  ((-sin(t1)*cos(t0)*c...⎤
⎢0     -cos(t0)             -cos(t0)  -sin(t0)*sin(t1)*sin...  -(-sin(t0)*sin(t1)*c...  ((-sin(t0)*sin(t1)*c...⎥
⎢1            0                    0  sin(t1)*cos(t2) + si...  -(-sin(t1)*sin(t2) +...  (-sin(t1)*sin(t2) + ...⎥
⎢0  0.183*cos(t0)  (0.21*cos(t1) + 0.18...  (sin(t1)*cos(t2) + s...  -(-(-sin(t0)*sin(t1)...  -(((-sin(t0)*sin(t1)...⎥
⎢0  0.183*sin(t0)  (0.21*cos(t1) + 0.18...  -(sin(t1)*cos(t2) + ...  (-(-sin(t1)*cos(t0)*...  (((-sin(t1)*cos(t0)*...⎥
⎣0            0  0.21*sin(t0)**2*sin(...  (-sin(t0)*sin(t1)*si...  (-(-sin(t0)*sin(t1)*...  (((-sin(t0)*sin(t1)*...⎦
```

Cargar y preparar los datos

```
problema_cinematico_inverso_gen.py > CinematicaInversa
def CinematicaDirecta(ejes, thetas, M):
    return calcular_T_robot(ejes, thetas, M)

def CinematicaInversa(rob ... 10, error_pet=1.00000000e-10, error
    """Resolución del prob ... para el ...
    tiempo = time.time()
    if robot is None:
        raise ValueError("El robot no está definido. Por favor, carga un robot válido.")
```

```
c:/Users/Huxsby/Documents/repgit/physics-II/problema_cinematico_inverso_gen.py
        Robot 'brazon_dron' creado.
        Tiempo de ejecución de get_ejes_helicoidales: 0.0010 segundos

Matriz de transformación homogénea inical Tsd:
 [[1.  0.  0.  0.1]
 [0.  1.  0.  0.1]
 [0.  0.  1.  0.1]
 [0.  0.  0.  1. ]]

Vectores oritentation y p_xyz (distancia al objetivo):
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[0.1 0.1 0.1]

Extrayendo dastos del robot:
Ejes helicoidales del robot: [array([ 0.,  0.,  1., -0., -0., -0.]), array([ 0.    , -1.    ,  0.    ,  0.183, -0.    , -0.    ]), array([ 0.    , -1.    ,  0.    ,  0.393, -0.    , -0.    ]), array([ 1.    ,  0.    ,  0.    , -0.    ,  0.423, -0.    ]), array([ 0.     , -1.     ,  0.     ,  0.423 , -0.     , -0.2215]), array([ 1.     ,  0.     ,  0.     , -0.     ,  0.4175, -0.     ])]
```

Calcular a Jacobiana e Aplicación de Newton-Rapson

```
Matriz Jacobiana del robot:
        Tiempo de cálculo de la Jacobiana del robot brazon_dron: 0.0659 segundos


⎡0       sin(t0)                   sin(t0)  -sin(t1)*sin(t2)*cos...  -(-sin(t1)*cos(t0)*c...  ((-sin(t1)*cos(t0)*c...⎤
⎢0      -cos(t0)                  -cos(t0)  -sin(t0)*sin(t1)*sin...  -(-sin(t0)*sin(t1)*c...  ((-sin(t0)*sin(t1)*c...⎥
⎢1             0                         0  sin(t1)*cos(t2) + si...  -(-sin(t1)*sin(t2) +...  (-sin(t1)*sin(t2) + ...⎥
⎢0  0.183*cos(t0)  (0.21*cos(t1) + 0.18...  (sin(t1)*cos(t2) + s...  -(-(-sin(t0)*sin(t1)...  -(((-sin(t0)*sin(t1)...⎥
⎢0  0.183*sin(t0)  (0.21*cos(t1) + 0.18...  -(sin(t1)*cos(t2) + ...  (-(-sin(t1)*cos(t0)*...  (((-sin(t1)*cos(t0)*...⎥
⎣0             0  0.21*sin(t0)**2*sin(...  (-sin(t0)*sin(t1)*si...  (-(-sin(t0)*sin(t1)*...  (((-sin(t0)*sin(t1)*...⎦

Iteraciones de la cinemática inversa:
        Iter (01) Vector giro: [-0.5515536  -0.2195781  -0.63305736  0.16421197  0.03437667  0.0433757 ] Error: True
        Iter (02) Vector giro: [ 0.24799932 -0.58861567  0.48525428  0.17396815  0.1261261  -0.10180005] Error: True
        Iter (03) Vector giro: [-1.16657742  0.35054606  1.65808028  0.08151227 -0.19482742 -0.10003129] Error: True
        Iter (04) Vector giro: [ 0.86825908  1.95456123  1.22254207 -0.14922991  0.19254147 -0.03083387] Error: True
        Iter (05) Vector giro: [ 1.34061553  1.22404644 -1.1496243  -0.28032914  0.19685582 -0.12242959] Error: True
        Iter (06) Vector giro: [-0.21848297  1.34397331 -0.46216792 -0.24913253  0.01436248  0.08728673] Error: True
        Iter (07) Vector giro: [ 0.22282501  0.50765971 -0.82152227 -0.15573278  0.0998981   0.04872153] Error: True
        Iter (08) Vector giro: [ 0.14380425 -0.23029403 -0.16817013  0.01198244  0.04114294 -0.02376379] Error: True
        Iter (09) Vector giro: [-0.02208585  0.0242982  -0.02712138 -0.00355161 -0.00151679  0.00462933] Error: True
        Iter (10) Vector giro: [-1.4520e-05 -4.9185e-04 -5.9256e-04 -7.9000e-06  4.1570e-05 -4.2630e-05] Error: True
        Iter (11) Vector giro: [-7.0e-08  2.0e-07 -1.2e-07 -3.0e-08  0.0e+00  2.0e-08] Error: True
        Iter (12) Vector giro: [ 0.  0.  0.  0. -0. -0.] Error: False → Solución valida
        Tiempo de cálculo total de la cinemática inversa: 0.6219 segundos
```

Analizar Resultados

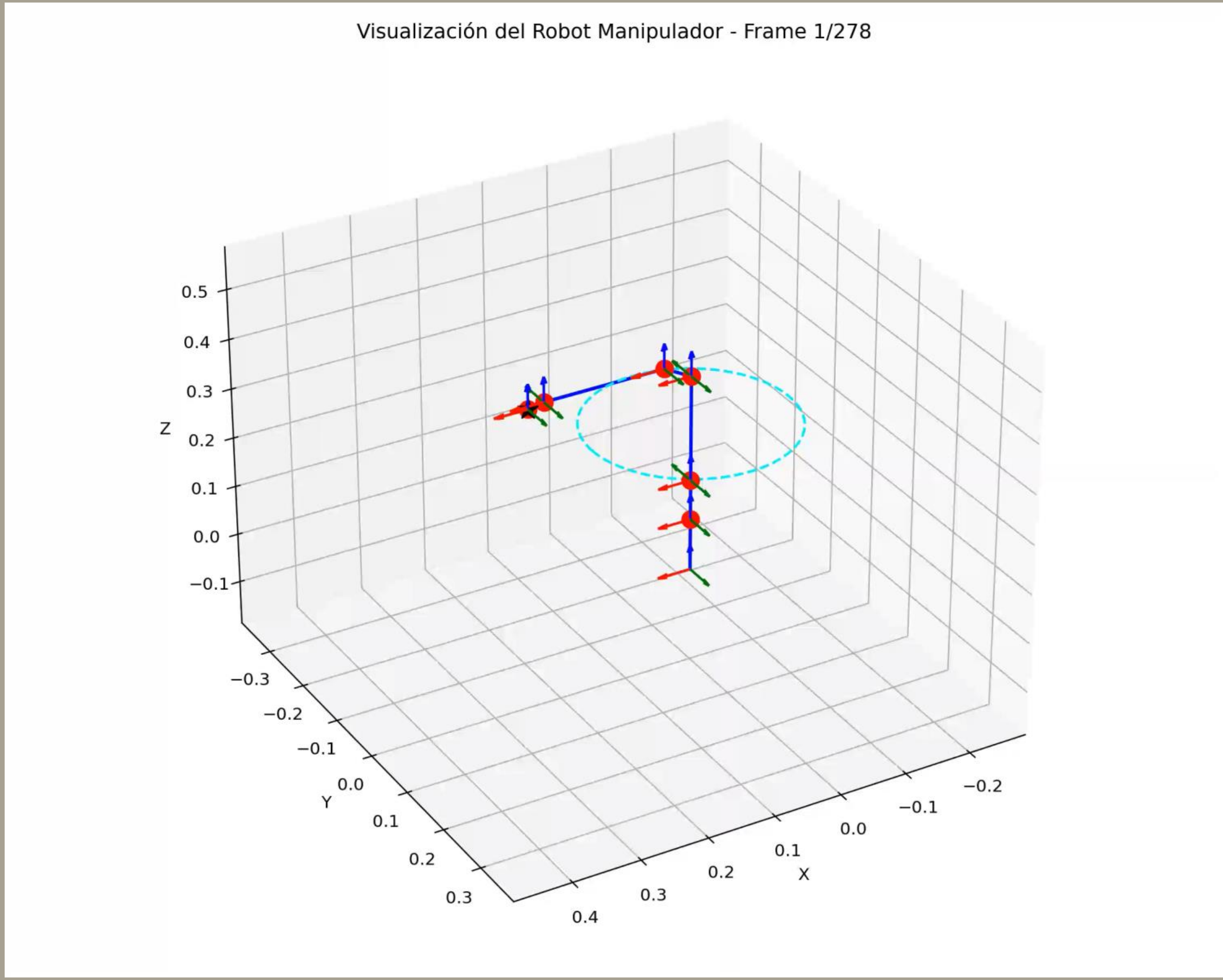
```
problema_cinematico_inverso_gen.py > CinematicaInversa
def CinematicaDirecta(ejes, thetas, M):
    return calcular_T_robot(ejes, thetas, M)

def CinematicaInversa(robot: Robot, thetas_actuales=None, p_xyz=[0.1, 0.1, 0.1], RPY=[0, 0, 0], error_oet=1.00000000e-10, error_pet=1.00000000e-10, error
```

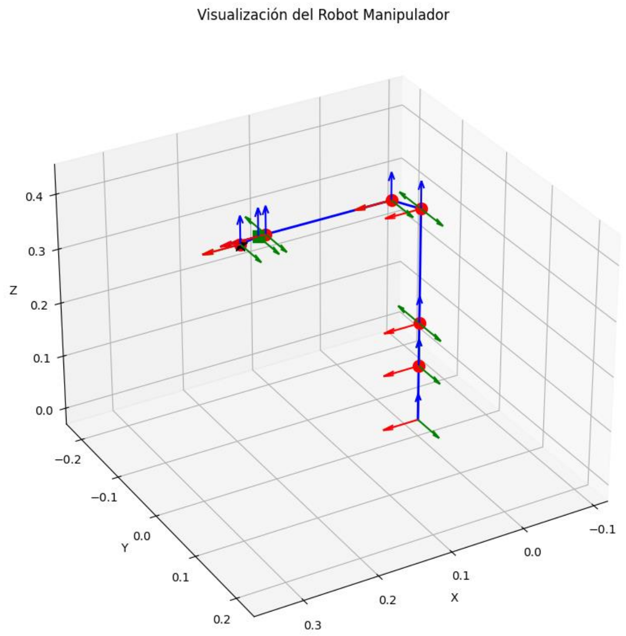
```
Coordenadas de las articulaciones:
 [13.507635282249646, -3.4211564466914752, -2.421332532622629, 23.863200132904787, -7.292415453429542, -17.80694589480073]
Error en w: 0.0
Error en v: 0.0
Número de iteraciones: 12

Matriz de transformación homogénea final Tsd re-calculada:
 [[ 1.  -0.   0.   0.1]
 [ 0.   1.   0.   0.1]
 [-0.  -0.   1.   0.1]
 [ 0.   0.   0.   1. ]]

Matriz de transformación homogénea final Tsd original:
 [[1.  0.  0.  0.1]
 [0.  1.  0.  0.1]
 [0.  0.  1.  0.1]
 [0.  0.  0.  1. ]]

Las thetas por las que ha pasado el robot son:
        [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        [0.0, 0.8962, -2.3296, 18.1818, 1.4334, -18.1818]
        [-1.488, 1.0469, -1.7265, 17.2602, 0.7481, -18.1882]
        [7.5519, -1.1609, -2.0872, 25.5368, -5.567, -21.5004]
        [12.6993, -3.106, -1.4916, 21.4522, -6.8108, -16.5607]
        [13.8522, -3.0792, -2.5378, 22.3361, -5.5467, -17.652]
        [13.3424, -2.9849, -2.4208, 21.9155, -6.7292, -15.9717]
        [13.3148, -3.4228, -2.5493, 23.8359, -6.1183, -17.9144]
        [13.4656, -3.5419, -2.3754, 24.1897, -7.2337, -18.0838]
        [13.5252, -3.4231, -2.4234, 23.8408, -7.271, -17.7697]
        [13.5077, -3.4211, -2.4213, 23.8641, -7.2922, -17.8074]
        [13.5076, -3.4212, -2.4213, 23.8632, -7.2924, -17.8069]
```

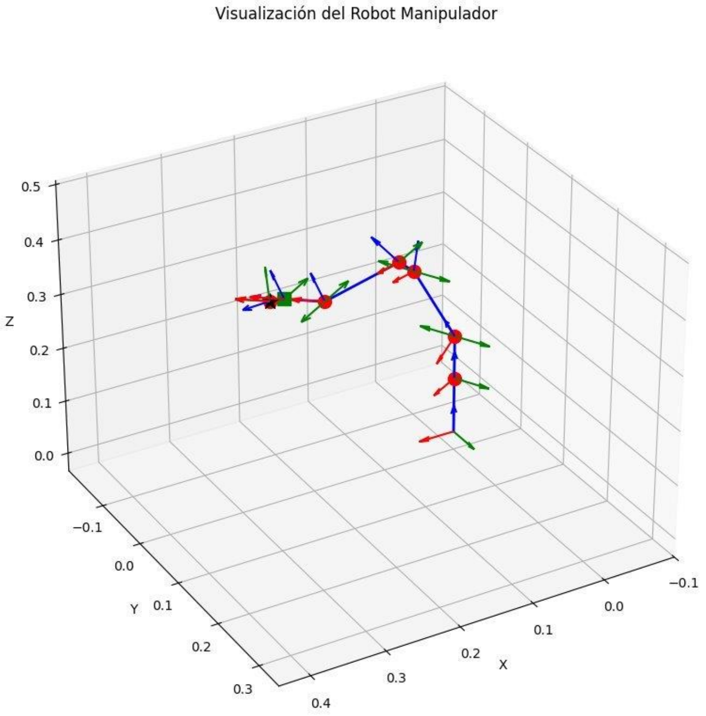
Simulación

```
PS C:\Users\Huxsby\Documents\repgit\physics-II> & C:/Users/Huxsby/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Huxsby/Docu
so_gen.py
        Robot 'niryo_one' creado.
        Tiempo de ejecución de get_ejes_helicoidales: 0.0000 segundos

Matriz de transformación homogénea inical Tsd:
 [[1.  0.  0.  0.1]
 [0.  1.  0.  0.1]
 [0.  0.  1.  0.1]
 [0.  0.  0.  1. ]]

Vectores oritentation y p_xyz (distancia al objetivo):
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[0.1 0.1 0.1]

Extrayendo dastos del robot:
Ejes helicoidales del robot: [array([ 0.,  0.,  1., -0., -0., -0.]), array([ 0.    , -1.    ,  0.    ,  0.183, -0.    , -0.    ]), array([ 0.
ay([ 1.    ,  0.    ,  0.    , -0.    ,  0.423, -0.    ]), array([ 0.     , -1.     ,  0.     ,  0.423 , -0.     , -0.2215]), array([ 1.     ,  0.

Matriz Jacobiana del robot:
        Tiempo de cálculo de la Jacobiana del robot niryo_one: 0.0692 segundos


⎡0       sin(t0)                   sin(t0)  -sin(t1)*sin(t2)*cos...  -(-sin(t1)*cos(t0)*c...  ((-sin(t1)*cos(t0)*c...⎤
⎢0      -cos(t0)                  -cos(t0)  -sin(t0)*sin(t1)*sin...  -(-sin(t0)*sin(t1)*c...  ((-sin(t0)*sin(t1)*c...⎥
⎢1             0                         0  sin(t1)*cos(t2) + si...  -(-sin(t1)*sin(t2) +...  (-sin(t1)*sin(t2) + ...⎥
⎢0  0.183*cos(t0)  (0.21*cos(t1) + 0.18...  (sin(t1)*cos(t2) + s...  -(-(-sin(t0)*sin(t1)...  -(((-sin(t0)*sin(t1)...⎥
⎢0  0.183*sin(t0)  (0.21*cos(t1) + 0.18...  -(sin(t1)*cos(t2) + ...  (-(-sin(t1)*cos(t0)*...  (((-sin(t1)*cos(t0)*...⎥
⎣0             0  0.21*sin(t0)**2*sin(...  (-sin(t0)*sin(t1)*si...  (-(-sin(t0)*sin(t1)*...  (((-sin(t0)*sin(t1)*...⎦
```

Aplicación Cinemática Inversa

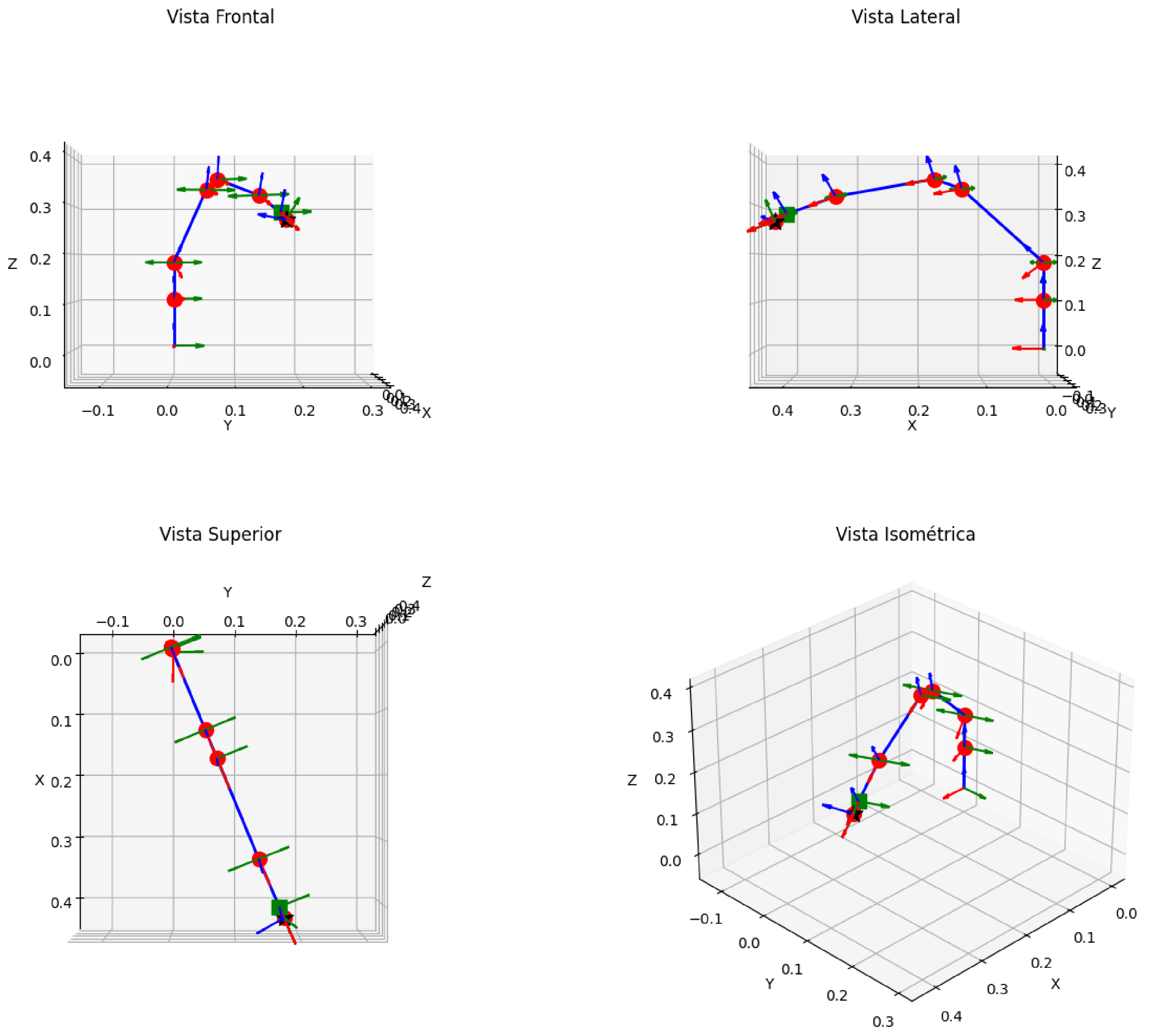




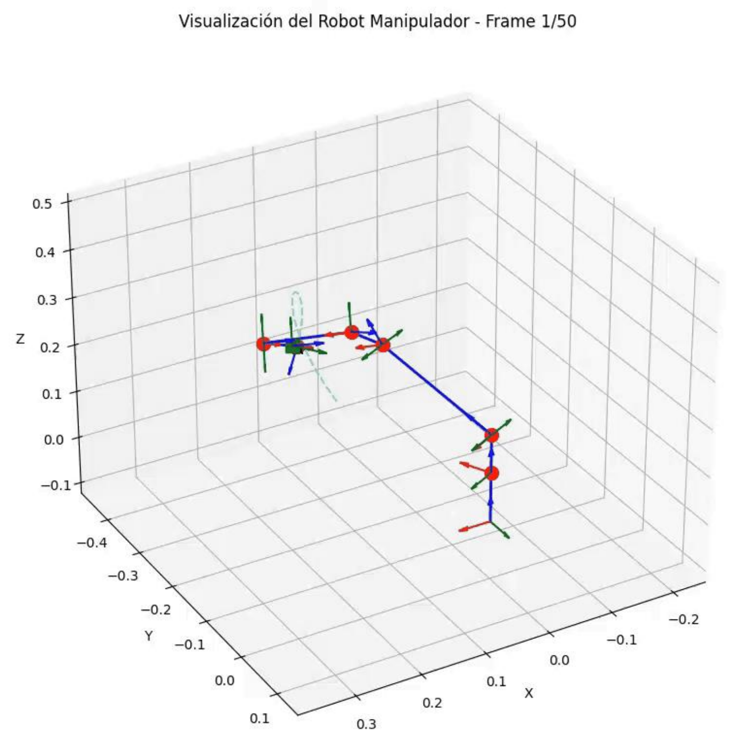


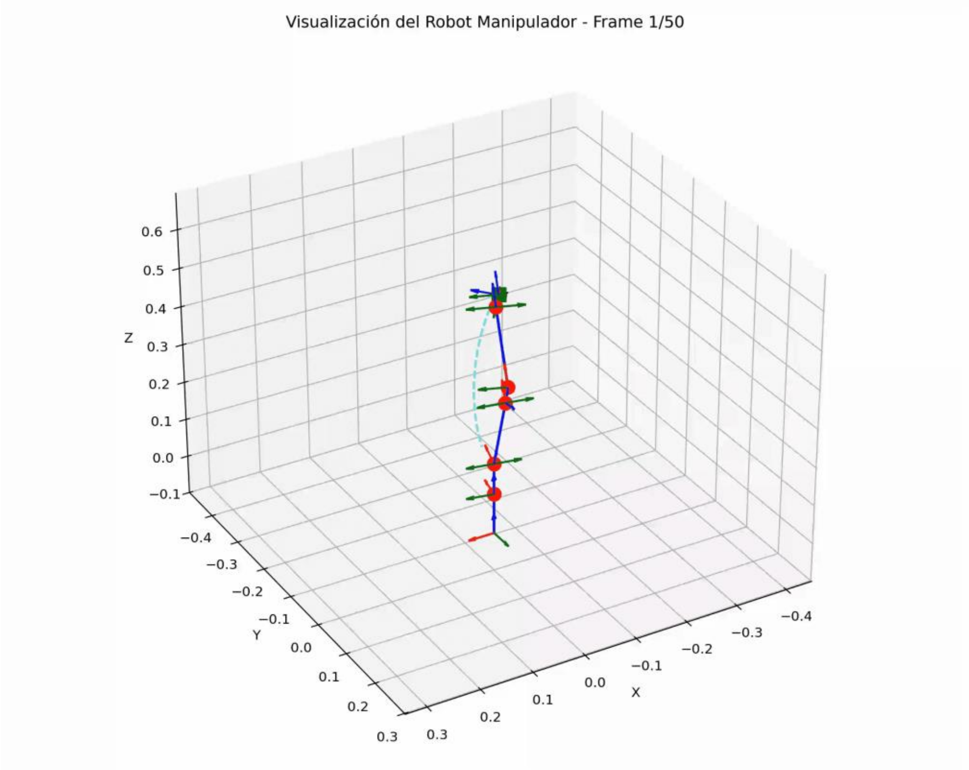
Configuración Nula

Configuración Estática



Vista Frontal
Vista Lateral
Vista Superior
Vista Isométrica
X
Y
Z





Animaciones con trayectorias procedurales

Webgrafía

- https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/
- https://makeblock.com.ar/robot-submarino-casero/
- https://www.nauticexpo.es/prod/umbilicals-international/product-58261-423703.html
- https://elvuelodeldrone.com/drones-profesionales/drones-acuaticos/chasing-m2-pro/
- https://github.com/Huxsby/physics-II