

BOTBLOQ: Ecosistema integral para el diseño, fabricación y programación de robots DIY

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI)

EXPEDIENTE: IDI-20150289

Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

ACRÓNIMO DEL PROYECTO: BOTBLOQ



Centro para el
Desarrollo
Tecnológico
Industrial



UNIÓN EUROPEA
Fondo Europeo de
Desarrollo Regional (FEDER)
Una manera de hacer Europa

ENTREGABLE E4.6. INTERFAZ DE INTEGRACIÓN EN BITBLOQ Y BOTBLOQ

RESUMEN DEL DOCUMENTO

Este entregable define la interfaz de integración con BOTBLOQ de la arquitectura robótica modular realizada en el paquete de trabajo 4. Además, se presenta el proceso seguido en Bitbloq para la generación de bloques que permitan la programación por parte de cualquier usuario de todos los robots *tipo* generados en el proyecto.

20 de enero de 2017

Índice

1. Introducción	3
2. Descripción de la Interfaz	4
2.1. Interfaz de robots móviles con ruedas	4
2.2. Interfaz de robots manipuladores	6
2.3. Interfaz de robots ápodos	7
2.4. Interfaz de robots bípedos	9
2.5. Interfaz de robots hexápodos	12
3. Programación en BOTBLOQ de un robot modular tipo rover	14
3.1. Servidor en la Intel Edison (no ROS)	14
3.2. Adaptación de Bitbloq para la programación del robot móvil	16
3.3. Versión en ROS	19
4. Programación en BOTBLOQ de un robot modular tipo manipulador	21
4.1. Servidor en la Intel Edison (no ROS)	22
4.2. Adaptación de Bitbloq para la programación del robot manipulador	25
4.3. Versión en ROS	27
5. Programación en BOTBLOQ de un robot modular tipo serpiente (solo ROS)	32
5.1. Nodo de ROS para el control de los robots serpientes	33
5.2. Implementación en Bitbloq de los bloques de control de la serpiente	37
6. Programación en BOTBLOQ de un robot modular tipo humanoide (solo ROS)	40
6.1. Nodo de Ros para el control de los robots humanoides	40
6.2. Implementacion en Bitbloq de los bloques de control del robot humanoide	42
7. Programación en BOTBLOQ de un robot modular tipo hexápodo (solo ROS)	43

7.1. Implementacion en Bitbloq de los bloques de control del robot he- xápodo	46
A. Cliente I2C en los módulos activos	48
A.0.1. cliente I2c en modulos servo	48
A.0.2. cliente I2c en módulos sensores	49
A.0.3. cliente I2c en modulos dinamixel	51
B. Código en ROS del Nodo Servidor para el control de los movi- mientos de robots serpiente	55
C. Código en ROS del Nodo Servidor para el control de los movi- mientos de robots humanoides	58
C.1. Caminada hacia delante	58
C.2. Caminada hacia atras	75
C.3. Girar sobre un pie (rotar sobre si mismo)	81
C.4. Giros	89
C.5. Reposo	100
C.6. Envío de comandos por I2C a los módulos de las articulaciones	106
D. Código en ROS del Nodo Servidor para el control de los movi- mientos de robots hexápodos	112
D.1. Caminada adelante/atras	112
D.2. Caminada lateral	131
D.3. Giros	139

1. Introducción

El presente entregable muestra la definición de la interfaz con BOTBLOQ que permite la programación de los robots modulares creados en el paquete de trabajo 4. Para la definición de dicha interfaz es necesario presentar los métodos de funcionamiento de cada robot y la implementación en el modulo de control de la arquitectura robótica (módulo que incorpora la Intel Edison) de sus controladores.

Para el control de todos los robots generados se ha usado ROS que es uno de los entornos de trabajo (framework) más usados hoy en día y que proporciona una gran variedad de métodos, planificadores y controladores de última generación desarrollados por la comunidad científica en los últimos años. Para ello, se ha realizado un proceso que automatiza la descripción de robots en URDF por medio de XACRO. Este método permite la generación automática de controladores en ROS a través de su descripción. Esto permite que diferentes robots parametrizados, basados en configuraciones *tipo*, puedan ser controlados desde ROS y por ende programados en BOTBLOQ.

Por lo tanto, en este documento presentamos la interfaz definida en ROS de los robot *tipo* (móvil con ruedas, manipulador, bípedo, ápodo, hexápodo). Todo el código mostrado en este documento corresponde a la versión prediseñada de estos robots. Los robots procedentes de un proceso de generación automática por medio de ontologías no es mostrado ya que existen infinitas posibilidades. Sin embargo, al basarse estos robots en parametrizaciones de las configuraciones *tipo* y gracias a la utilización de Xacro y URDF como ya se ha comentado, el procedimiento de programación no varía en Bitbloq siendo su control interno transparente al usuario.

El procedimiento de programación es similar para todos los robots presentados. Básicamente, cuando un usuario quiere programar un robot *tipo* en Bitbloq lo elige de la lista de robots disponibles en la ventana hardware de Bitbloq. Una vez seleccionado, el usuario de Bitbloq dispone de todos los bloques asignados a dicho robot que provienen de la definición de la interfaz que se muestra en el apartado siguiente. Con esos bloques el usuario puede realizar el programa que quiera y que permita al robot alcanzar el comportamiento deseado. Ese programa de bloques es traducido a python y subido automáticamente por *ssh* al controlador del robot (Intel Edison). Una vez en el controlador el programa se ejecuta como un *nodo* de ROS y se conecta internamente por medio de las *actionlib* de ROS al *nodo*

controlador de dicho robot.

2. Descripción de la Interfaz

La interfaz para BOTBLOQ se corresponde con la definición de los métodos de funcionamiento para cada robot, por ejemplo avanzar, rodar, etc. Estos métodos generarán bloques de programación en Bitbloq que los usuarios podrán utilizar de la misma manera que utilizan otros bloques matemáticos, de control, etc.

Los métodos que se desarrollan para los diferentes robots están en continuo proceso de ampliación y mejora. Esto es debido a la complejidad de los robots y a las infinitas posibilidades que sus configuraciones permiten (y que son ámbito de investigación de numerosos trabajos como se muestra en el entregable 4.7). Así pues, a continuación se describen los métodos, a fecha de redacción de este entregable, para cada robot. En apartados posteriores se mostrará cómo han sido incluidos en Bitbloq y su uso junto a los controladores dedicados de los robots.

2.1. Interfaz de robots móviles con ruedas

Los robots móviles con ruedas diseñados para usarse con Bitbloq están compuestos por dos o cuatro módulos activos de rotación continua y dos módulos activos con sensores de infrarrojos. Un ejemplo de un robot construido con nuestra plataforma modular a partir de esta configuración *tipo* puede verse en la Figura 1. Para dichos robots se han definido los siguientes métodos de funcionamiento como interfaz a Bitbloq:

- **Avanzar/retroceder** con una velocidad de x m/s. Este método tiene como parámetros de entrada la dirección de movimiento (avanzar o retroceder) y la velocidad lineal a la que se quiere que se mueva el robot. No produce ninguna variable de salida. Como resultado hará que, en el caso de avanzar, las ruedas del lado derecho del robot se muevan en sentido horario y las del lado izquierdo en sentido antihorario a una velocidad de giro determinada por el radio de las mismas y la velocidad lineal deseada. En el caso de retroceder la dirección de movimiento de las ruedas será la inversa.

- **Girar derecha/derecha** con una velocidad de $\theta \text{ rad/s}$. Este método tiene como parámetros de entrada la dirección de giro (izquierda/derecha) y la velocidad angular a la que se quiere que gire el robot. No produce ninguna variable de salida. Como resultado hará que, en el caso de giro a la derecha, las ruedas de ambos lados del robot se muevan en sentido antihorario horario a una velocidad de giro determinada por el radio de las mismas, la distancia entre ejes del robot y la velocidad angular deseada. En el caso de giro a la derecha la dirección de movimiento de las ruedas será horaria.
- **Leer sensores** infrarrojos izquierdo/derecho. Este método tiene como parámetro de entrada el módulo sensor que se quiere leer (izquierdo o derecho). Como parámetro de salida tiene la lectura de dicho módulo sensor (blanco o negro)
- **Parar**. No tiene parámetros de entrada ni de salida.



Figura 1: Robot obtenido a partir de la configuración *tipo* de robot móvil

2.2. Interfaz de robots manipuladores

La configuración de robot manipulador que actualmente está integrada en BOTBLOQ es una configuración de robot antropomorfo de 3 grados de libertad (*gdl*) y una pinza angular (aunque existe otra versión de 6 *gdl* cuya configuración también será incluida en BOTBLOQ en breve como mejora). Un ejemplo de un robot construido con nuestra plataforma modular a partir de esta configuración tipo puede verse en la Figura 2. El número de grados de libertad establece tres variables de estado correspondientes a las coordenadas cartesianas *X Y Z* lo que ha permitido implementar los métodos de control articular y en posición de dicho robot que a continuación se definen:

- **Mover Articulación** q_i a la posición articular θ . Este método permite mover independientemente cualquier articulación del robot manipulador. Tiene como parámetros de entrada el número de la articulación, la velocidad de giro y la posición articular en grados a la que se quiere mover. No tiene ninguna variable de salida.
- **Ir a la posición cartesiana *X Y Z*.** Ir a la posición cartesiana *X Y Z*. Este método coloca el extremo del robot en una determinada posición cartesiana. Para ello se resuelve la cinemática inversa. Como variables de entrada tiene la posición cartesiana (en versiones posteriores se planea también definir la velocidad lineal o articular con la que se quiere mover el robot). No tiene parámetros de salida.
- **Existe solución** Cinemática Inversa a la posición cartesiana *X Y Z*. El sistema calcula el problema cinemático inverso a una posición y responde si tiene solución. Como variable de entrada tiene la posición cartesiana. Como variable de salida devuelve un valor booleano que expresa si existe solución o no.
- **Abrir/cerrar pinza.** Este método abre o cierra la pinza que tiene el robot manipulador en el extremo. Tiene una variable booleana de entrada que expresa la apertura o cierre de la pinza. No tiene parámetros de salida.

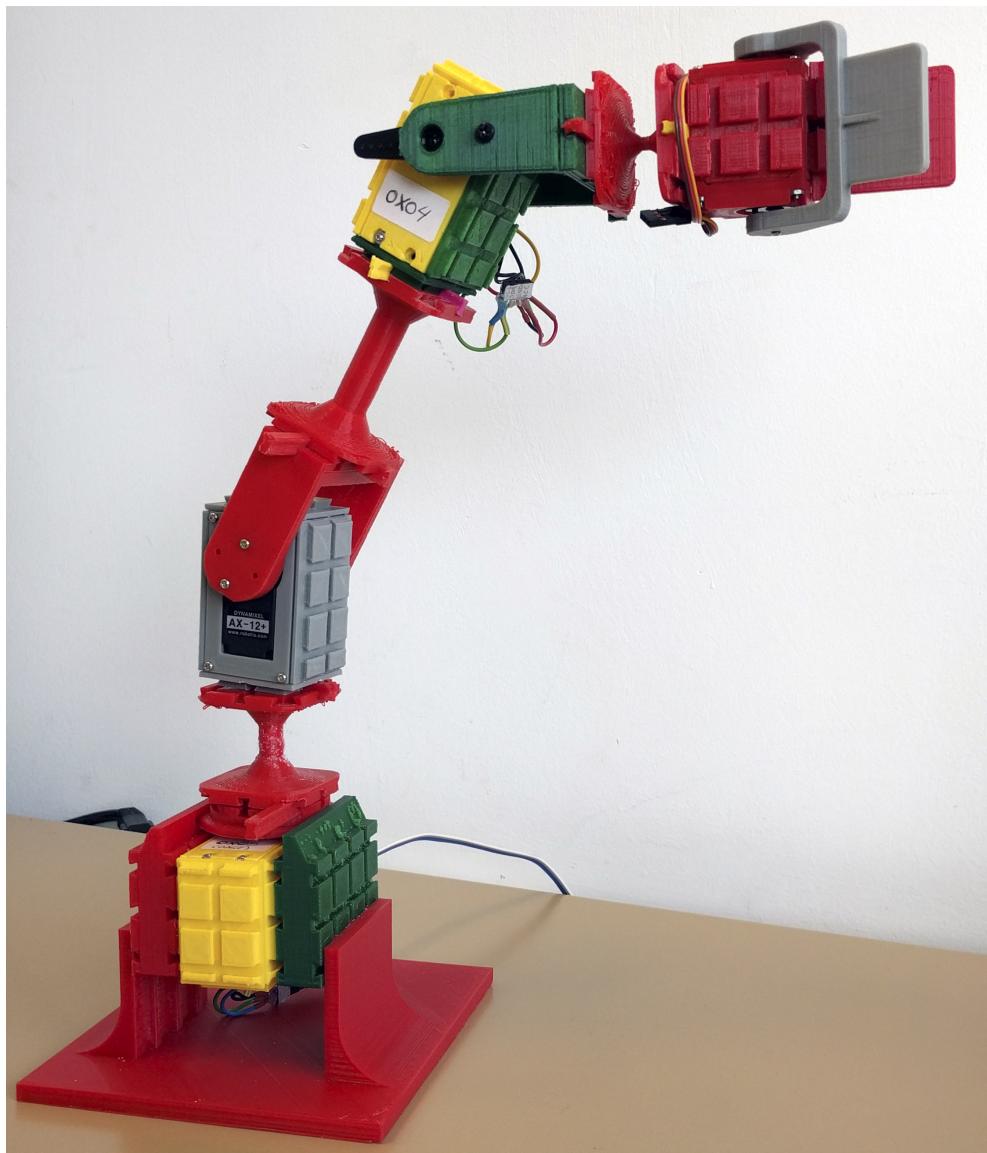


Figura 2: Robot obtenido a partir de la configuración tipo de robot manipulador

2.3. Interfaz de robots ápodos

El tipo de robots ápodos incluidos en BOTBLOQ asemejan a una serpiente. Están compuestos por 6 o más módulos activos de rotación no continua (módulos de posición) dispuestos a 90° entre sí. Esta configuración permite realizar la mayoría de los movimientos complejos que una serpiente real puede desarrollar. La

velocidad de los movimientos del robot dependerán de muchos factores (condiciones del suelo, carga, etc) por lo que todos los parámetros de entrada que se refieran a velocidades serán parámetros aproximados. Un ejemplo de un robot construido con nuestra plataforma modular a partir de esta configuración tipo puede verse en la Figura 3. A continuación se define la interfaz de los métodos que generan dichos movimientos:

- **Reptar hacia delante/atrás.** Este método hace moverse a la serpiente hacia delante o hacia atrás. El método genera un movimiento tipo gusano aunque a bajo nivel en el controlador se puede elegir un método de serpenteo. Como variable de entrada tiene la dirección de movimiento (adelante o atrás) y la velocidad en m/s a la que se desea mover la serpiente. No tiene variables de salida.
- **Reptar hacia derecha/izquierda.** Este método hace moverse a la serpiente en un movimiento curvilíneo hacia la izquierda o hacia la derecha. Como parámetro de entrada tiene la dirección de movimiento (izquierda o derecha) y la velocidad. No tiene variables de salida.
- **Rodar izquierda/derecha.** Este movimiento hace rotar la serpiente sobre su eje longitudinal hacia la izquierda o hacia la derecha. Como parámetro de entrada tiene la dirección de rotación. No tiene variables de salida.
- **Trasladarse lateralmente izquierda/derecha.** Este método genera un movimiento lineal hacia la derecha o hacia la izquierda. Como parámetro de entrada tiene la dirección del movimiento y la velocidad. No tiene variables de salida.
- **Parar.** Detiene cualquier movimiento de la serpiente.

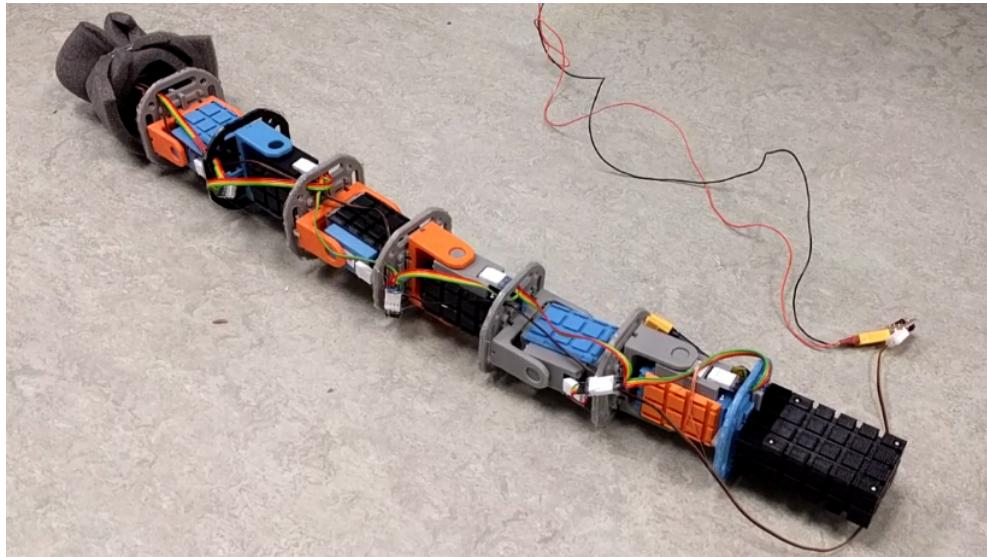


Figura 3: Robot obtenido a partir de la configuración tipo de robot serpiente

2.4. Interfaz de robots bípedos

La configuración de robot bípedo disponible para botbloq se asemeja a un robot humanoide y tiene 6 módulos activos de rotación no continua (módulos de posición) de alto par para cada pierna, 3 módulos activos de posición para cada brazo y un módulo activo de posición para la cabeza, lo que hace un total de 19 grados de libertad. Un ejemplo de un robot construido con nuestra plataforma modular a partir de esta configuración tipo puede verse en la Figura 4. Al igual que con los robots ápidos, la velocidad de los movimientos del robot dependerán de muchos factores (condiciones del suelo, carga, etc) por lo que todos los parámetros de entrada que se refieran a velocidades serán aproximados. A continuación se describe la interfaz de los métodos de movimiento generados hasta la fecha:

- **Caminar hacia delante/atrás.** Este método activa la caminada del robot hacia delante o atrás. Esta caminada corresponde a unos movimientos predefinidos cuya frecuencia puede aumentar o disminuir para cambiar la velocidad lineal del robot. Por lo tanto, la interfaz de este método tiene como parámetros de entrada la dirección y la velocidad (que será una aproximación) de movimiento. No tiene parámetros de salida.
- **Moverse lateralmente derecha/izquierda.** Este movimiento corresponde

a un movimiento de piernas (apertura y cierre desfasado para cada pierna) que permite un movimiento lineal lateral hacia la izquierda o hacia la derecha. Por lo tanto la interfaz de este método tendrá como variables de entrada la dirección del desplazamiento (izquierda o derecha) y la velocidad (que será una aproximación) de dicho desplazamiento

- **Rotar izquierda/derecha.** Este movimiento define giros sobre una pierna lo que permite rotar al robot hacia la izquierda o hacia la derecha. Por lo tanto los parámetros de entrada serán dirección de giro y velocidad de rotación (aproximada)
- **Agacharse.** Este método permite al robot situarse de manera estable en *cucillas*. La interfaz del método no tiene parámetros de entrada ni de salida.
- **Subir escaleras.** Este es un método avanzado en el que se realiza una secuencia de movimientos predeterminados que permite subir al robot escalones de pequeño tamaño. En futuras versiones se incluirá realimentación procedente de sensores por lo que el método cambiará sustancialmente. Actualmente este método no tiene parámetros de entrada o salida.
- **Parar.** Detiene cualquier movimiento en ejecución. No tiene parámetros de entrada o salida.

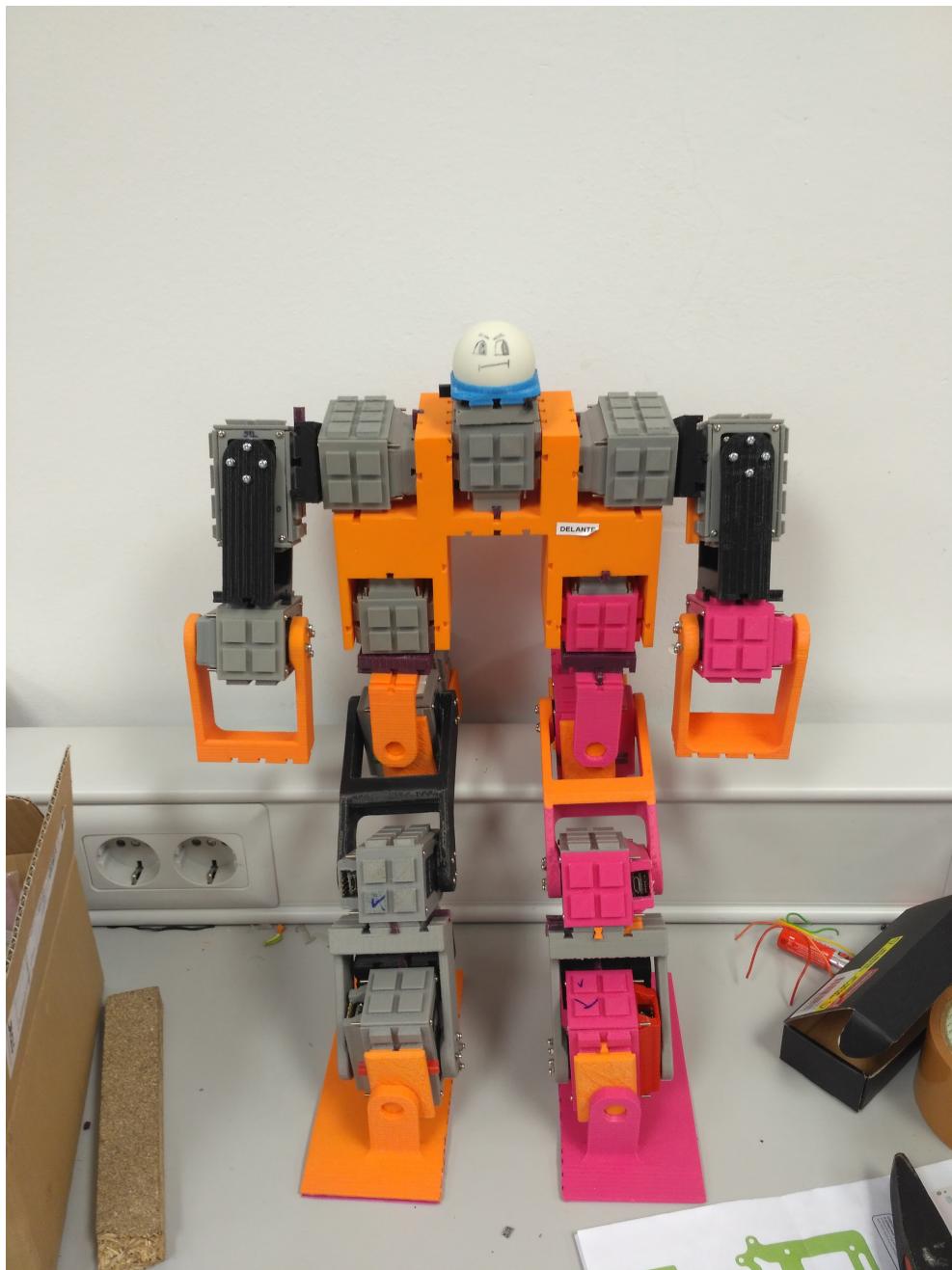


Figura 4: Robot obtenido a partir de la configuración tipo de robot humanoide

2.5. Interfaz de robots hexápodos

La configuración de robot hexápodo disponible para su programación en botbloq está compuesta por 3 módulos activos de posición, por pata, lo que hace un total 18 grados de libertad. Un ejemplo de un robot diseñado con nuestra plataforma modular a partir de esta configuración tipo puede verse en la Figura 5 Los métodos que se definen para los robots de este tipo son los siguientes:

- **Avanzar/retroceder.** Permite avanzar o retroceder en movimientos lineales. Los parámetros de entrada serán su la dirección y la velocidad del movimiento. Dicha velocidad dependerá del método implementado para el movimiento y de las condiciones de rozamiento del suelo por lo que será aproximada. No tiene variables de salida.
- **Girar izquierda/derecha.** Este método permite al robot girar sobre su eje central hacia la izquierda o hacia la derecha. Este giro también dependerá del método y de las condiciones de la superficie por lo que la velocidad de nuevo será aproximada. No tiene variables de salida.
- **Moverse lateralmente izquierda/derecha.** Movimientos lineales hacia la izquierda o derecha. Parámetros de entrada dirección y velocidad aproximada. No tiene variables de salida.
- **Parar.** Detiene cualquier movimiento



Figura 5: Robot obtenido a partir de la configuración tipo de robot hexápodo

3. Programación en BOTBLOQ de un robot modular tipo rover

Se han creado dos servidores para la Intel Edison que implementan los métodos que pueden ser llamados desde cualquier código programado en Bitbloq para el manejo del robot. El primer servidor es un controlador básico que incluye las funcionalidades ofrecidas a través de la interfaz descrita en la sección 2:

- Avanzar/retroceder
- Girar derecha/derecha
- Leer sensores infrarrojos izquierdo/derecho
- Parar

El segundo controlador está implementado en ROS y permite utilizar toda la potencia de ROS para usuarios más avanzados.

A continuación se describe la versión del servidor sin ROS. La versión en ROS será presentada en el apartado 3.3.

3.1. Servidor en la Intel Edison (no ROS)

El siguiente código implementa, siguiendo la interfaz detallada con anterioridad, la parte de código que se ejecuta en la Intel Edison y actúa a modo de servidor de funcionalidades del robot:

```
1 #!/usr/bin/env python
2
3
4 import sys
5 sys.path.append('/root/mraa/build/src/python')
6 import mraa
7 import time
8 import ctypes
9
10
11 class BotbloqVehicle():
12
```

```
13     def __init__(self):
14         self.x = mraa.I2c(0)
15
16     def moveWheel(self, address, speed):
17         self.x.address(address)
18         self.x.writeByte(ctypes.c_uint8(182).value)
19         time.sleep(0.001)
20         self.x.writeByte(ctypes.c_uint8(int(speed)).value)
21         time.sleep(0.001)
22
23     def move(self, delay, speed, direction):
24         velocidad_avance = int(90.0 + float(velocidad) * 90 / 100)
25         velocidad_retroceso = int(90.0 - float(velocidad) * 90 /
26                                     100)
27
28         if (direction == "FORWARD"):
29             moveWheel(0x04, velocidad_avance)
30             moveWheel(0x06, velocidad_avance)
31             moveWheel(0x03, velocidad_retroceso)
32             moveWheel(0x05, velocidad_retroceso)
33         elif (direction == "BACKWARD"):
34             moveWheel(0x04, velocidad_retroceso)
35             moveWheel(0x06, velocidad_retroceso)
36             moveWheel(0x03, velocidad_avance)
37             moveWheel(0x05, velocidad_avance)
38         elif (direction == "TURN_LEFT"):
39             moveWheel(0x04, velocidad_retroceso)
40             moveWheel(0x06, velocidad_retroceso)
41             moveWheel(0x03, velocidad_retroceso)
42             moveWheel(0x05, velocidad_retroceso)
43         elif (direction == "TURN_RIGHT"):
44             moveWheel(0x04, velocidad_avance)
45             moveWheel(0x06, velocidad_avance)
46             moveWheel(0x03, velocidad_avance)
47             moveWheel(0x05, velocidad_avance)
48             time.sleep(delay)
49
50     def wait(self, delay):
51         moveWheel(0x04, 90)
52         moveWheel(0x06, 90)
53         moveWheel(0x03, 90)
```

```
53         moveWheel(0x05, 90)
54
55         time.sleep(delay)
56
57     def stop(self):
58         moveWheel(0x04, 90)
59         moveWheel(0x06, 90)
60         moveWheel(0x03, 90)
61         moveWheel(0x05, 90)
62
63         exit()
64
65     def readIRSensor(self, side):
66         if side == "LEFT":
67             address = 0x32
68         else:
69             address = 0x33
70         self.x.address(address)
71         return self.x.readByte()
```

3.2. Adaptación de Bitbloq para la programación del robot móvil

Bitbloq ha sido adaptado de modo que se han creado bloques que generan código en python que, a través de la interfaz descrita, se conecta al servidor de la Intel Edison.

Primero se ha añadido el robot a la lista de robots disponibles en la pestaña hardware tal como aparece en la siguiente imagen:

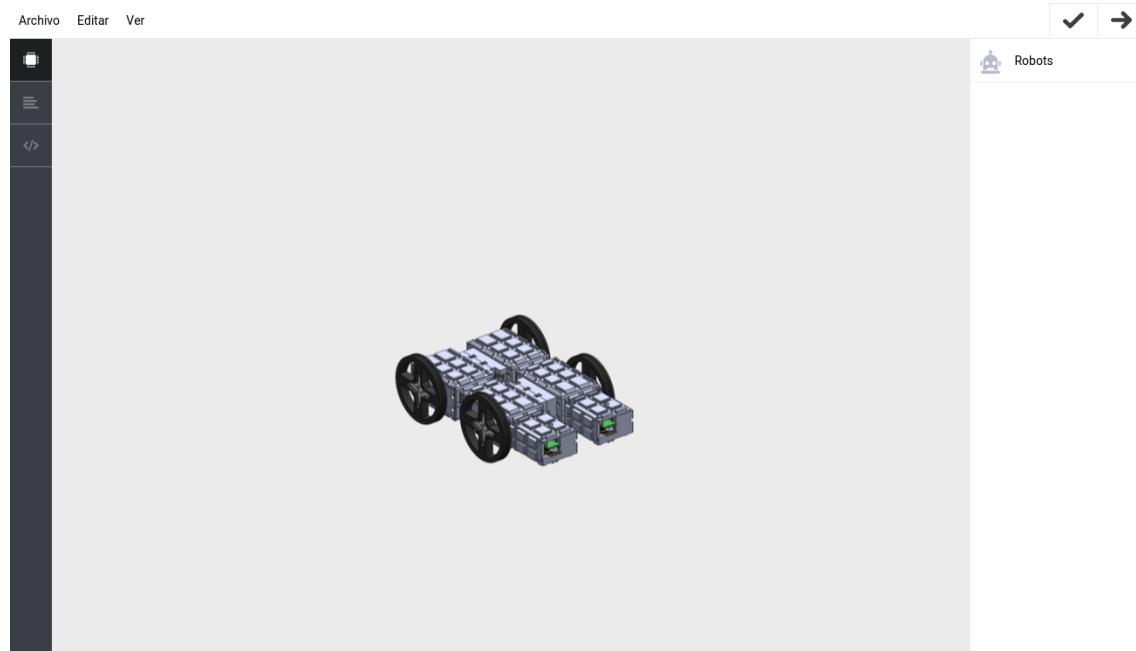


Figura 6: Pestaña Hardware para el robot modular móvil

En la pestaña bloques se han creado todos los bloques que implementan los métodos de funcionamiento descritos por la interfaz del robot. La siguiente figura muestra una captura de pantalla de dichos bloques.



Figura 7: Pestaña bloques del robot modular móvil

Estos bloques pueden ser utilizados dentro de Bitbloq al igual que el resto de bloques disponibles, pudiendo crear programas completos que definan el movimiento del robot según las necesidades del usuario. La siguiente imagen muestra un ejemplo de programa creado para que el robot sea capaz de seguir líneas:

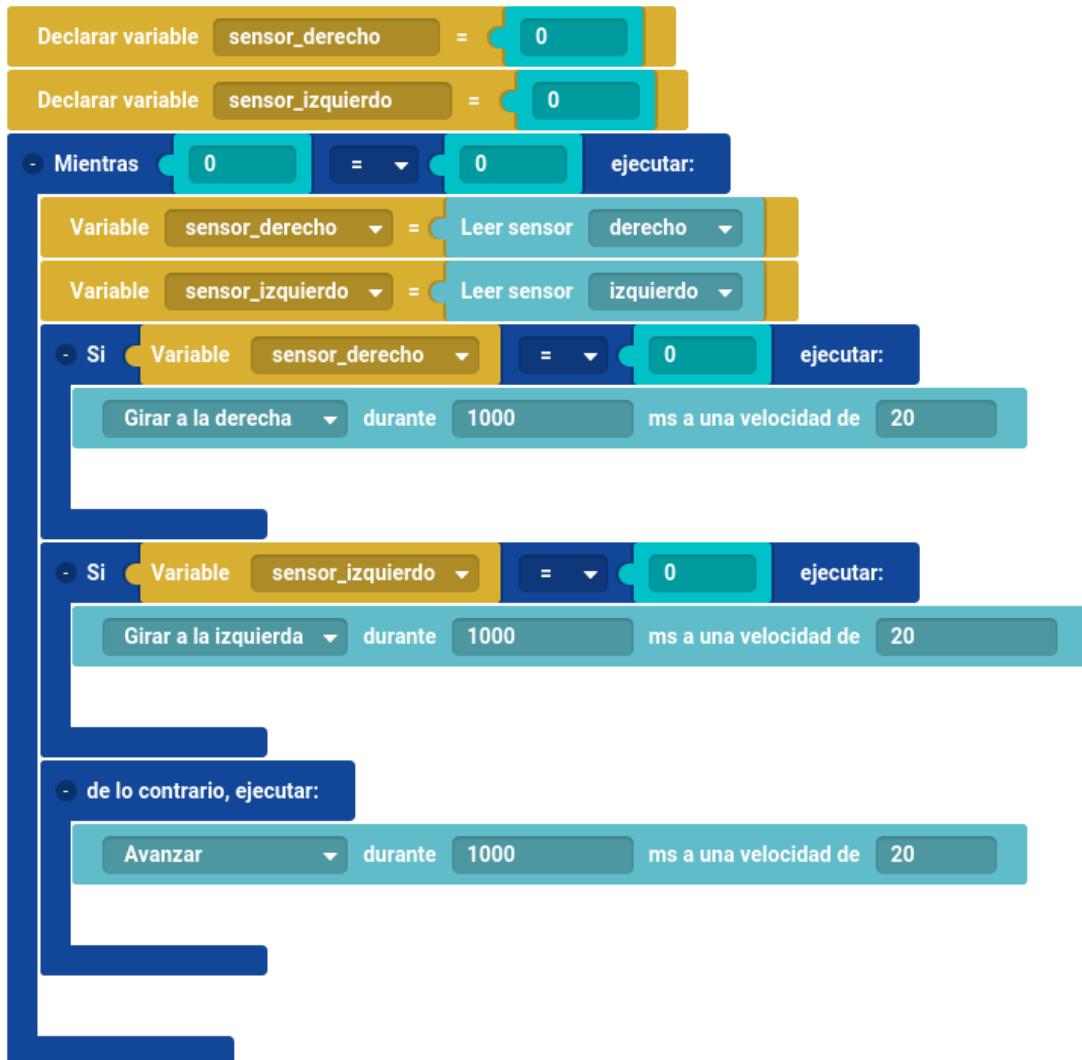


Figura 8: Código en Bitbloq

Estos bloques generan un código en python que tal como se expuso en la sección introductoria es enviado por *ssh* a la Intel Edison del robot. Una vez en la Intel edison el programa se ejecuta utilizando, por medio de la interfaz descrita, el código del controlador del robot que actúa por medio del puerto I2C sobre los módulos activos de las ruedas y los sensores. La siguiente imagen muestra el código python generado con el código de bloques de Bitbloq anterior.

```
1 # coding=utf-8
2
```

```
3 import BotbloqVehicle
4
5
6 vehicle = BotbloqVehicle.BotbloqVehicle()
7
8
9 sensor_derecho = 0
10 sensor_izquierdo = 0
11 while (0 == 0):
12     sensor_derecho = vehicle.readIRSensor("RIGHT")
13     sensor_izquierdo = vehicle.readIRSensor("LEFT")
14     if (sensor_derecho == 0):
15         vehicle.move("1000", "20", "TURN_RIGHT")
16
17     if (sensor_izquierdo == 0):
18         vehicle.move("1000", "20", "TURN_LEFT")
19
20     else:
21         vehicle.move("1000", "20", "FORWARD")
```

3.3. Versión en ROS

Se ha desarrollado una versión de servidor de movimientos basada en ROS en la Intel Edison para ser utilizada por usuarios de BOTBLOQ avanzados. En esta versión el código que genera botbloq no está basado en el uso de cabeceras de funciones previamente predefinidas, sino que hace llamadas mediante tópicos a un nodo servidor que se está ejecutando continuamente en la Intel. Es decir, cuando un usuario genera un código en Bitbloq este código se sube por medio de *ssh* a la Intel y se ejecuta automáticamente como un nodo más que se subscribe al nodo de control. La estructura de los nodos programados para controlar el robot móvil se muestra en la siguiente figura:

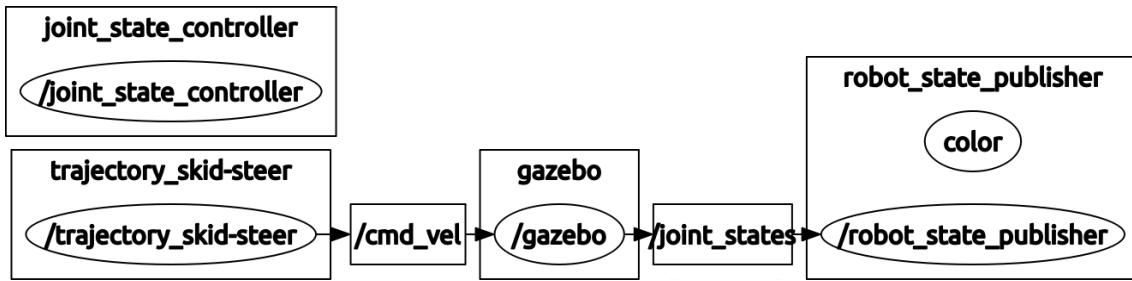


Figura 9: Código en Bitbloq

El código del servidor en la Intel Edison al que se conecta cualquier programa realizado en BOTBLOQ es el siguiente:

```

1 import rospy
2 from std_msgs.msg import String
3 from sensor_msgs.msg import JointState
4 import mraa
5 sys.path.append('/root/mraa/build/src/python')
6 # import mraa
7 import math
8 import ctypes
9
10 x = mraa.I2c(0)
11 conversion_vel = 90.0/6.25
12
13 def callback(data):
14     # print data.velocity
15     mi_lista= [a * b for a, b in zip(data.velocity, [x+
16         conversion_vel for x in [0]*len(data.velocity)])]
17     mi_lista = [round(x+90) for x in mi_lista]
18
19     # RF
20     x.address(0x05)
21     x.writeByte(ctypes.c_uint8(183).value)
22     rospy.sleep(rospy.Duration(0, 5000000))
23     x.writeByte(ctypes.c_uint8(mi_lista[2]).value)
24     rospy.sleep(rospy.Duration(0, 5000000))
25
26     # RR
27     x.address(0x04)
28     x.writeByte(ctypes.c_uint8(183).value)
  
```

```
28     rospy.sleep(rospy.Duration(0, 5000000))
29     x.writeByte(ctypes.c_uint8(mi_lista[4]).value)
30     rospy.sleep(rospy.Duration(0, 5000000))

31
32     # LF
33     x.address(0x07)
34     x.writeByte(ctypes.c_uint8(183).value)
35     rospy.sleep(rospy.Duration(0, 5000000))
36     x.writeByte(ctypes.c_uint8(mi_lista[1]).value)
37     rospy.sleep(rospy.Duration(0, 5000000))

38
39     # LR
40     x.address(0x08)
41     x.writeByte(ctypes.c_uint8(183).value)
42     rospy.sleep(rospy.Duration(0, 5000000))
43     x.writeByte(ctypes.c_uint8(mi_lista[3]).value)
44     rospy.sleep(rospy.Duration(0, 5000000))

45
46
47
48 def listener():
49
50     rospy.init_node('listener', anonymous=True)
51
52     rospy.Subscriber('/joint_states', JointState, callback)
53
54     rospy.spin()
55
56 if __name__ == '__main__':
57     listener()
```

4. Programación en BOTBLOQ de un robot modular tipo manipulador

Al igual que con el robot móvil, se han creado dos servidores para la Intel Edison que implementan los métodos que pueden ser llamados desde cualquier código programado desde Bitbloq para el manejo de este robot. El primer servidor es un controlador básico que incluye las siguientes funcionalidades ofrecidas a través de

la interfaz descrita en la sección 2:

- Mover Articulación i n grados
- Ir a la posición cartesiana (X Y Z)
- Existe solución Cinematica Inversa (X Y Z)
- Abrir/cerrar pinza

4.1. Servidor en la Intel Edison (no ROS)

El siguiente código implementa, siguiendo la interfaz detallada con anterioridad, la parte de código que se ejecuta en la Intel Edison y actúa a modo de servidor de funcionalidades del robot:

```
1 #!/usr/bin/env python
2
3
4 import PyKDL as kdl
5 import time
6 import math
7 import ctypes
8 import sys
9 sys.path.append('/root/mraa/build/src/python')
10 import mraa
11
12
13 class BotbloqManipulator():
14     def __init__(self):
15         self.chain = kdl.Chain()
16
17         joint0 = kdl.Joint(kdl.Joint.RotZ)
18         frame0 = kdl.Frame(kdl.Vector(0.0, 0.0, 0.0))
19         segment0 = kdl.Segment(joint0, frame0)
20         self.chain.addSegment(segment0)
21
22         joint1 = kdl.Joint(kdl.Joint.RotX)
23         frame1 = kdl.Frame(kdl.Vector(0, 0, 0.15))
24         segment1 = kdl.Segment(joint1, frame1)
25         self.chain.addSegment(segment1)
```

```

26
27     joint2 = kdl.Joint(kdl.Joint.RotX)
28     frame2 = kdl.Frame(kdl.Vector(0, 0, 0.15))
29     segment2 = kdl.Segment(joint2, frame2)
30     self.chain.addSegment(segment2)
31
32     links = self.chain.getNrOfJoints()
33     self.limit_joints = [[-math.pi / 2, math.pi / 2], [-math.
34                           pi / 2, math.pi / 2], [-math.pi / 2, math.pi / 2]]
35
36     self.solver_fk = kdl.ChainFkSolverPos_recursive(self.chain
37                                                     )
37     self.solver_vik = kdl.ChainIkSolverVel_pinv(self.chain)
38     self.solver_ik = kdl.ChainIkSolverPos_NR(self.chain, self.
39                                               solver_fk, self.solver_vik)
40
41
42     self.q = kdl.JntArray(links)
43     self.frame = kdl.Frame.Identity()
44
45     def moveJoint(self, joint, angle):
46         angle = math.radians(angle)
47         self.q[joint - 1] = angle
48
49         self.solver_fk.JntToCart(self.q, self.frame)
50         self.writeI2C(self.address, list(self.q))
51
52     def move(self, x, y, z):
53         self.frame.p[0] = x
54         self.frame.p[1] = y
55         self.frame.p[2] = z
56
57         self.solver_ik.CartToJnt(self.q, self.frame, self.q)
58         self.writeI2C(self.address, list(self.q))
59
60     def canMove(self, x, y, z):
61         self.frame.p[0] = x
62         self.frame.p[1] = y
63         self.frame.p[2] = z

```

```

64     ok = True
65
66     self.solver_ik.CartToJnt(self.q, self.frame, self.q)
67     angles = list(self.q)
68
69     if (list(self.q) == [0.0, 0.0, 0.0]) & ([x, y, z] != [0.0,
70         0.0, 0.3]):
71         ok = False
72
73     else:
74         for i in range(len(list(self.q))):
75             if (angles[i] < self.limit_joints[i][0]) | (angles
76                 [i] > self.limit_joints[i][1]):
77                 ok = False
78
79     return ok
80
81
82
83
84
85
86
87
88
89
90
91 def mapValue(value, arg_min, arg_max, sol_min, sol_max):
92     sol = value * ((float(sol_max) - float(sol_min)) / (float(
93         arg_max) - float(arg_min)))
94     return int(sol)

```

Cabe resaltar que este servidor utiliza la librería KDL para la soluciones cinemáticas necesarias como son cinemática directa, cinemática inversa o generación de trayectorias.

4.2. Adaptación de Bitbloq para la programación del robot manipulador

Bitbloq ha sido adaptado de modo que se han creado bloques que generan código en python que, a través de la interfaz descrita, se conecta al servidor de la Intel Edison.

Se ha añadido el robot a la lista de robots disponibles en la pestaña *hardware* tal como aparece en la siguiente imagen:



Figura 10: Pestaña Hardware

Siguiendo los métodos de control y la interfaz de los mismos se han generado los siguientes bloques:



Figura 11: Pestaña módulos

Utilizando esos bloques se han realizado varios programas de prueba. En siguiente programa representa un ejemplo sencillo de la potencia de Bitbloq para programar robots manipuladores

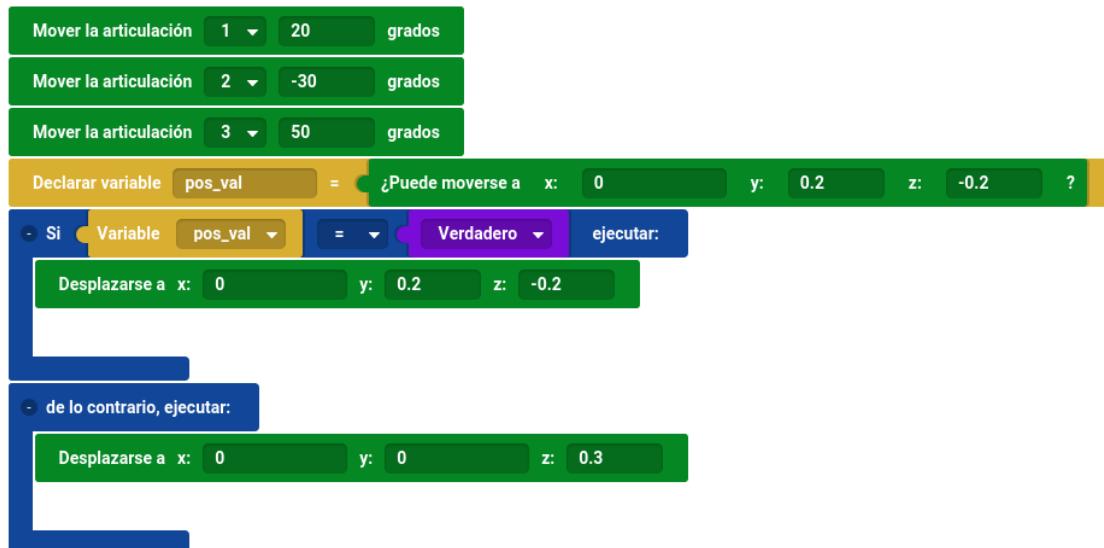


Figura 12: Código en Bitbloq

El código en python generado automáticamente a través de Bitbloq del programa anterior es el siguiente:

```
1 # coding=utf-8
2
```

```
3 import BotbloqManipulator
4
5
6 manipulator = BotbloqManipulator.BotbloqManipulator()
7
8
9 manipulator.moveJoint(1, 20)
10 manipulator.moveJoint(2, -30)
11 manipulator.moveJoint(3, 50)
12 pos_val = vehicle.canMove(0.0, 0.2, -0.2)
13 if(pos_val == True):
14     manipulator.move(0.0, 0.2, -0.2)
15
16 else:
17     manipulator.move(0.0, 0.0, 0.3)
```

4.3. Versión en ROS

Al igual que con el robot móvil, se ha desarrollado una versión para BOTBLOQ en ROS de manera que usuarios avanzados puedan utilizar este tipo de robots en aspectos de investigación y desarrollo. Se ha utilizado la herramienta *MoveIt!* de ROS para la implementación del controlador de robots manipuladores. Para poder realizar este control, primero se realiza de forma automática la configuración del robot usando el asistente de configuración que proporciona *MoveIt!*. Una vez configurado el robot, se han generado los paquetes que permiten visualizar el robot modelado. Los paquetes necesarios para poder generar las trayectorias, se generan desde BOTBLOQ en la IntelEdison usando el plugin de *MoveIt!* para Rviz y las librerías de *MoveIt!* para python. Las trayectorias que el usuario programa en BOTBLOQ son introducidas en un cliente en la Edison mediante *ssh*. Este cliente utiliza ActionLib para mandar las trayectorias al Nodo de ROS que actúa como servidor y que está ejecutándose continuamente en la Intel Edison. Este Nodo servidor es el encargado final de enviar por medio de I2C los comandos a los módulos de cada articulación del robot.

La estructura de los nodos programados para controlar el robot manipulador se muestra en la siguiente figura (como puede observarse es equivalente a la de un robot manipulador comercial ABB120 utilizado como robot de referencia para el

desarrollo del manipulador):

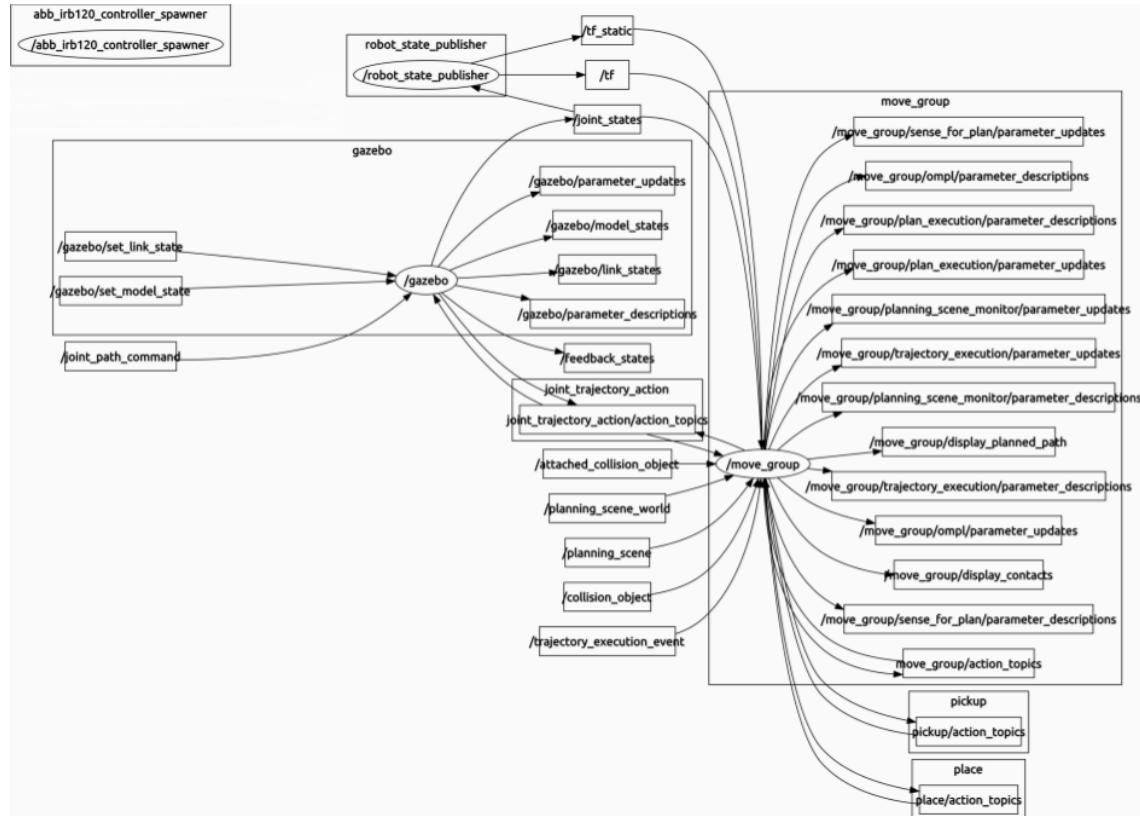


Figura 13: Rosgraph del servidor de control del manipulador para BOTBLOQ

El código del servidor en la Intel Edison al que se conecta cualquier programa realizado en BOTBLOQ es el siguiente:

```

1  #! /usr/bin/env python
2
3  import roslib
4  import rospy
5  import sys
6  sys.path.append('/root/mraa/build/src/python')
7  import mraa
8  import math
9  #from ctypes import c_ubyte
10 from control_msgs.msg import FollowJointTrajectoryAction ,
    FollowJointTrajectoryGoal , FollowJointTrajectoryActionFeedback ,
    FollowJointTrajectoryActionResult
  
```

```
11 from control_msgs.msg import FollowJointTrajectoryFeedback ,  
     FollowJointTrajectoryResult  
12 import actionlib  
13 from control_msgs.msg._FollowJointTrajectoryActionFeedback import  
     FollowJointTrajectoryActionFeedback  
14 from std_msgs.msg import Empty  
15 import ctypes  
16  
17 x = mraa.I2c(0)  
18  
19 class TrajectoryAction():  
20     # create messages that are used to publish feedback/result  
21     _feedback = FollowJointTrajectoryFeedback()  
22     _result   = FollowJointTrajectoryResult()  
23  
24     def __init__(self):  
25         self._action_name = 'Prueba Accion'  
26         self._as = actionlib.SimpleActionServer(''  
                joint_trajectory_action' , FollowJointTrajectoryAction ,  
                execute_cb=self.execute_cb , auto_start = False)  
27         self._as.start()  
28  
29     '''x.address(0x07)  
30     x.writeByte(183)  
31     rospy.sleep(rospy.Duration(0 , 5000000))  
32     x.writeByte(ctypes.c_uint8(5).value)'''  
33  
34  
35     def execute_cb(self , goal):  
36         self._result.SUCCESSFUL  
37         self._result.error_code = 10  
38         self._result.error_string = 'Leido y enviado'  
39         self._as.set_succeeded(self._result)  
40         # helper variables  
41         r = rospy.Rate(50)  
42         success = True  
43         print goal.trajectory.points  
44  
45         # start executing the action  
46         for i in xrange(0 , len(goal.trajectory.points)):  
47             # check that preempt has not been requested by the client
```

```
48     print goal.trajectory.points[i].positions
49     # goal.trajectory.points[i].velocities
50
51     x.address(0x05)
52     x.writeByte(ctypes.c_uint8(181).value)
53     rospy.sleep(rospy.Duration(0,5000000))
54     j_1v = goal.trajectory.points[i].velocities[0]
55     #x.writeByte(ctypes.c_uint8(int(round(j_1v))).value)
56     x.writeByte(ctypes.c_uint8(1).value)
57     rospy.sleep(rospy.Duration(0, 30000000))
58     x.writeByte(ctypes.c_uint8(182).value)
59     rospy.sleep(rospy.Duration(0, 30000000))
60     j_1 = goal.trajectory.points[i].positions[0]
61     j_1b = ctypes.c_uint8(int(round(j_1*180/math.pi)+90)).value
62     x.writeByte(j_1b)
63     rospy.sleep(rospy.Duration(0, 30000000))
64
65
66     x.address(0x07)
67     x.writeByte(ctypes.c_uint8(181).value)
68     rospy.sleep(rospy.Duration(0, 30000000))
69     j_2v = goal.trajectory.points[i].velocities[1]
70     #x.writeByte(ctypes.c_uint8(int(round(j_2v))).value)
71     x.writeByte(ctypes.c_uint8(1).value)
72     rospy.sleep(rospy.Duration(0, 30000000))
73     x.writeByte(ctypes.c_uint8(182).value)
74     rospy.sleep(rospy.Duration(0, 30000000))
75     j_2 = goal.trajectory.points[i].positions[1]
76     j_2b = ctypes.c_uint8(int(round(j_2*180/math.pi)+90)).value
77     x.writeByte(j_2b)
78
79     x.address(0x08)
80     x.writeByte(ctypes.c_uint8(181).value)
81     rospy.sleep(rospy.Duration(0, 30000000))
82     j_3v = goal.trajectory.points[i].velocities[2]
83     x.writeByte(ctypes.c_uint8(1).value)
84     rospy.sleep(rospy.Duration(0, 30000000))
85     #x.writeByte(ctypes.c_uint8(int(round(j_3v))).value)
86     x.writeByte(ctypes.c_uint8(182).value)
87     rospy.sleep(rospy.Duration(0, 30000000))
88     j_3 = goal.trajectory.points[i].positions[2]
```

```
89     j_3b = ctypes.c_uint8( int( round(j_3*180/math.pi)+90) ).value
90     x.writeByte(j_3b)
91     rospy.sleep(rospy.Duration(0, 30000000))
92
93     x.address(0x03)
94     x.writeByte(ctypes.c_uint8(181).value)
95     rospy.sleep(rospy.Duration(0, 30000000))
96     j_4v = goal.trajectory.points[i].velocities[3]
97     #x.writeByte(ctypes.c_uint8(int(round(j_4v))).value)
98     x.writeByte(ctypes.c_uint8(1).value)
99     rospy.sleep(rospy.Duration(0, 30000000))
100    x.writeByte(ctypes.c_uint8(182).value)
101    rospy.sleep(rospy.Duration(0, 30000000))
102    j_4 = goal.trajectory.points[i].positions[3]
103    j_4b = ctypes.c_ushort(int(round(j_4*180/math.pi)+90)).value
104    x.writeByte(j_4b)
105    rospy.sleep(rospy.Duration(0, 30000000))
106
107    x.address(0x04)
108    x.writeByte(ctypes.c_uint8(181).value)
109    rospy.sleep(rospy.Duration(0, 30000000))
110    j_5v = goal.trajectory.points[i].velocities[4]
111    #x.writeByte(ctypes.c_uint8(int(round(j_5v))).value)
112    x.writeByte(ctypes.c_uint8(1).value)
113    rospy.sleep(rospy.Duration(0, 30000000))
114    x.writeByte(ctypes.c_uint8(182).value)
115    rospy.sleep(rospy.Duration(0, 30000000))
116    j_5 = goal.trajectory.points[i].positions[4]
117    j_5b = ctypes.c_uint8(int(round(j_5*180/math.pi)+90)).value
118    x.writeByte(j_5b)
119    rospy.sleep(rospy.Duration(0, 30000000))
120
121    x.address(0x06)
122    x.writeByte(ctypes.c_uint8(181).value)
123    rospy.sleep(rospy.Duration(0, 30000000))
124    j_6v = goal.trajectory.points[i].velocities[5]
125    #x.writeByte(ctypes.c_uint8(int(round(j_6v))).value)
126    x.writeByte(ctypes.c_uint8(1).value)
127    rospy.sleep(rospy.Duration(0, 30000000))
128    x.writeByte(ctypes.c_uint8(182).value)
129    rospy.sleep(rospy.Duration(0, 30000000))
```

```
130     j_6 = goal.trajectory.points[i].positions[5]
131     j_6b = ctypes.c_uint8(int(round(j_6*180/math.pi)+90)).value
132     x.writeByte(j_6b)
133     rospy.sleep(rospy.Duration(0, 30000000))
134
135
136     if self._as.is_preempt_requested():
137         rospy.loginfo('%s: Preempted' % self._action_name)
138         #self._as.set_preempted()
139         #success = False
140         break
141
142     r.sleep()
143
144     '''if success:
145         #self._result.sequence = self._feedback.sequence
146         rospy.loginfo('%s: Succeeded' % self._action_name)
147         self._result.SUCCESSFUL
148         self._result.error_code = 10
149         self._result.error_string = 'Leido y enviado',
150         self._as.set_succeeded(self._result)'''
151
152 if __name__ == '__main__':
153     rospy.init_node('servidor')
154     TrajectoryAction()
155     rospy.spin()
```

5. Programación en BOTBLOQ de un robot modular tipo serpiente (solo ROS)

Los robots Serpiente o ápodos son robots con múltiples grados de libertad (se clasifican como robots redundantes). Esto implica que sean altamente modulares ya que todo su cuerpo está compuesto por módulos iguales con distinta orientación. En concreto, el robot utilizado en BOTBLOQ cuenta con seis servos intercalados formando 90°. Estos robots necesitan de unos controladores específicos de alta complejidad. Es por ello que se ha decidido realizar su integración con Bitbloq directamente con una única versión en ROS. Esto permite probar de manera efectiva

diferentes estrategias ya disponibles en el propio ROS y abstraerlas al programador en Bitbloq.

Al igual que con todos los robots se han diseñado los bloques de Bitbloq atendiendo a la definición del interfaz:

- Reptar hacia delante/atras
- Reptar hacia derecha/izquierda
- Rodar izquierda/derecha
- Trasladarse lateralmente izquierda/derecha
- Parar

5.1. Nodo de ROS para el control de los robots serpientes

Como se ha comentado anteriormente, estos robots al ser tan complejos solo es posible controlarlos mediante sistemas de control complejos. Es por esto que para su programación con BOTBLOQ se ha realizado directamente en un sistema basado en ROS. El principio de funcionamiento es similar a lo visto con los servidores ROS para el manipulador y el robot móvil: se ha creado un Nodo servidor en el que se publican los tópicos ROS de movimiento. Los nodos clientes generados por un programador en BOTBLOQ son subidos por *ssh* a la Intel Edison y estos se conectan por medio de Actionlib al controlador de la serpiente. Con las ordenes procedentes de los clientes el controlador ejecuta las acciones de control necesarias para la generación de trayectorias, basándose para ello en la generación de ondas senoidales desfasadas tanto para los módulos colocados en vertical como para los colocados en horizontal.

La siguiente figura muestra el esquema de Nodos en ROS creados para el Servidor de Control de la serpiente:

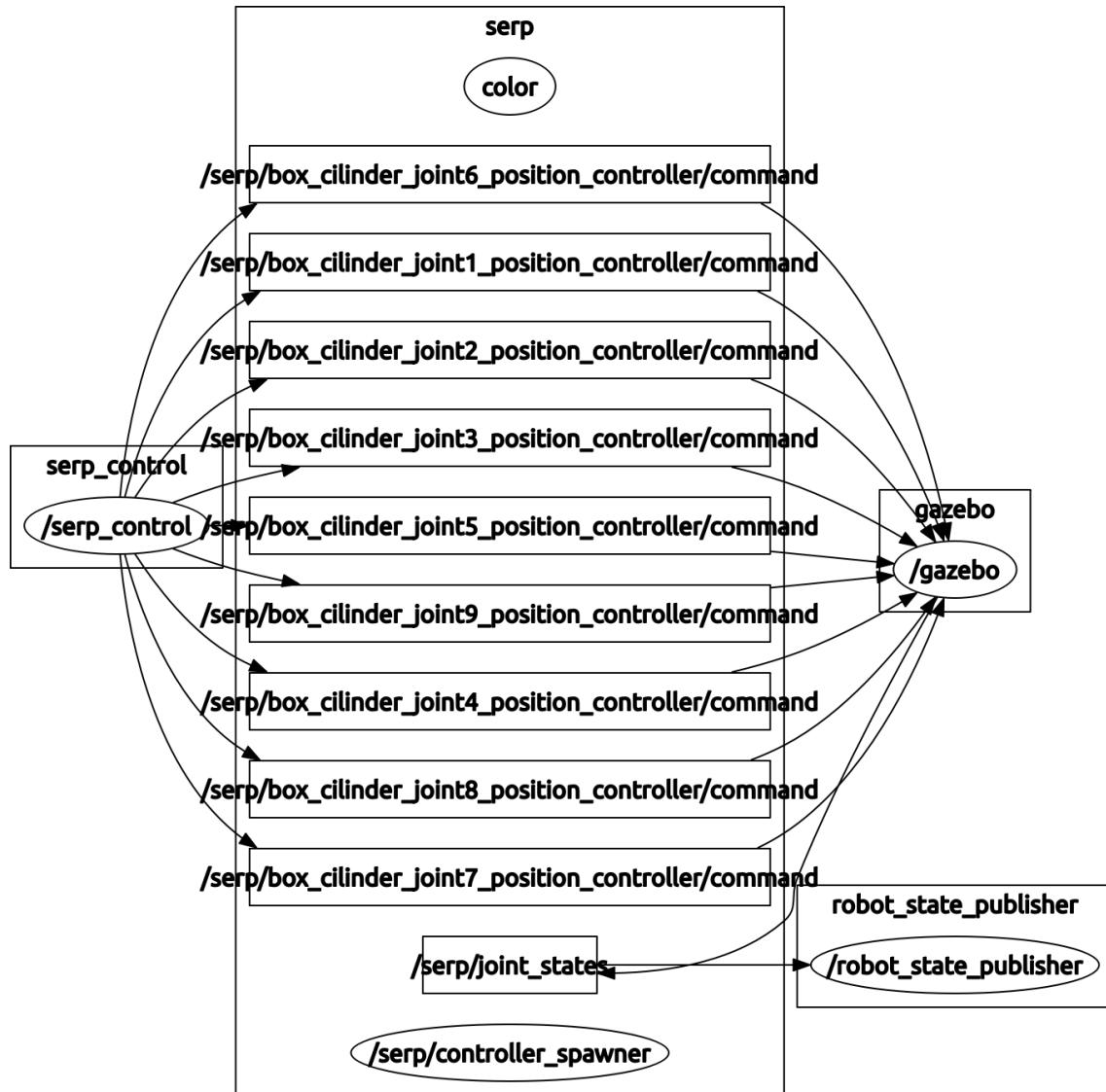


Figura 14: ROSGraph del servidor de control de la serpiente para BOTBLOQ

EL código del nodo servidor en la Intel Edison al que se conecta cualquier programa generado en botbloq se presenta a continuación. Como se puede observar, los mensajes recibidos por clientes externos son convertidos en las correspondientes directivas para la generación de movimientos y enviados por medio del bus I2C a los módulos del robot.

```

1#!/usr/bin/env python
2import rospy

```

```
 3 from std_msgs.msg import Float64
 4 import ctypes
 5 import sys
 6 import math
 7 sys.path.append('/root/mraa/build/src/python')
 8 # import mraa
 9 # x = mraa.I2c(0)
10
11
12
13 pos3= None
14 pos4= None
15 pos5= None
16 pos6= None
17 pos7= None
18 pos8= None
19
20
21 '''def enviar_i2c(servo ,dato):
22     x.address(servo)
23     x.writeByte(ctypes.c_uint8(181).value)
24     rospy.sleep(rospy.Duration(0, 30000000))
25     x.writeByte(ctypes.c_uint8(1).value)
26     rospy.sleep(rospy.Duration(0, 30000000))
27     x.writeByte(ctypes.c_uint8(182).value)
28     rospy.sleep(rospy.Duration(0, 30000000))
29     j_1b = ctypes.c_uint8(int(round(dato*180/math.pi)+90)).value
30     x.writeByte(j_1b)
31     rospy.sleep(rospy.Duration(0, 30000000))'''
32
33 def enviar():
34     global pos3
35     global pos4
36     global pos5
37     global pos6
38     global pos7
39     global pos8
40     # enviar_i2c(0x03, pos3)
41     # enviar_i2c(0x04, pos4)
42     # enviar_i2c(0x05, pos5)
43     # enviar_i2c(0x06, pos6)
```

```
44     # enviar_i2c(0x07, pos7)
45     # enviar_i2c(0x08, pos8)
46
47     print("Posicion1 {}".format(pos3))
48     print("Posicion2 {}".format(pos4))
49     print("Posicion3 {}".format(pos5))
50     print("Posicion4 {}".format(pos6))
51     print("Posicion5 {}".format(pos7))
52     print("Posicion6 {}".format(pos8))
53
54
55 def callback1(data):
56     global pos3
57     pos3 = data.data
58
59
60 def callback2(data):
61     global pos4
62     pos4 = data.data
63
64
65 def callback3(data):
66     global pos5
67     pos5 = data.data
68
69
70 def callback4(data):
71     global pos6
72     pos6 = data.data
73
74
75 def callback5(data):
76     global pos7
77     pos7 = data.data
78
79
80 def callback6(data):
81     global pos8
82     pos8 = data.data
83     enviar()
84
```

```
85
86 def listener():
87
88     rospy.init_node('listener', anonymous=True)
89
90     rospy.Subscriber("serp/box_cylinder_joint1_position_controller
91                     /command", Float64, callback1, queue_size=1)
92     rospy.Subscriber("serp/box_cylinder_joint2_position_controller
93                     /command", Float64, callback2, queue_size=1)
94     rospy.Subscriber("serp/box_cylinder_joint3_position_controller
95                     /command", Float64, callback3, queue_size=1)
96     rospy.Subscriber("serp/box_cylinder_joint4_position_controller
97                     /command", Float64, callback4, queue_size=1)
98     rospy.Subscriber("serp/box_cylinder_joint5_position_controller
99                     /command", Float64, callback5, queue_size=1)
100    rospy.Subscriber("serp/box_cylinder_joint6_position_controller
101                     /command", Float64, callback6, queue_size=1)
102
103 if __name__ == '__main__':
104     listener()
```

El código del controlador específico que genera, por medio de ondas, los diferentes tipos de movimientos presentados en la interfaz se ha incluido por su extensión en el anexo B

5.2. Implementación en Bitbloq de los bloques de control de la serpiente

Se ha incluido el robot serpiente en la lista de robots disponibles en Bitbloq tal como se muestra en la siguiente imagen:

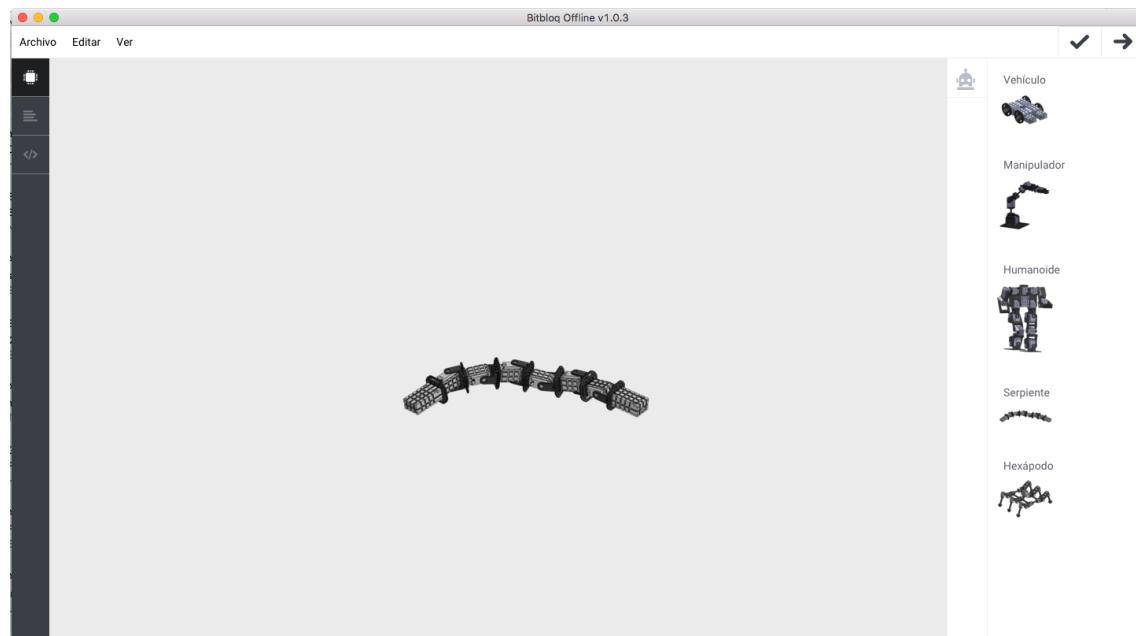


Figura 15: Pestaña Hardware robots serpiente

Siguiendo la interfaz descrita para la serpiente se han creado (hasta la fecha) los siguiente bloques en Bitbloq que permiten el control de los movimientos de las serpientes modulares:

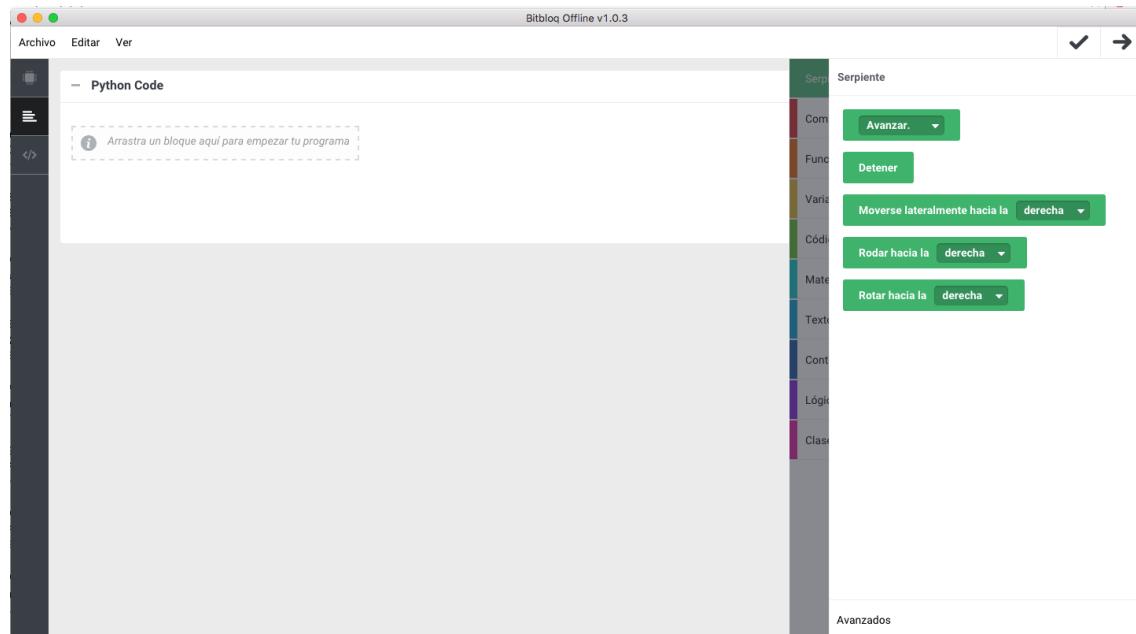


Figura 16: Pestaña bloques robots ápodos

El siguiente código muestra un ejemplo de utilización de dichos bloques para la generación de un comportamiento sencillo en la serpiente.

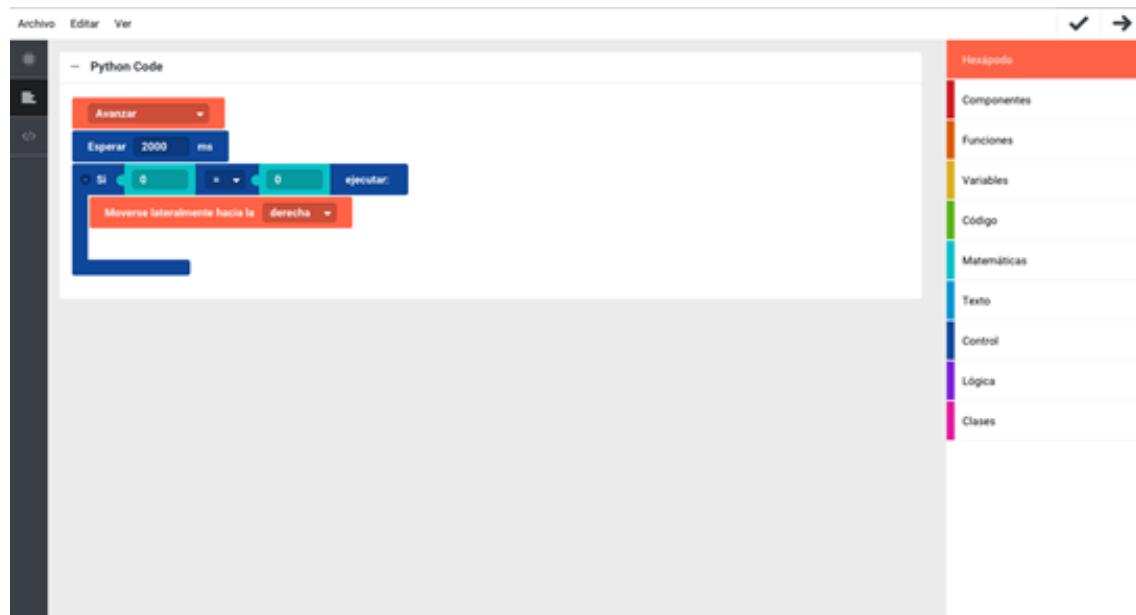


Figura 17: Código en Bitbloq

6. Programación en BOTBLOQ de un robot modular tipo humanoide (solo ROS)

Los robots humanoides, por su gran número de articulaciones y sus complejas cadenas cinemáticas, son los robots más difíciles de controlar en la actualidad. Por ello para este tipo de robots se ha diseñado un controlador en ROS que haga de interfaz entre el código generado en botbloq y los movimientos del robot. Al igual con los robots anteriores existe un Nodo Servidor ROS alojado en la Intel Edison al cual se conecta por medio de ActionLibs los clientes creados en botbloq (que son subido por medio de *ssh* a la Intel Edison).

Al igual que con el resto de robots, se ha utilizado la interfaz descrita con anterioridad para su integración con Bitbloq:

- Caminar hacia delante/atrás
- Moverse lateralmente derecha/izquierda
- Rotar izquierda/derecha
- Agacharse
- Subir escaleras
- Parar

6.1. Nodo de Ros para el control de los robots humanoides

Para realizar el nodo controlador en ROS primero ha sido necesario realizar el modelado del robot usando XACRO. La implementación de los controles cinemáticos se ha realizado por medio de la librería KDL. Ésta contiene solvers genéricos para cadenas cinemáticas que permiten resolver tanto la cinemática directa como la cinemática inversa.

Con el modelo del robot creado se generan las cadenas cinemáticas para cada una de las piernas. Usando estas cadenas cinemáticas se resuelve la cinemática inversa para generar una caminada estable. La caminada para el robot particular prediseñado en BOTBLOQ fue analizada y optimizada en una etapa previa por medio de la exportación de los datos por Rosbag y su tratamiento en Matlab.

En particular, para determinar la caminada se utilizan FFTs las cuales obtienen las componentes frecuenciales principales para cada articulación. Para ello en primer lugar se eliminaron las zonas de descanso donde la articulación se encuentra en reposo, también se eliminaron la componente continua que fue añadida más tarde durante la reconstrucción de la señal usando:

$$A = \sum_{i=0}^N a_i \sin(\omega_i t + \phi_i) \quad (1)$$

donde N es el número de armónicos usados para reconstruir las señales.

El resultado es un controlador que planifica caminadas estables. En resumen, si por ejemplo un usuario de BOTBLOQ utiliza el bloque avanzar (ver figura 20) se generará un código que se subirá a la Intel por *ssh* y que se conectará al por ActionLibs al controlador previo instalado en la Intel Edison. Este controlador enviará por I2C los comandos a los módulos de las articulaciones del robot humanoide. Debido al tamaño del mismo, el código de este controlador integrado en ROS está incluido en el Anexo C

A modo de explicación de dicho código, la siguiente figura muestra el esquema de Nodos en ROS creados para el Servidor de Control:

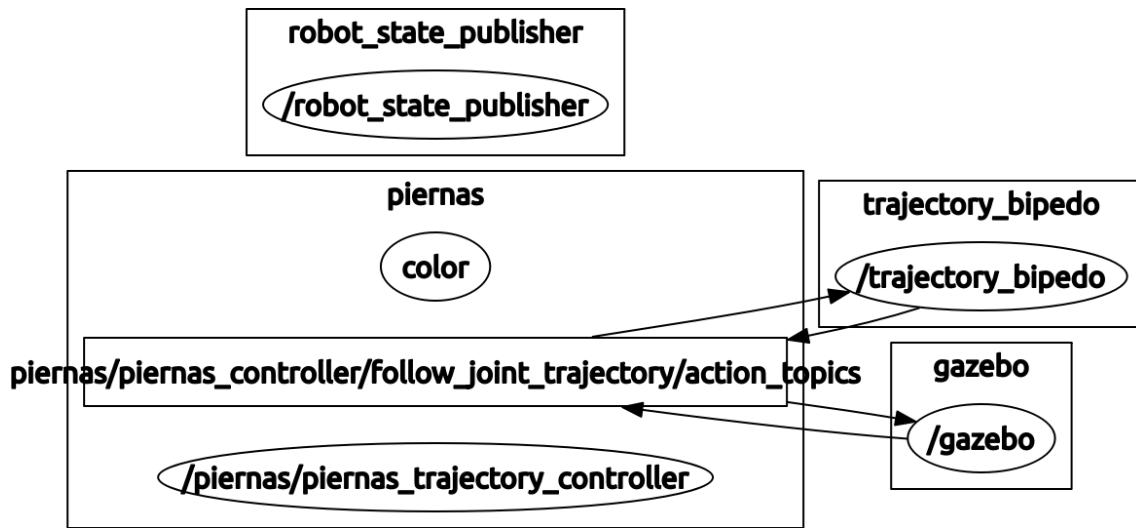


Figura 18: ROSGraph del servidor de control de los robots humanoides para BOTBLOQ

6.2. Implementacion en Bitbloq de los bloques de control del robot humanoide

La configuración del robot humanoide ha sido incluida en la pestaña hardware de robots disponibles tal como aparece en la siguiente imagen.

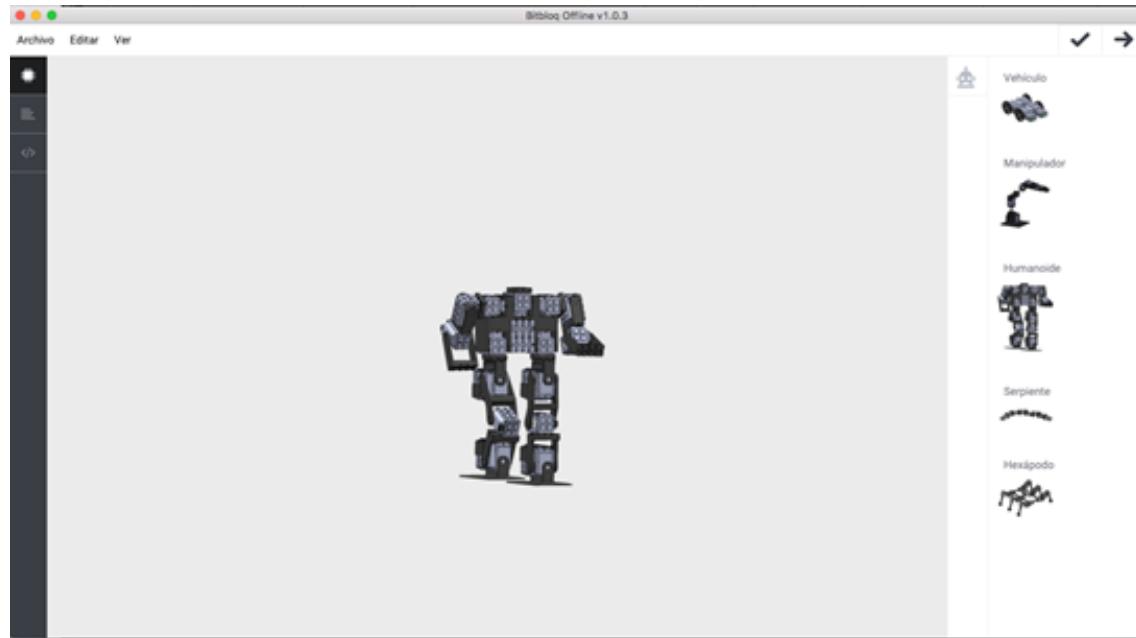


Figura 19: Pestaña Hardware robots humanoides

La siguiente figura muestra los bloques generados para el control de movimientos en Bitbloq del robot humanoide:

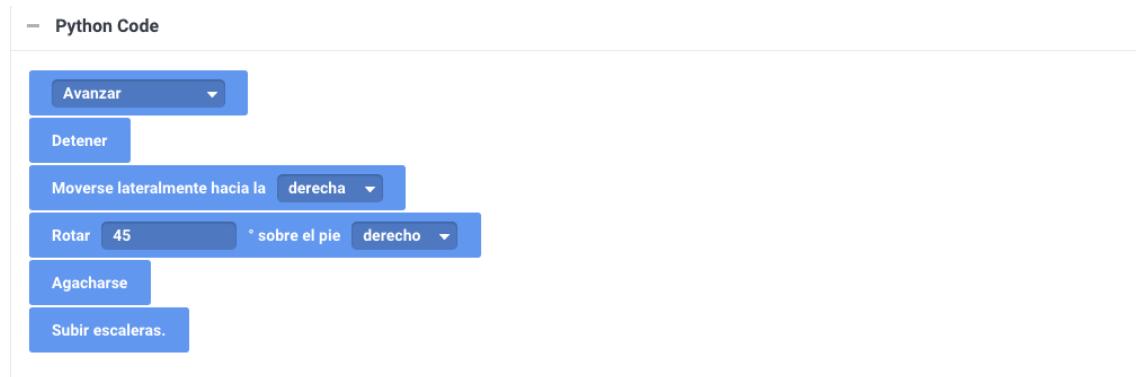


Figura 20: Pestaña módulos

El siguiente código muestra un ejemplo de un sencillo código de bloques que implementa una serie de movimientos para el robot humanoide.

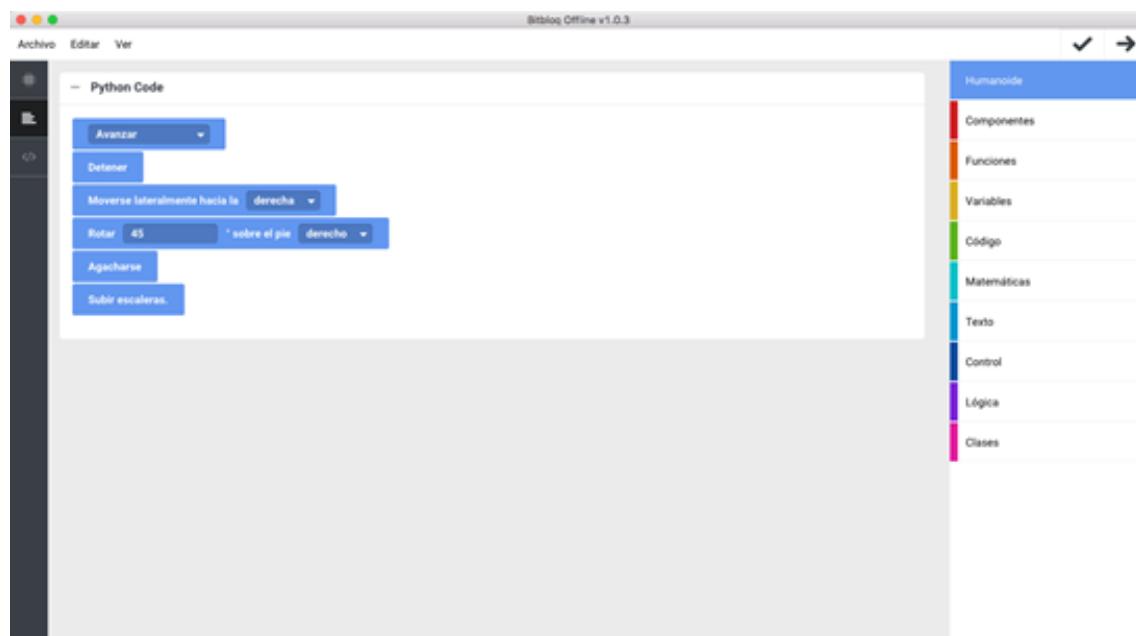


Figura 21: Código en Bitbloq

7. Programación en BOTBLOQ de un robot modular tipo hexápodo (solo ROS)

Los robots hexápodos son otro tipos de robot con una alta complejidad debido su número de patas. Dado que un robot puede ser estáticamente estable en tres o más patas, un robot hexápodo tiene una gran flexibilidad para moverse. Si las piernas se incapacitan, el robot puede aún ser capaz de caminar. Las estrategias de control varían en función del número de patas que se quiera tener en contacto con el suelo en cada instante. Para poder disponer de diferentes estrategias se ha optado por utilizar solamente ROS para la integración del servidor de control al que se conecten los programas generados en Bitbloq. El desarrollo de dicho controlador ha influido en la siguiente definición de la interfaz:

- Avanzar/retroceder

- Girar izquierda/derecha
- Moverse lateralmente izquierda/derecha
- Parar

Los diferentes métodos que definen la interfaz han sido implementados en ROS para el control de robot hexápedo. Debido al tamaño de los mismos, el código del controlador integrado en ROS no se incluye en esta sección pero si en el Anexo D

A modo de explicación de dicho código, la siguiente figura muestra el esquema de Nodos en ROS creados para el Servidor de Control:

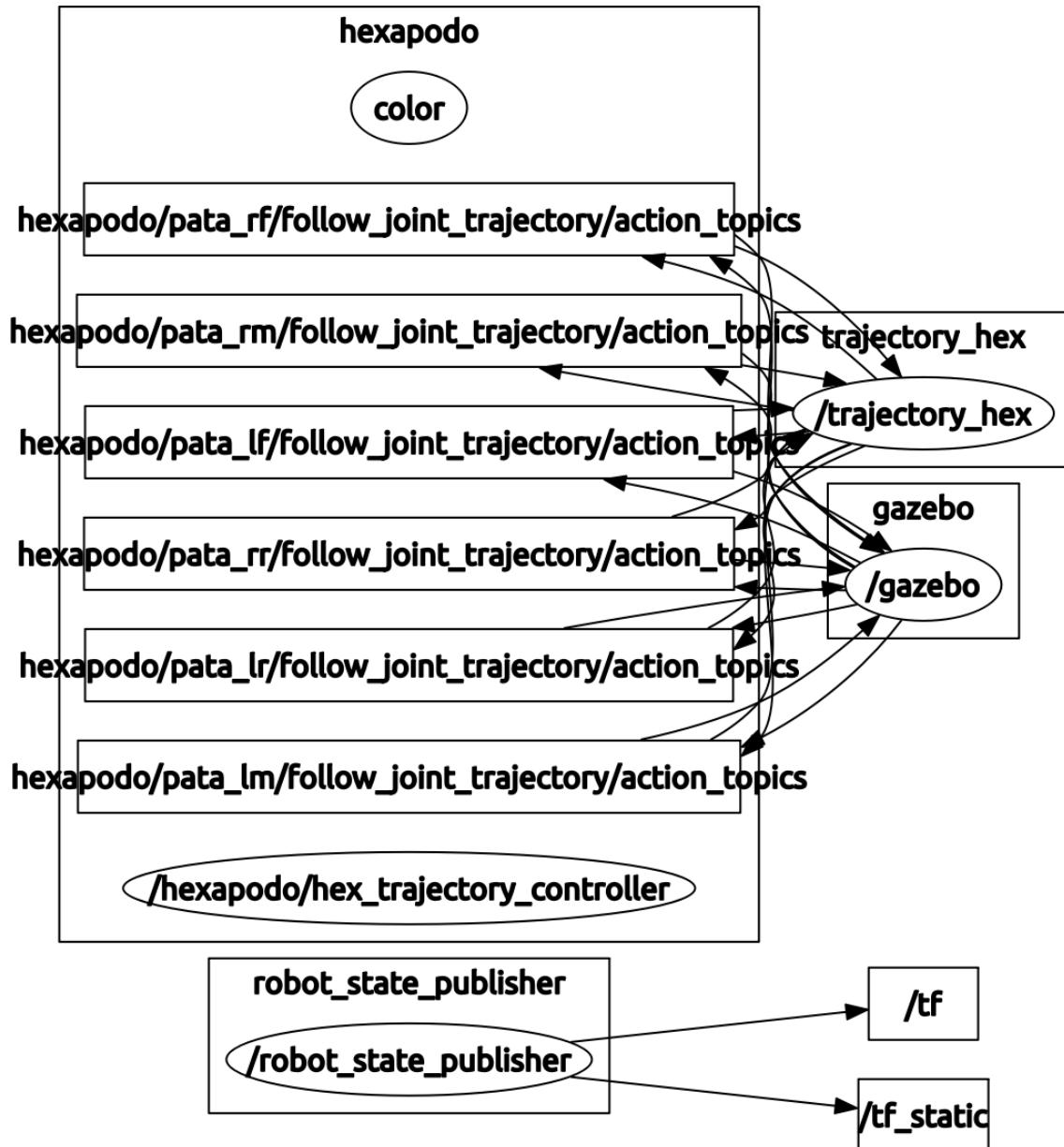


Figura 22: ROSGraph del servidor de control de los robots hexápodos para BOT-BLOQ

7.1. Implementacion en Bitbloq de los bloques de control del robot hexápodo

El robot hexápodo ha sido integrado con el resto de robots disponibles en la lista hardware de Bitbloq como aparece en la siguiente figura:

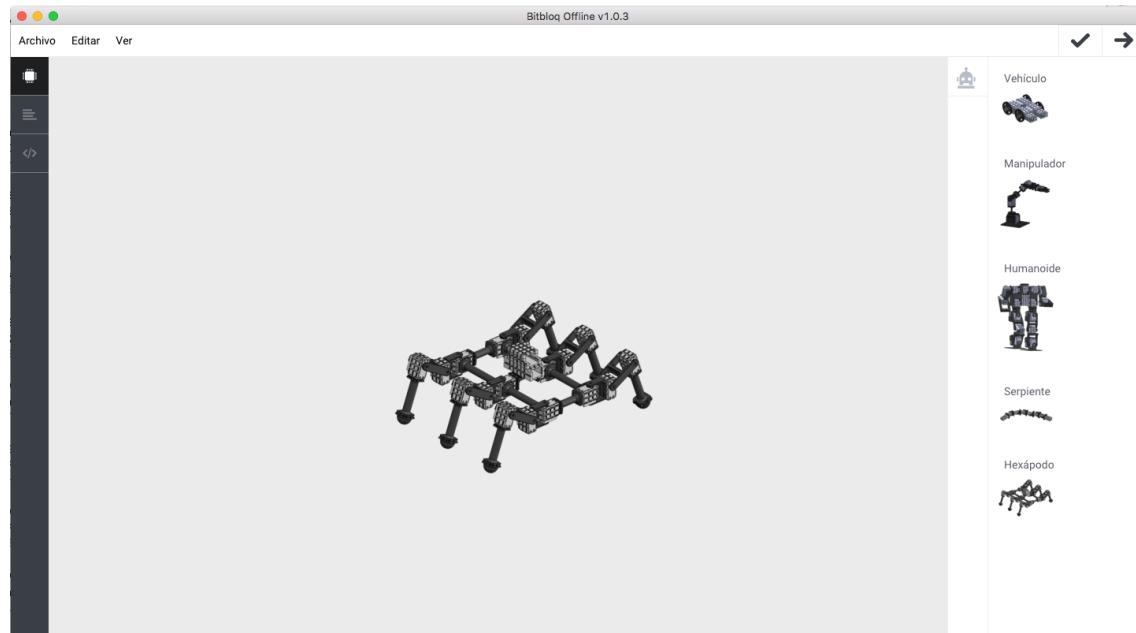


Figura 23: Pestaña Hardware

Los módulos de Bitbloq para este robot definidos a través de la interfaz se presentan en la siguiente imagen:

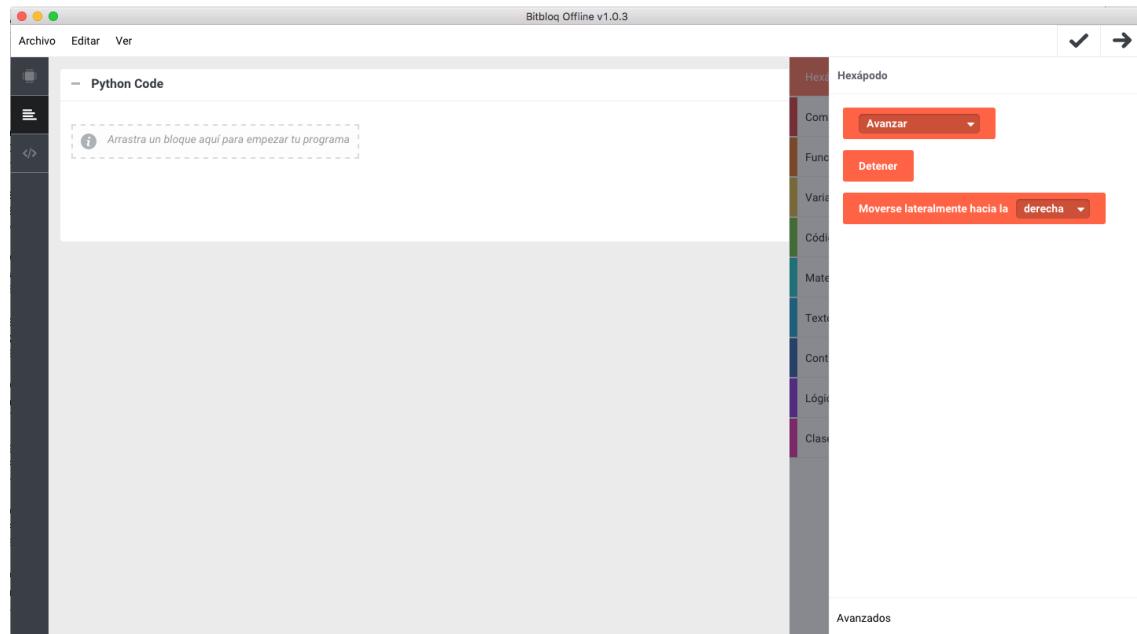


Figura 24: Pestaña módulos hexapodo

A continuación se muestra un programa de ejemplo para la programación de movimientos del robot hexápodo con los módulos creados:

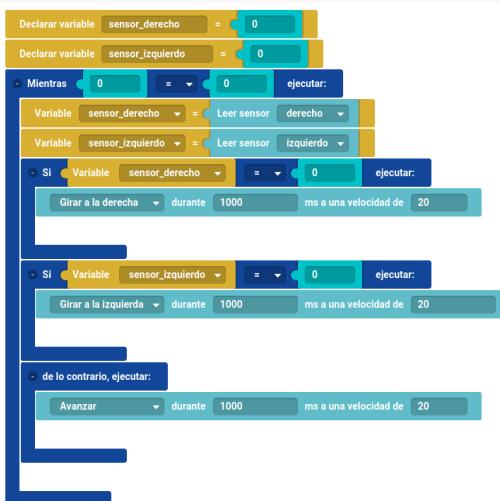


Figura 25: Código en Bitbloq

Al igual que con todos los robots presentados en este entregable, el código de

bloques programado en Bitbloq es traducido a python y subido por *ssh* a la Intel Edison, donde por medio de actionLibs de ROS se conectará al Nodo controlador del robot y efectuará el comportamiento definido por el usuario de Bitbloq.

A. Cliente I2C en los módulos activos

A.0.1. cliente I2c en modulos servo

```
1
2 #include <SoftRcPulseOut.h>
3 #include <TinyWireS.h>
4
5 SoftRcPulseOut myservo;
6
7 #define SERVO_PIN      3
8
9 #define NOW           1
10
11 #define DIR_I2C 0x03
12
13 int byte_rec = 0, pos_rec = 90, espera = 20, tipo = 0, vel = 1,
     pos_actual = 0;
14
15 void receiveEvent(uint8_t howMany)
16 {
17     byte_rec = TinyWireS.receive();
18     if (tipo == 1)
19     {
20         vel = byte_rec;
21         tipo = 0;
22     }
23     else if (tipo == 2)
24     {
25         pos_rec = byte_rec;
26         tipo = 0;
27     }
28     else if (tipo == 3)
29     {
30         espera = byte_rec;
```

```
31     tipo = 0;
32 }
33 else if (byte_rec == 181 && tipo == 0)
34 {
35     tipo = 1;
36 }
37 else if (byte_rec == 182 && tipo == 0)
38 {
39     tipo = 2;
40 }
41 else if (byte_rec == 183 && tipo == 0)
42 {
43     tipo = 3;
44 }
45 }
46
47 void setup()
48 {
49     TinyWireS.begin(DIR_I2C);
50     TinyWireS.onReceive(receiveEvent);
51     myservo.attach(SERVO_PIN);
52 }
53
54
55 void loop()
56 {
57     if (pos_actual+vel < pos_rec) pos_actual = pos_actual+ vel;
58     else if (pos_actual -vel > pos_rec) pos_actual = pos_actual- vel
59     ;
60     else pos_actual=pos_rec;
61
62     myservo.write(pos_actual);
63     delay(espera);
64     SoftRcPulseOut::refresh(NOW);
65
66     TinyWireS_stop_check();
67 }
```

A.0.2. cliente I2c en módulos sensores

```
1 #include <TinyWireS.h>
2
3 #define DIRECCION_I2C 0x32
4 //#define LED_OLIMEXINO 0
5 #define IR_SENSOR 3
6
7 int sensor=1,i=0;
8 uint8_t estado_sensor=0;
9
10 void setup()
11 {
12
13   TinyWireS.begin(DIRECCION_I2C);
14   TinyWireS.onRequest(requestEvent);
15
16   pinMode(IR_SENSOR,INPUT);
17   //pinMode(LED_OLIMEXINO, OUTPUT);
18   //digitalWrite(LED_OLIMEXINO,HIGH);
19
20 }
21
22
23 void loop()
24 {
25   sensor=digitalRead(IR_SENSOR);
26   if(sensor==0){
27     //digitalWrite(LED_OLIMEXINO,LOW);
28     estado_sensor=0;
29   }
30   else if(sensor==1){
31     //digitalWrite(LED_OLIMEXINO,HIGH);
32     estado_sensor=1;
33   }
34
35 delay(20);
36
37
38 TinyWireS_stop_check();
39 }
40
41 void requestEvent(){
```

```
42     TinyWireS.send(estado_sensor);  
43  
44 }
```

A.0.3. cliente I2c en modulos dinamixel

```
1 #include <TinyWireS.h>  
2 #include <DynamixelSerial.h>  
3  
4 #define DIRECCION_I2C 0x07  
5 #define LED_OLIMEXINO 1  
6  
7 int byte_recibido = 0;  
8 int posicion_recibida = 512;  
9 int espera = 20;  
10 int tipo = 0;  
11 int velocidad = 1000;  
12 int bloquear_servo = 1;  
13 int posicion_servo, posicion_servo_grabada;  
14 int contador_bytes = 0;  
15 int contador_bucle = 0;  
16 int variable_a_convertir = 0;  
17 int x=0;  
18 int mover=1;  
19  
20 uint8_t bytes_variable[] = {0, 0, 0, 0};  
21 uint8_t variable_convertida[] = {0, 0, 0, 0};  
22  
23 void conversor_entero_a_bytes(int variable_a_convertir)  
24 {  
25  
26     while(variable_a_convertir > 255)  
27     {  
28  
29         variable_a_convertir = variable_a_convertir - 255;  
30         variable_convertida[contador_bucle] = 255;  
31         contador_bucle++;  
32     }  
33 }  
34 }
```

```
35     variable_convertida [contador_bucle] = variable_a_convertir ;
36     contador_bucle = 0;
37
38 }
39
40 void receiveEvent( uint8_t howMany)
41 {
42
43     byte_recibido = TinyWireS.receive();
44
45     if (tipo == 1)
46     {
47         velocidad = byte_recibido;
48         tipo = 0;
49     }
50     else if (tipo == 2)
51     {
52         posicion_recibida = posicion_recibida+byte_recibido;
53         x++;
54         if (x==4){
55             tipo=0;
56             x=0;
57             mover=1;
58             digitalWrite(LED_OLIMEXINO,HIGH);
59         }
60     }
61     else if (tipo == 3)
62     {
63         espera = byte_recibido;
64         tipo = 0;
65     }
66     else if (byte_recibido == 181 && tipo == 0)
67     {
68         tipo = 1;
69     }
70     else if (byte_recibido == 182 && tipo == 0)
71     {
72         posicion_recibida=0;
73         mover=0;
74         tipo = 2;
75         digitalWrite(LED_OLIMEXINO,LOW);
```

```
76     }
77     else if (byte_recibido == 183 && tipo == 0)
78     {
79         tipo = 3;
80     }
81     else if (byte_recibido == 184 && tipo == 0)
82     {
83         bloquear_servo = 0;
84         tipo=4;
85     }
86 }
87
88 void requestEvent()
89 {
90     if (tipo==4){
91         if(contador_bytes == 0)
92         {
93             posicion_recibida=posicion_servo ;
94
95             conversor_entero_a_bytes(posicion_servo);
96             TinyWireS.send(variable_convertida [ contador_bytes ]) ;
97             contador_bytes++;
98
99         }
100        else if(contador_bytes == 1)
101        {
102
103            TinyWireS.send(variable_convertida [ contador_bytes ]) ;
104            contador_bytes++;
105
106        }
107        else if(contador_bytes == 2)
108        {
109
110            TinyWireS.send(variable_convertida [ contador_bytes ]) ;
111            contador_bytes++;
112
113        }
114        else if(contador_bytes == 3)
115        {
116
```

```
117     TinyWireS.send(variable_convertida[contador_bytes]);  
118     contador_bytes = 0;  
119  
120     bloquear_servo = 1;  
121     variable_convertida[0]=0;  
122     variable_convertida[1]=0;  
123     variable_convertida[2]=0;  
124     variable_convertida[3]=0;  
125  
126     tipo=0;  
127 }  
128 }  
129 }  
130 }  
131  
132  
133 void setup()  
134 {  
135     delay(1400);  
136     TinyWireS.begin(DIRECCION_I2C);  
137     TinyWireS.onReceive(receiveEvent);  
138     TinyWireS.onRequest(requestEvent);  
139  
140     Dynamixel.begin(19200, 4);  
141     pinMode(LED_OLIMEXINO, OUTPUT);  
142     // digitalWrite(LED_OLIMEXINO,HIGH);  
143  
144 }  
145 }  
146  
147  
148 void loop()  
149 {  
150  
151     if(mover==1){  
152         Dynamixel.moveSpeed(1, posicion_recibida, velocidad);  
153         delay(espera);  
154     }  
155  
156     while(bloquear_servo==0){  
157         Dynamixel.torqueStatus(1, false);
```

```
158     digitalWrite(LED_OLIMEXINO,LOW) ;
159     posicion_servo = Dynamixel.readPosition(1) ;
160     delay(espera) ;
161 }
162 digitalWrite(LED_OLIMEXINO,HIGH) ;
163
164
165 TinyWireS_stop_check() ;
166
167 }
```

B. Código en ROS del Nodo Servidor para el control de los movimientos de robots serpiente

El siguiente código corresponde a un servidor en ROS para la generación y control de todos los modos de funcionamiento descritos en la interfaz de este tipos de robots

```
1 /*
2 * gusano.cpp
3 *
4 * Created on: 20 de feb. de 2016
5 * Author: kaiser
6 */
7
8 #include "gusano.h"
9
10 #include "OscillatorSerp.h"
11
12 //-- Grados to rad
13 #define DEG2RAD(g) ((g)*M_PI)/180
14
15 Gusano::Gusano()
16 {
17     //-- inicialmente 0 servos
18     _nservos=0;
19 }
```

```
20 //-- fase inicial
21 _phase0=0;
22
23 //-- diferencia inicial de fase entre servos
24 _diferencia_fase=-120;
25 }
26
27 void Gusano::add_servo( std::string topico )
28 {
29     Osc[_nservos].attach( topico );
30
31     //-- calculo de la fase de cada servo
32     Osc[_nservos].SetFase(DEG2RAD(_phase0 + _nservos*
33         _diferencia_fase));
34     _nservos++;
35
36 void Gusano::refresh()
37 {
38     //-- actualizar posicion
39     for ( int i=0; i<_nservos; i++)
40         Osc[ i ].actualizaPosicionServo();
41 }
42
43
44 void Gusano::set_wave(Wave w, int servo )
45 {
46     //-- Asignamos la ola a un servo o a todos
47     if ( servo >= 0 && servo < _nservos)
48     {
49         Osc[ servo ].SetPosicion(w.A);
50         Osc[ servo ].SetOffset(w.O);
51         Osc[ servo ].SetPeriodo(w.T);
52         Osc[ servo ].SetFase(DEG2RAD(w.PD));
53     }
54     else
55     {
56         for ( int i=0; i<_nservos; i++)
57         {
58             Osc[ i ].SetPosicion(w.A);
59             Osc[ i ].SetOffset(w.O);
```

```

60         Osc[ i ]. SetPeriodo(w.T);
61         Osc[ i ]. SetFase(DEG2RAD(w.PHASE0 + i*w.PD));
62     }
63 }
64
65 _phase0=w.PHASE0;
66 }
67
68 void Gusano::SetA(int A, int servo)
69 {
70     //-- Actualizar la posicion de solo un servo de la serpiente si
71     // se especifica
72     if (servo >= 0 && servo < _nservos)
73         Osc[ servo ]. SetPosicion(A);
74     else
75         for (int i=0; i<_nservos; i++)
76             Osc[ i ]. SetPosicion(A);
77 }
78 void Gusano::SetT(unsigned int T, int servo)
79 {
80     //-- Actualizar el periodo de solo un servo de la serpiente si
81     // se especifica
82     if (servo >= 0 && servo < _nservos)
83         Osc[ servo ]. SetPeriodo(T);
84     else
85         for (int i=0; i<_nservos; i++)
86             Osc[ i ]. SetPeriodo(T);
87 }
88 void Gusano::SetO(int O, int servo)
89 {
90     //-- Actualizar offset de solo un servo de la serpiente si se
91     // especifica
92     if (servo >= 0 && servo < _nservos)
93         Osc[ servo ]. SetOffset(O);
94     else
95         for (int i=0; i<_nservos; i++)
96             Osc[ i ]. SetOffset(O);
97 }
```

```
98 void Gusano::SetPd( int Pd, int servo )
99 {
100     //-- Actualizar el desfase de solo un servo de la serpiente si
101     //   se especifica
102     if ( servo >= 0 && servo < _nservos )
103         Osc[ servo ].SetFase(DEG2RAD(Pd));
104     else
105         for ( int i=0; i<_nservos; i++)
106             Osc[ i ].SetFase(DEG2RAD(_phase0 + i *Pd));
107
108 void Gusano::SetPh0( int Ph0 )
109 {
110     _phase0 = Ph0;
111     SetPd(_diferencia_fase);
112 }
```

C. Código en ROS del Nodo Servidor para el control de los movimientos de robots humanoides

C.1. Caminada hacia delante

```
1 #!/usr/bin/env python
2
3 import actionlib
4 import rospy
5
6 from control_msgs.msg import FollowJointTrajectoryAction ,
6     FollowJointTrajectoryGoal
7 from trajectory_msgs.msg import JointTrajectory ,
7     JointTrajectoryPoint
8 import urdf_parser_py.urdf
9 import urdf_parser_py.xml_reflection
10 import kdl_parser_py.urdf
11 import PyKDL
12 import copy
```

```
13 from angles import normalize
14
15 import pickle
16
17 class TrajectoryDemo():
18     def __init__(self):
19
20         filename = '/home/kaiser/WS_ROS/catkin_ws/src/urdf_serp/
21             urdf/urdf_exportado4.urdf'
22         datos_articulaciones = open('datos_articulaciones.pkl', 'wb')
23
24         robot = urdf_parser_py.urdf.URDF.from_xml_file(filename)
25
26         (ok, tree) = kdl_parser_py.urdf.treeFromFile(filename)
27         cadena_der_up_down = tree.getChain("base_link", "pie_der_link")
28         cadena_der_down_up = tree.getChain("pie_der_link", "base_link")
29         cadena_izq_up_down = tree.getChain("base_link", "pie_izq_link")
30         cadena_izq_down_up = tree.getChain("pie_izq_link", "base_link")
31
32         print cadena_der_up_down.getNrOfSegments()
33         fksolver_der_up_down = PyKDL.ChainFkSolverPos_recursive(
34             cadena_der_up_down)
35         fksolver_der_down_up = PyKDL.ChainFkSolverPos_recursive(
36             cadena_der_down_up)
37         fksolver_izq_up_down = PyKDL.ChainFkSolverPos_recursive(
38             cadena_izq_up_down)
39         fksolver_izq_down_up = PyKDL.ChainFkSolverPos_recursive(
40             cadena_izq_down_up)
41
42         vik_der_up_down = PyKDL.ChainIkSolverVel_pinv(
43             cadena_der_up_down)
44         ik_der_up_down = PyKDL.ChainIkSolverPos_NR(
45             cadena_der_up_down, fksolver_der_up_down,
46             vik_der_up_down)
```

```

40     vik_der_down_up = PyKDL.ChainIkSolverVel_pinv(
41         cadena_der_down_up)
42     ik_der_down_up = PyKDL.ChainIkSolverPos_NR(
43         cadena_der_down_up, fksolver_der_down_up,
44         vik_der_down_up)
45
46     vik_izq_up_down = PyKDL.ChainIkSolverVel_pinv(
47         cadena_izq_up_down)
48     ik_izq_up_down = PyKDL.ChainIkSolverPos_NR(
49         cadena_izq_up_down, fksolver_izq_up_down,
50         vik_izq_up_down)
51
52     vik_izq_down_up = PyKDL.ChainIkSolverVel_pinv(
53         cadena_izq_down_up)
54     ik_izq_down_up = PyKDL.ChainIkSolverPos_NR(
55         cadena_izq_down_up, fksolver_izq_down_up,
56         vik_izq_down_up)
57
57
58     nj_izq = cadena_der_up_down.getNrOfJoints()
59     posicionInicial_der_up_down = PyKDL.JntArray(nj_izq)
60     posicionInicial_der_up_down[0] = 0.3
61     posicionInicial_der_up_down[1] = -0.3
62     posicionInicial_der_up_down[2] = 0
63     posicionInicial_der_up_down[3] = 0.6
64     posicionInicial_der_up_down[4] = -0.3
65     posicionInicial_der_up_down[5] = -0.3
66
66
67     nj_izq = cadena_izq_up_down.getNrOfJoints()
68     posicionInicial_izq_up_down = PyKDL.JntArray(nj_izq)
69     posicionInicial_izq_up_down[0] = 0.3
70     posicionInicial_izq_up_down[1] = -0.3
71     posicionInicial_izq_up_down[2] = 0.0
72     posicionInicial_izq_up_down[3] = 0.6
73     posicionInicial_izq_up_down[4] = -0.3
74     posicionInicial_izq_up_down[5] = -0.3
75
75
76     nj_izq = cadena_der_down_up.getNrOfJoints()
77     posicionInicial_der_down_up = PyKDL.JntArray(nj_izq)
78     posicionInicial_der_down_up[5] = 0.3
79     posicionInicial_der_down_up[4] = -0.3
80     posicionInicial_der_down_up[3] = 0.0

```

```

72     posicionInicial_der_down_up [2] = 0.6
73     posicionInicial_der_down_up [1] = -0.3
74     posicionInicial_der_down_up [0] = -0.3
75
76     nj_izq = cadena_izq_down_up.getNrOfJoints()
77     posicionInicial_izq_down_up = PyKDL.JntArray(nj_izq)
78     posicionInicial_izq_down_up [5] = 0.3
79     posicionInicial_izq_down_up [4] = -0.3
80     posicionInicial_izq_down_up [3] = 0.0
81     posicionInicial_izq_down_up [2] = 0.6
82     posicionInicial_izq_down_up [1] = -0.3
83     posicionInicial_izq_down_up [0] = -0.3
84     print "Forward kinematics"
85     finalFrame_izq_up_down = PyKDL.Frame()
86     finalFrame_izq_down_up = PyKDL.Frame()
87     finalFrame_der_up_down = PyKDL.Frame()
88     finalFrame_der_down_up = PyKDL.Frame()
89
90
91     print "Rotational Matrix of the final Frame: "
92     print finalFrame_izq_up_down.M
93     print "End-effector position: ", finalFrame_izq_up_down.p
94
95     rospy.init_node('trajectory_demo')
96
97     # Set to True to move back to the starting configurations
98     reset = rospy.get_param('~reset', False)
99
100    # Set to False to wait for arm to finish before moving
101    head
102    sync = rospy.get_param('~sync', True)
103
104    # Which joints define the arm?
105    piernas_joints = ['cilinder_blue1_box1_der_joint', ,
106                      'cilinder_blue_box1_der_joint', ,
107                      'cilinder_blue_box2_der_joint', ,
108                      'cilinder_blue_box4_der_joint', ,
109                      'cilinder_blue1_box6_der_joint', ,
110                      'cilinder_blue_box6_der_joint', ,
111                      'cilinder_blue1_box1_izq_joint', ,
112                      'cilinder_blue_box1_izq_joint', ,

```

```

107                 cilinder_blue_box2_izq_joint' ,
108                 'cilinder_blue_box4_izq_joint' , ,
109                 cilinder_blue1_box6_izq_joint' , ,
110                 cilinder_blue_box6_izq_joint' ]
111
112
113     piernas_goal = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
114                         0.0, 0.0, 0.0]
115
116     #111111111111111111111111111111111111111111111111111111111111111111
117     print "Inverse Kinematics"
118     fkssolver_izq_up_down.JntToCart(posicionInicial_izq_up_down
119                                         , finalFrame_izq_up_down)
120     fkssolver_izq_down_up.JntToCart(posicionInicial_izq_down_up
121                                         , finalFrame_izq_down_up)
122     q_init_izq_up_down = posicionInicial_izq_up_down # initial angles
123     desiredFrame = finalFrame_izq_up_down
124     desiredFrame.p[0] = finalFrame_izq_up_down.p[0]
125     desiredFrame.p[1] = finalFrame_izq_up_down.p[1]
126     desiredFrame.p[2] = finalFrame_izq_up_down.p[2]
127     print "Desired Position: ", desiredFrame.p
128     q_out_izq_up_down = PyKDL.JntArray(cadena_izq_up_down.
129                                         getNrOfJoints())
130     ik_izq_up_down.CartToJnt(q_init_izq_up_down, desiredFrame,
131                               q_out_izq_up_down)
132     print "Output angles in rads: ", q_out_izq_up_down
133
134     piernas_goal[6] = q_out_izq_up_down[0]
135     piernas_goal[7] = q_out_izq_up_down[1]
136     piernas_goal[8] = q_out_izq_up_down[2]
137     piernas_goal[9] = q_out_izq_up_down[3]
138     piernas_goal[10] = q_out_izq_up_down[4]
139     piernas_goal[11] = q_out_izq_up_down[5]
140
141
142     print "Inverse Kinematics"
143     fkssolver_der_up_down.JntToCart(posicionInicial_der_up_down
144                                         , finalFrame_der_up_down)
145     fkssolver_der_down_up.JntToCart(posicionInicial_der_down_up
146                                         , finalFrame_der_down_up)

```

```

136     q_init_der_up_down = posicionInicial_der_up_down #  
137         initial angles  
138     # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))  
139     desiredFrame = finalFrame_der_up_down  
140     desiredFrame.p[0] = finalFrame_der_up_down.p[0]  
141     desiredFrame.p[1] = finalFrame_der_up_down.p[1]  
142     desiredFrame.p[2] = finalFrame_der_up_down.p[2]+0.02 #0.02  
143     print "Desired Position: ", desiredFrame.p  
144     q_out_der_up_down = PyKDL.JntArray(cadena_der_up_down.  
145         getNrOfJoints())  
146     ik_der_up_down.CartToJnt(q_init_der_up_down, desiredFrame,  
147         q_out_der_up_down)  
148     print "Output angles in rads: ", q_out_der_up_down  
149  
150     piernas_goal[0] = q_out_der_up_down[0]  
151     piernas_goal[1] = q_out_der_up_down[1]  
152     piernas_goal[2] = q_out_der_up_down[2]  
153     piernas_goal[3] = q_out_der_up_down[3]  
154     piernas_goal[4] = q_out_der_up_down[4]  
155     piernas_goal[5] = q_out_der_up_down[5]  
156  
157     #121212122121212121212121212121212121212121212121212  
158     piernas_goal12 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,  
159         0.0, 0.0, 0.0, 0.0]  
160     print "Inverse Kinematics"  
161     desiredFrame = PyKDL.Frame()  
162     fksolver_izq_up_down.JntToCart(q_out_izq_up_down,  
163         desiredFrame)  
164     fksolver_izq_down_up.JntToCart(posicionInicial_izq_down_up  
165         , finalFrame_izq_down_up)  
166     q_init_izq_up_down = posicionInicial_izq_up_down #  
167         initial angles  
168     desiredFrame.p[0] = desiredFrame.p[0]  
169     desiredFrame.p[1] = desiredFrame.p[1]  
170     desiredFrame.p[2] = desiredFrame.p[2]  
171     print "Desired Position: ", desiredFrame.p  
172     q_out_izq_up_down = PyKDL.JntArray(cadena_izq_up_down.  
173         getNrOfJoints())  
174     ik_izq_up_down.CartToJnt(q_init_izq_up_down, desiredFrame,  
175         q_out_izq_up_down)  
176     print "Output angles in rads: ", q_out_izq_up_down

```

```

168
169      piernas_goal12[6] = q_out_izq_up_down[0]
170      piernas_goal12[7] = q_out_izq_up_down[1]
171      piernas_goal12[8] = q_out_izq_up_down[2]
172      piernas_goal12[9] = q_out_izq_up_down[3]
173      piernas_goal12[10] = q_out_izq_up_down[4]
174      piernas_goal12[11] = q_out_izq_up_down[5]
175
176      print "Inverse Kinematics"
177      desiredFrame0 = PyKDL.Frame()
178      fkSolver_der_up_down.JntToCart(q_out_der_up_down,
179          desiredFrame0)
180      fkSolver_der_down_up.JntToCart(posicionInicial_der_down_up
181          , finalFrame_der_down_up)
182      q_init_der_up_down = posicionInicial_der_up_down # initial angles
183      # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))
184      desiredFrame0.p[0] = desiredFrame0.p[0]
185      desiredFrame0.p[1] = desiredFrame0.p[1] -0.07
186      desiredFrame0.p[2] = desiredFrame0.p[2]
187      print "Desired Position: ", desiredFrame0.p
188      q_out_der_up_down = PyKDL.JntArray(cadena_der_up_down.
189          getNrOfJoints())
190      ik_der_up_down.CartToJnt(q_init_der_up_down, desiredFrame0
191          , q_out_der_up_down)
192      print "Output angles in rads: ", q_out_der_up_down
193
194      piernas_goal12[0] = q_out_der_up_down[0]
195      piernas_goal12[1] = q_out_der_up_down[1]
196      piernas_goal12[2] = q_out_der_up_down[2]
197      piernas_goal12[3] = q_out_der_up_down[3]
198      piernas_goal12[4] = q_out_der_up_down[4]
199      piernas_goal12[5] = q_out_der_up_down[5]
200
201      # 2222222222222222222222222222222222222222222222222222
202      piernas_goal2 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
203          0.0, 0.0, 0.0]
204      print "Inverse Kinematics"
205      desiredFrame2 = PyKDL.Frame()
206      fkSolver_izq_down_up.JntToCart(posicionInicial_izq_down_up
207          , desiredFrame2)
```

```

202     q_init_izq_down_up = posicionInicial_izq_down_up #  
203         initial angles  
204     # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))  
205     desiredFrame2.p[0] = desiredFrame2.p[0] -0.06 #0.05  
206     desiredFrame2.p[1] = desiredFrame2.p[1] -0.06#0.06  
207     desiredFrame2.p[2] = desiredFrame2.p[2] -0.01 #0.02  
208     print "Desired Position2222aaa: ", desiredFrame2.p  
209     #print desiredFrame2.M  
210     q_out_izq_down_up = PyKDL.JntArray(cadena_izq_up_down.  
211         getNrOfJoints())  
212     ik_izq_down_up.CartToJnt(q_init_izq_down_up, desiredFrame2  
213         , q_out_izq_down_up)  
214     print "Output angles in rads2222: ", q_out_izq_down_up  
215  
216     piernas_goal2[6] = q_out_izq_down_up[5]#+0.1  
217     piernas_goal2[7] = q_out_izq_down_up[4]#-0.05  
218     piernas_goal2[8] = q_out_izq_down_up[3]  
219     piernas_goal2[9] = q_out_izq_down_up[2]  
220     piernas_goal2[10] = q_out_izq_down_up[1]  
221     piernas_goal2[11] = q_out_izq_down_up[0]  
222  
223     #q_out_izq_down_up[4] -=0.1  
224  
225     print "Inverse Kinematics"  
226     q_init_der_down_up = PyKDL.JntArray(6)  
227     q_init_der_down_up[0] = q_out_der_up_down[5] # PROBLEMAS  
228         CON LA ASIGNACION  
229     q_init_der_down_up[1] = q_out_der_up_down[4]  
230     q_init_der_down_up[2] = q_out_der_up_down[3]  
231     q_init_der_down_up[3] = q_out_der_up_down[2]  
232     q_init_der_down_up[4] = q_out_der_up_down[1]  
233     q_init_der_down_up[5] = q_out_der_up_down[0]+0.08  
234     fkssolver_der_down_up.JntToCart(q_init_der_down_up,  
235         finalFrame_der_down_up)  
236     # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))  
237     desiredFrame3 = finalFrame_der_down_up  
238     desiredFrame3.p[0] = finalFrame_der_down_up.p[0]- 0.05  
239     desiredFrame3.p[1] = finalFrame_der_down_up.p[1]- 0.04  
240     desiredFrame3.p[2] = finalFrame_der_down_up.p[2]-0.02  
241     print "Desired Position2222der: ", desiredFrame3.p

```

```

237     q_out_der_down_up = PyKDL.JntArray(cadena_der_down_up .
238         getNrOfJoints())
239     ik_der_down_up.CartToJnt(q_init_der_down_up, desiredFrame3
240         , q_out_der_down_up)
241     print "Output angles in rads2222der: ", q_out_der_down_up
242     print "VALOR", q_out_der_up_down[5]
243     piernas_goal2[0] = -0.3
244     piernas_goal2[1] = -0.3
245     piernas_goal2[2] = 0
246     piernas_goal2[3] = 0.6
247     piernas_goal2[4] = 0.3
248     piernas_goal2[5] = -0.3
249
250
251     # 3333333333333333333333333333333333333333333333333333333333333333
252     piernas_goal3 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
253     print "Inverse Kinematics"
254     desiredFrame4 = PyKDL.Frame()
255     fkSolver_izq_down_up.JntToCart(q_out_izq_down_up,
256         desiredFrame4)
257     # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))
258     desiredFrame4.p[0] = desiredFrame4.p[0]
259     desiredFrame4.p[1] = desiredFrame4.p[1] - 0.02
260     desiredFrame4.p[2] = desiredFrame4.p[2]
261     ik_izq_down_up.CartToJnt(q_out_izq_down_up, desiredFrame4 ,
262         q_out_izq_down_up)
263
264     q_init_izq_up_down[0] = q_out_izq_down_up[5]
265     q_init_izq_up_down[1] = q_out_izq_down_up[4]
266     q_init_izq_up_down[2] = q_out_izq_down_up[3]
267     q_init_izq_up_down[3] = q_out_izq_down_up[2]
268     q_init_izq_up_down[4] = q_out_izq_down_up[1]
269     q_init_izq_up_down[5] = q_out_izq_down_up[0]
270
271     desiredFrame5 = PyKDL.Frame()
272     fkSolver_izq_up_down.JntToCart(q_init_izq_up_down ,
273         desiredFrame5)
274     desiredFrame5.p[0] = desiredFrame5.p[0]
275     desiredFrame5.p[1] = desiredFrame5.p[1] - 0.1
276     desiredFrame5.p[2] = desiredFrame5.p[2]+0.01
277     print "Desired Position: ", desiredFrame5.p

```

```
273     q_out_izq_up_down2 = PyKDL.JntArray(cadena_izq_up_down .  
274         getNrOfJoints ())  
275     ik_izq_up_down.CartToJnt(q_init_izq_up_down , desiredFrame5  
276         , q_out_izq_up_down2)  
277     print "Output angles in rads: " , q_out_izq_up_down2  
278     piernas_goal3 [6] = q_out_izq_up_down2 [0]  
279     piernas_goal3 [7] = q_out_izq_up_down2 [1]  
280     piernas_goal3 [8] = q_out_izq_up_down2 [2]  
281     piernas_goal3 [9] = q_out_izq_up_down2 [3]  
282     piernas_goal3 [10] = q_out_izq_up_down2 [4]#+0.15  
283     piernas_goal3 [11] = q_out_izq_up_down2 [5]-0.08  
284  
285     print "Inverse Kinematics"  
286  
287     piernas_goal3 [0] = -0.3  
288     piernas_goal3 [1] = -0.3  
289     piernas_goal3 [2] = 0  
290     piernas_goal3 [3] = 0.6  
291     piernas_goal3 [4] = 0.3  
292     piernas_goal3 [5] = -0.3  
293  
294     # 1212121221212121212121212121212121212121212121  
295     piernas_goal121 = [0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 ,  
296         0.0 , 0.0 , 0.0 , 0.0]  
297     print "Inverse Kinematics"  
298     desiredFrame6 = PyKDL.Frame()  
299     fksolver_izq_up_down .JntToCart(q_out_izq_up_down2 ,  
300         desiredFrame6)  
301     fksolver_izq_down_up .JntToCart(posicionInicial_izq_down_up  
302         , finalFrame_izq_down_up)  
303     q_init_izq_up_down = q_out_izq_up_down2 # initial angles  
304         #CUIDADO  
305     desiredFrame6.p [0] = desiredFrame6.p [0]  
306     desiredFrame6.p [1] = desiredFrame6.p [1]-0.05  
307     desiredFrame6.p [2] = desiredFrame6.p [2]#+0.01  
308     print "Desired Position: " , desiredFrame6.p  
309     q_out_izq_up_down = PyKDL.JntArray(cadena_izq_up_down .  
310         getNrOfJoints ())  
311     ik_izq_up_down.CartToJnt(q_init_izq_up_down , desiredFrame6  
312         , q_out_izq_up_down)  
313     print "Output angles in rads_izq_121: " , q_out_izq_up_down
```

```

306
307     piernas_goal121 [6] = q_out_izq_up_down[0]
308     piernas_goal121 [7] = q_out_izq_up_down[1]
309     piernas_goal121 [8] = q_out_izq_up_down[2]
310     piernas_goal121 [9] = q_out_izq_up_down[3]
311     piernas_goal121 [10] = q_out_izq_up_down[4]
312     piernas_goal121 [11] = q_out_izq_up_down[5]
313
314     print "Inverse Kinematics"
315     desiredFrame06 = PyKDL.Frame()
316     fkSolver_der_up_down.JntToCart(q_out_der_up_down,
317                                     desiredFrame06)
318     fkSolver_der_down_up.JntToCart(posicionInicial_der_down_up
319                                     , finalFrame_der_down_up)
320     q_init_der_up_down = q_out_der_up_down # initial angles
321     # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))
322     desiredFrame06.p[0] = desiredFrame06.p[0]
323     desiredFrame06.p[1] = desiredFrame06.p[1]
324     desiredFrame06.p[2] = desiredFrame06.p[2]
325     print "Desired Position: ", desiredFrame06.p
326     q_out_der_up_down = PyKDL.JntArray(cadena_der_up_down.
327                                         getNrOfJoints())
328     ik_der_up_down.CartToJnt(q_init_der_up_down,
329                               desiredFrame06, q_out_der_up_down)
330     print "Output angles in rads: ", q_out_der_up_down
331
332     q_out_der_up_down21 = PyKDL.JntArray(6)
333     q_out_der_up_down21 = [-.3, -.3, 0, .6, .3, -.3]
334     piernas_goal121[0] = q_out_der_up_down21[0]
335     piernas_goal121[1] = q_out_der_up_down21[1]
336     piernas_goal121[2] = q_out_der_up_down21[2]
337     piernas_goal121[3] = q_out_der_up_down21[3]
338     piernas_goal121[4] = q_out_der_up_down21[4]
339     piernas_goal121[5] = q_out_der_up_down21[5]
340
341     # 55555555555555555555555555555555555555555555555
342     piernas_goal25 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
343                       0.0, 0.0, 0.0, 0.0]
344     print "Inverse Kinematics"
345     desiredFrame7= PyKDL.Frame()
346     q_init_izq_down_up3 = PyKDL.JntArray(6)

```

```

342     q_init_izq_down_up3[0] = q_out_izq_up_down[5] * 1
343     q_init_izq_down_up3[1] = q_out_izq_up_down[4] * 1
344     q_init_izq_down_up3[2] = q_out_izq_up_down[3] * 1
345     q_init_izq_down_up3[3] = q_out_izq_up_down[2] * 1
346     q_init_izq_down_up3[4] = q_out_izq_up_down[1] * 1
347     q_init_izq_down_up3[5] = q_out_izq_up_down[0] * 1
348     fkssolver_izq_down_up.JntToCart(q_init_izq_down_up3 ,
349                                         desiredFrame7)
350 # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))
351 desiredFrame7.p[0] = desiredFrame7.p[0] + 0.05#0.03 #  
PROBAR A PONERLO MAYOR
352 desiredFrame7.p[1] = desiredFrame7.p[1] - 0.06#0.04
353 desiredFrame7.p[2] = desiredFrame7.p[2] + 0.005
354 print "Desired Position2222: ", desiredFrame7.p
355 q_out_izq_down_up5 = PyKDL.JntArray(cadena_izq_up_down .
356                                         getNrOfJoints())
357 ik_izq_down_up.CartToJnt(q_init_izq_down_up3 ,
358                             desiredFrame7, q_out_izq_down_up5)
359 print "Output angles in rads2222AAAAA: " ,
360         q_out_izq_down_up5
361
362 piernas_goal25[6] = q_out_izq_down_up5[5]
363 piernas_goal25[7] = q_out_izq_down_up5[4]
364 piernas_goal25[8] = q_out_izq_down_up5[3]
365 piernas_goal25[9] = q_out_izq_down_up5[2]
366 piernas_goal25[10] = q_out_izq_down_up5[1]-0.05
367 piernas_goal25[11] = q_out_izq_down_up5[0]+0.05
368
369 print "Inverse Kinematics"
370 q_init_der_down_up31 = PyKDL.JntArray(6)
371 q_init_der_down_up31[0] = q_out_der_up_down21[5] *1 #  
PROBLEMAS CON LA ASIGNACION
372 q_init_der_down_up31[1] = q_out_der_up_down21[4] *1
373 q_init_der_down_up31[2] = q_out_der_up_down21[3] *1
374 q_init_der_down_up31[3] = q_out_der_up_down21[2] *1
375 q_init_der_down_up31[4] = q_out_der_up_down21[1] *1
376 q_init_der_down_up31[5] = q_out_der_up_down21[0] *1
377 desiredFrame7 = PyKDL.Frame()
378 fkssolver_der_down_up.JntToCart(q_init_der_down_up31 ,
379                                     desiredFrame7)
380 # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))

```

```
376     desiredFrame7.p[0] = desiredFrame7.p[0] + 0.05
377     desiredFrame7.p[1] = desiredFrame7.p[1] - 0.06
378     desiredFrame7.p[2] = desiredFrame7.p[2] - 0.02
379     print "Desired Position2222der: ", desiredFrame7.p
380     q_out_der_down_up71 = PyKDL.JntArray(cadena_der_down_up.
381         getNrOfJoints())
381     ik_der_down_up.CartToJnt(q_init_der_down_up31,
382         desiredFrame7, q_out_der_down_up71)
382     print "Output angles in rads22222der: ",
383         q_out_der_down_up71
383     piernas_goal25[0] = q_out_der_down_up71[5]
384     piernas_goal25[1] = q_out_der_down_up71[4]
385     piernas_goal25[2] = q_out_der_down_up71[3]
386     piernas_goal25[3] = q_out_der_down_up71[2]
387     piernas_goal25[4] = q_out_der_down_up71[1]
388     piernas_goal25[5] = q_out_der_down_up71[0]
389
390 # 3333333333333333333333333333333333333333333333333
391     piernas_goal341 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
392     print "Inverse Kinematics"
393     desiredFrame441 = PyKDL.Frame()
394     fkSolver_der_down_up.JntToCart(q_out_der_down_up71,
395         desiredFrame441)
395     # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))
396     desiredFrame441.p[0] = desiredFrame441.p[0]
397     desiredFrame441.p[1] = desiredFrame441.p[1] - 0.02
398     desiredFrame441.p[2] = desiredFrame441.p[2] - 0.015
399     ik_der_down_up.CartToJnt(q_out_der_down_up71,
400         desiredFrame441, q_out_der_down_up71)
401
401     q_init_der_up_down[0] = q_out_der_down_up71[5]
402     q_init_der_up_down[1] = q_out_der_down_up71[4]
403     q_init_der_up_down[2] = q_out_der_down_up71[3]
404     q_init_der_up_down[3] = q_out_der_down_up71[2]
405     q_init_der_up_down[4] = q_out_der_down_up71[1]
406     q_init_der_up_down[5] = q_out_der_down_up71[0]
407
408     desiredFrame541 = PyKDL.Frame()
409     fkSolver_der_up_down.JntToCart(q_init_der_up_down,
410         desiredFrame541)
410     desiredFrame541.p[0] = desiredFrame541.p[0] - 0.03
```

```

411     desiredFrame541.p[1] = desiredFrame541.p[1] - 0.1
412     desiredFrame541.p[2] = desiredFrame541.p[2] - 0.01 #nada
413     print "Desired Position: ", desiredFrame541.p
414     q_out_der_up_down241 = PyKDL.JntArray(cadena_izq_up_down.
415                                             getNrOfJoints())
416     ik_der_up_down.CartToJnt(q_init_der_up_down,
417                               desiredFrame541, q_out_der_up_down241)# con
418                               desiredFrame5 va
419     print "Output angles in radsaaaaa: ", q_out_der_up_down241
420
421     print "Inverse Kinematics"
422
423     piernas_goal341[6] = 0.3
424     piernas_goal341[7] = -0.3
425     piernas_goal341[8] = 0
426     piernas_goal341[9] = 0.6
427     piernas_goal341[10] = -0.3
428     piernas_goal341[11] = -0.3
429
430     piernas_goal341[0] = q_out_der_up_down241[0]#+0.1
431     piernas_goal341[1] = q_out_der_up_down241[1]
432     piernas_goal341[2] = q_out_der_up_down241[2]
433     piernas_goal341[3] = q_out_der_up_down241[3]
434     piernas_goal341[4] = q_out_der_up_down241[4]# -0.1
435     piernas_goal341[5] = q_out_der_up_down241[5]# -0.01
436
437     pickle.dump(piernas_goal, datos_articulaciones, -1)
438     pickle.dump(piernas_goal12, datos_articulaciones, -1)
439     pickle.dump(piernas_goal2, datos_articulaciones, -1)
440     pickle.dump(piernas_goal3, datos_articulaciones, -1)
441     pickle.dump(piernas_goal121, datos_articulaciones, -1)
442     pickle.dump(piernas_goal25, datos_articulaciones, -1)
443     pickle.dump(piernas_goal341, datos_articulaciones, -1)
444
445     datos_articulaciones.close()
446
# Connect to the right arm trajectory action server
rospy.loginfo('Waiting for right arm trajectory controller
...')


```

```
447     arm_client = actionlib.SimpleActionClient('/piernas/  
        piernas_controller/follow_joint_trajectory',  
        FollowJointTrajectoryAction)  
448 #/piernas/piernas_controller/follow_joint_trajectory  
449 arm_client.wait_for_server()  
450  
451 rospy.loginfo('...connected.')  
452  
453  
454  
455 # Create a single-point arm trajectory with the  
#      piernas_goal as the end-point  
456 arm_trajectory = JointTrajectory()  
457 arm_trajectory.joint_names = piernas_joints  
458 arm_trajectory.points.append(JointTrajectoryPoint())  
459 arm_trajectory.points[0].positions = piernas_goal  
460 arm_trajectory.points[0].velocities = [0.0 for i in  
    piernas_joints]  
461 arm_trajectory.points[0].accelerations = [0.0 for i in  
    piernas_joints]  
462 arm_trajectory.points[0].time_from_start = rospy.Duration  
    (2.0)  
463 arm_trajectory.points.append(JointTrajectoryPoint())  
464 arm_trajectory.points[1].positions = piernas_goal12  
465 arm_trajectory.points[1].velocities = [0.0 for i in  
    piernas_joints]  
466 arm_trajectory.points[1].accelerations = [0.0 for i in  
    piernas_joints]  
467 arm_trajectory.points[1].time_from_start = rospy.Duration  
    (4.0)  
468 arm_trajectory.points.append(JointTrajectoryPoint())  
469 arm_trajectory.points[2].positions = piernas_goal2  
470 arm_trajectory.points[2].velocities = [0.0 for i in  
    piernas_joints]  
471 arm_trajectory.points[2].accelerations = [0.0 for i in  
    piernas_joints]  
472 arm_trajectory.points[2].time_from_start = rospy.Duration  
    (6.0)  
473 arm_trajectory.points.append(JointTrajectoryPoint())  
474 arm_trajectory.points[3].positions = piernas_goal3
```

```

475     arm_trajectory.points[3].velocities = [0.0 for i in
476         piernas_joints]
477     arm_trajectory.points[3].accelerations = [0.0 for i in
478         piernas_joints]
479     arm_trajectory.points[3].time_from_start = rospy.Duration
480         (8.0)
481     arm_trajectory.points.append(JointTrajectoryPoint())
482     arm_trajectory.points[4].positions = piernas_goal121
483     arm_trajectory.points[4].velocities = [0.0 for i in
484         piernas_joints]
485     arm_trajectory.points[4].accelerations = [0.0 for i in
486         piernas_joints]
487     arm_trajectory.points[4].time_from_start = rospy.Duration
488         (10.0)
489     arm_trajectory.points.append(JointTrajectoryPoint())
490     arm_trajectory.points[5].positions = piernas_goal25
491     arm_trajectory.points[5].velocities = [0.0 for i in
492         piernas_joints]
493     arm_trajectory.points[5].accelerations = [0.0 for i in
494         piernas_joints]
495     arm_trajectory.points[5].time_from_start = rospy.Duration
496         (12.0)
497     arm_trajectory.points.append(JointTrajectoryPoint())
498     arm_trajectory.points[6].positions = piernas_goal341
499     arm_trajectory.points[6].velocities = [0.0 for i in
500         piernas_joints]
501     arm_trajectory.points[6].accelerations = [0.0 for i in
502         piernas_joints]
503     arm_trajectory.points[6].time_from_start = rospy.Duration
504         (14.0)

505     '''arm_trajectory.points.append(JointTrajectoryPoint())
506     arm_trajectory.points[7].positions = piernas_goal12
507     arm_trajectory.points[7].velocities = [0.0 for i in
508         piernas_joints]
509     arm_trajectory.points[7].accelerations = [0.0 for i in
510         piernas_joints]
511     arm_trajectory.points[7].time_from_start = rospy.Duration
512         (17.5)
513     arm_trajectory.points.append(JointTrajectoryPoint())
514     arm_trajectory.points[8].positions = piernas_goal2

```

```

501     arm_trajectory.points[8].velocities = [0.0 for i in
502         piernas_joints]
503     arm_trajectory.points[8].accelerations = [0.0 for i in
504         piernas_joints]
505     arm_trajectory.points[8].time_from_start = rospy.Duration
506         (19.0)
507     arm_trajectory.points.append(JointTrajectoryPoint())
508     arm_trajectory.points[9].positions = piernas_goal3
509     arm_trajectory.points[9].velocities = [0.0 for i in
510         piernas_joints]
511     arm_trajectory.points[9].accelerations = [0.0 for i in
512         piernas_joints]
513     arm_trajectory.points[9].time_from_start = rospy.Duration
514         (21.0)
515     arm_trajectory.points.append(JointTrajectoryPoint())
516     arm_trajectory.points[10].positions = piernas_goal121
517     arm_trajectory.points[10].velocities = [0.0 for i in
518         piernas_joints]
519     arm_trajectory.points[10].accelerations = [0.0 for i in
520         piernas_joints]
521     arm_trajectory.points[10].time_from_start = rospy.Duration
522         (23.0)
523     arm_trajectory.points.append(JointTrajectoryPoint())
524     arm_trajectory.points[11].positions = piernas_goal25
525     arm_trajectory.points[11].velocities = [0.0 for i in
526         piernas_joints]
527     arm_trajectory.points[11].accelerations = [0.0 for i in
528         piernas_joints]
529     arm_trajectory.points[11].time_from_start = rospy.Duration
530         (25.0)
531     arm_trajectory.points.append(JointTrajectoryPoint())
532     arm_trajectory.points[12].positions = piernas_goal341
533     arm_trajectory.points[12].velocities = [0.0 for i in
534         piernas_joints]
535     arm_trajectory.points[12].accelerations = [0.0 for i in
536         piernas_joints]
537     arm_trajectory.points[12].time_from_start = rospy.Duration
538         (28.0) ''
539     # Send the trajectory to the arm action server
540     rospy.loginfo('Moving the arm to goal position... ')

```

```
527
528
529     # Create an empty trajectory goal
530     piernas_goal = FollowJointTrajectoryGoal()
531
532     # Set the trajectory component to the goal trajectory
533     # created above
534     piernas_goal.trajectory = arm_trajectory
535
536     # Specify zero tolerance for the execution time
537     piernas_goal.goal_time_tolerance = rospy.Duration(0.01)
538
539     # Send the goal to the action server
540     arm_client.send_goal(piernas_goal)
541
542     if not sync:
543         # Wait for up to 5 seconds for the motion to complete
544         arm_client.wait_for_result(rospy.Duration(5.0))
545
546         arm_client.wait_for_result()
547         print arm_client.get_result()
548
549         rospy.loginfo('... done')
550
551 if __name__ == '__main__':
552     try:
553         TrajectoryDemo()
554     except rospy.ROSInterruptException:
555         pass
```

C.2. Caminada hacia atrás

```
1 #!/usr/bin/env python
2
3 import actionlib
4 import rospy
5 from scipy.io import loadmat
6
```

```
 7 from control_msgs.msg import FollowJointTrajectoryAction ,  
     FollowJointTrajectoryGoal  
 8 from trajectory_msgs.msg import JointTrajectory ,  
     JointTrajectoryPoint  
 9 import urdf_parser_py.urdf  
10 import urdf_parser_py.xml_reflection  
11 import kdl_parser_py.urdf  
12 import PyKDL  
13 import copy  
14 from angles import normalize  
15  
16 import pickle  
17  
18 class TrajectoryDemo():  
19     def __init__(self):  
20  
21         x = loadmat('estruc_piernas_fase3.mat')  
22         # print x  
23         piernas_der = x['estruc_piernaDer_fase']  
24         art1_der = piernas_der[0, 0]  
25         art2_der = piernas_der[0, 1]  
26         art3_der = piernas_der[0, 2]  
27         art4_der = piernas_der[0, 3]  
28         art5_der = piernas_der[0, 4]  
29         art6_der = piernas_der[0, 5]  
30  
31         piernas_izq = x['estruc_piernaIzq_fase']  
32         art1_izq = piernas_izq[0, 0]  
33         art2_izq = piernas_izq[0, 1]  
34         art3_izq = piernas_izq[0, 2]  
35         art4_izq = piernas_izq[0, 3]  
36         art5_izq = piernas_izq[0, 4]  
37         art6_izq = piernas_izq[0, 5]  
38  
39         rospy.init_node('trajectory_demo')  
40  
41         # Set to True to move back to the starting configurations  
42         reset = rospy.get_param('~reset', False)  
43  
44         # Set to False to wait for arm to finish before moving  
           head
```

```

45     sync = rospy.get_param('~sync', True)
46
47     # Which joints define the arm?
48     piernas_joints = [ 'cylinder_blue_box1_der_joint' , ,
49                         'cylinder_blue1_box2_der_joint' , ,
50                         'cylinder_blue_box2_der_joint' ,
51                         'cylinder_blue_box4_der_joint' , ,
52                         'cylinder_blue1_box6_der_joint' , ,
53                         'cylinder_blue_box6_der_joint' ,
54                         'cylinder_blue_box1_izq_joint' , ,
55                         'cylinder_blue1_box2_izq_joint' , ,
56                         'cylinder_blue_box2_izq_joint' ,
57                         'cylinder_blue_box4_izq_joint' , ,
58                         'cylinder_blue1_box6_izq_joint' , ,
59                         'cylinder_blue_box6_izq_joint' ]
60
61
62     # Connect to the right arm trajectory action server
63     rospy.loginfo('Waiting for right arm trajectory controller
... ')
64
65     arm_client = actionlib.SimpleActionClient('/piernas/
66         piernas_controller/follow_joint_trajectory',
67         FollowJointTrajectoryAction)
68     #/piernas/piernas_controller/follow_joint_trajectory
69     arm_client.wait_for_server()
70
71     rospy.loginfo('... connected.')
72
73     arm_trajectory = JointTrajectory()
74     arm_trajectory.joint_names = piernas_joints
75
76     # piernas_goalF0 = [0, -0.3, -0.4, 0.8, -0.3, -0.4,
77     #                   0, -0.4, -0.3, 0.6, +0.4, -0.3]
78     # arm_trajectory.points.append(JointTrajectoryPoint())
79     # arm_trajectory.points[0].positions = piernas_goalF0
80     # arm_trajectory.points[0].velocities = [0.0 for k in
81     #                                         piernas_goalF0]
82     # arm_trajectory.points[0].accelerations = [0.0 for k in
83     #                                         piernas_goalF0]

```

```

73      # arm_trajectory.points[0].time_from_start = rospy.
74          Duration(2)
75      #
76      # piernas_goalF01 = [0 * art1_der['senialCompleta'][0, 0],
77      #                      0.4*art2_der['senialCompleta'][0, 0],art3_der['
78      #                      senialCompleta'][0, 0] ,
79      #                      art4_der['senialCompleta'][0, 0] , 0.4*
80      #                      art5_der['senialCompleta'][0, 0],art6_der['
81      #                      senialCompleta'][0, 0] +0.1,
82      #                      0 * art1_izq['senialCompleta'][0, 0] ,
83      #                      0.4*art2_izq['senialCompleta'][0, 0],art3_izq['
84      #                      senialCompleta'][0, 0] ,
85      #                      art4_izq['senialCompleta'][0, 0] , 0.4*
86      #                      art5_izq['senialCompleta'][0, 0],art6_izq['
87      #                      senialCompleta'][0, 0] +0.1]
88      # arm_trajectory.points.append(JointTrajectoryPoint())
89      # arm_trajectory.points[0].positions = piernas_goalF01
90      # arm_trajectory.points[0].velocities = [0.0 for k in
91      #                                         piernas_goalF01]
92      # arm_trajectory.points[0].accelerations = [0.0 for k in
93      #                                             piernas_goalF01]
94      # arm_trajectory.points[0].time_from_start = rospy.
95          Duration(2)

96      piernas_goalF01 = [0 * art1_der['senialCompleta'][0,
97          art1_der['senialCompleta'].shape[1]-1],
98          0.6*art2_der['senialCompleta'][0,
99          art1_der['senialCompleta'].shape
100         [1]-1],
101         art3_der['senialCompleta'][0, art1_der
102             ['senialCompleta'].shape[1]-1] ,
103             art4_der['senialCompleta'][0, art1_der['
104             senialCompleta'].shape[1]-1] ,
105             0.6*art5_der['senialCompleta'][0,
106             art1_der['senialCompleta'].shape
107             [1]-1]+0.015,
108             art6_der['senialCompleta'][0, art1_der
109                 ['senialCompleta'].shape[1]-1]
110                 -0.03,
111             0 * art1_izq['senialCompleta'][0,
112             art1_der['senialCompleta'].shape
113             [1]-1]
114             ]

```

```
        [1]-1],  
92      0.6*art2_izq[ 'senialCompleta '][0,  
93          art1_der[ 'senialCompleta '].shape  
94          [1]-1],  
95      art3_izq[ 'senialCompleta '][0, art1_der  
96          [ 'senialCompleta '].shape[1]-1] ,  
97      art4_izq[ 'senialCompleta '][0, art1_der[  
98          'senialCompleta '].shape[1]-1] ,  
99      0.6*art5_izq[ 'senialCompleta '][0,  
100         art1_der[ 'senialCompleta '].shape  
101         [1]-1]-0.03,  
102     art6_izq[ 'senialCompleta '][0, art1_der  
103         [ 'senialCompleta '].shape[1]-1]  
104         -0.03]  
105  
106     arm_trajectory.points.append(JointTrajectoryPoint())  
107     arm_trajectory.points[0].positions = piernas_goalF01  
108     arm_trajectory.points[0].velocities = [0.0 for k in  
109       piernas_goalF01]  
110     arm_trajectory.points[0].accelerations = [0.0 for k in  
111       piernas_goalF01]  
112     arm_trajectory.points[0].time_from_start = rospy.Duration  
113       (2)  
114  
115  
116     ind = 1;  
117     salto =5;  
118     for i in range( int(art1_der[ 'senialCompleta '].shape[1])  
119       -1,int(art1_der[ 'senialCompleta '].shape[1]*0.75) , -  
120       salto):  
121       # print i  
122       # print int(art1_der[ 'senialCompleta '].shape[1]*0.2)  
123       piernas_goalF = [0*art1_der[ 'senialCompleta '][0, i],  
124           0.6*art2_der[ 'senialCompleta '][0, i], art3_der[  
125           'senialCompleta '][0, i],  
126           art4_der[ 'senialCompleta '][0, i],  
127           0.6*art5_der[ 'senialCompleta '][0,  
128             i]+0.015, art6_der[ 'senialCompleta  
129             '][0, i]-0.03,  
130           0*art1_izq[ 'senialCompleta '][0, i],  
131           0.6*art2_izq[ 'senialCompleta '][0,  
132             i], art3_izq[ 'senialCompleta '][0,
```

```

112             i] ,
113             art4_izq[ 'senialCompleta'][0 , i] ,
114             0.6*art5_izq[ 'senialCompleta'][0 ,
115             i]-0.03, art6_izq[ 'senialCompleta
116             ][0 , i]-0.03]
117             arm_trajectory . points . append( JointTrajectoryPoint ())
118             arm_trajectory . points [ind] . positions = piernas_goalF
119             # arm_trajectory . points [ind] . velocities = [0.0 for j
120                 in piernas_goalF]
121             # arm_trajectory . points [ind] . accelerations = [0.0 for
122                 j in piernas_goalF]
123             # art1_der[ 'tiempo_completo'][0 , i]
124             # 0.01 * (ind + 1)
125             arm_trajectory . points [ind] . time_from_start = rospy .
126             Duration( 2+art1_der[ 'tiempo_completo'][0 , art1_der
127             [ 'senialCompleta' ]. shape[1]-i])
128             # print 1+art1_der[ 'tiempo_completo'][0 , i]
129             ind+=1
130             # print art1_der[ 'senialCompleta'][0 , i]
131
132
133
134
135
136
137
138
139
140
141
142
143
# Create an empty trajectory goal
piernas_goal = FollowJointTrajectoryGoal()

# Set the trajectory component to the goal trajectory
# created above
piernas_goal.trajectory = arm_trajectory

# Specify zero tolerance for the execution time
piernas_goal.goal_time_tolerance = rospy.Duration(0.001)

# Send the goal to the action server
arm_client.send_goal(piernas_goal)

if not sync:
    # Wait for up to 5 seconds for the motion to complete
    arm_client.wait_for_result(rospy.Duration(5.0))

arm_client.wait_for_result()
# arm_client.send_goal(piernas_goal)

```

```
144         print arm_client.get_result()
145
146
147         rospy.loginfo('... done')
148
149 if __name__ == '__main__':
150     try:
151         TrajectoryDemo()
152     except rospy.ROSInterruptException:
153         pass
```

C.3. Girar sobre un pie (rotar sobre si mismo)

```
1 #!/usr/bin/env python
2
3 import actionlib
4 import rospy
5 from scipy.io import loadmat
6
7 from control_msgs.msg import FollowJointTrajectoryAction,
8     FollowJointTrajectoryGoal
9 from trajectory_msgs.msg import JointTrajectory,
10    JointTrajectoryPoint
11 import urdf_parser_py.urdf
12 import urdf_parser_py.xml_reflection
13 import kdl_parser_py.urdf
14 import PyKDL
15 import copy
16 from angles import normalize
17
18 import pickle
19
20 class TrajectoryDemo():
21     def __init__(self):
22         porcentaje = 0.7
23         porcentaje1 = 0.7
24         porcentaje16 = 0.8
25         des6 = 0.1
26
27         # porcentaje = 0.5497 # 0.5
```

```

26      # porcentaje1 = 0.5497 # 1
27      # porcentaje16 = 0.6497 # 1
28      # des6 = 0.1 # 0.15
29
30      giro = 1
31      girod = 0#1
32      x = loadmat('estruc_piernas_fase3.mat')
33      # print x
34      piernas_der = x['estruc_piernaDer_fase']
35      art1_der = piernas_der[0, 0]
36      art2_der = piernas_der[0, 1]
37      art3_der = piernas_der[0, 2]
38      art4_der = piernas_der[0, 3]
39      art5_der = piernas_der[0, 4]
40      art6_der = piernas_der[0, 5]
41
42      piernas_izq = x['estruc_piernaIzq_fase']
43      art1_izq = piernas_izq[0, 0]
44      art2_izq = piernas_izq[0, 1]
45      art3_izq = piernas_izq[0, 2]
46      art4_izq = piernas_izq[0, 3]
47      art5_izq = piernas_izq[0, 4]
48      art6_izq = piernas_izq[0, 5]
49
50      rospy.init_node('trajectory_demo')
51
52      # Set to True to move back to the starting configurations
53      reset = rospy.get_param('~reset', False)
54
55      # Set to False to wait for arm to finish before moving
56          head
57      sync = rospy.get_param('~sync', True)
58
59      # Which joints define the arm?
60      piernas_joints = ['cilinder_blue_box1_der_joint', ,
61                         'cilinder_blue1_box2_der_joint', ,
62                         'cilinder_blue_box2_der_joint',
63                         'cilinder_blue_box4_der_joint', ,
64                         'cilinder_blue1_box6_der_joint', ,
65                         'cilinder_blue_box6_der_joint', ,

```

```

61          'cylinder_blue_box1_izq_joint' , ,
62          cylinder_blue1_box2_izq_joint' , ,
63          cylinder_blue_box2_izq_joint' ,
64          'cylinder_blue_box4_izq_joint' , ,
65          cylinder_blue1_box6_izq_joint' , ,
66          cylinder_blue_box6_izq_joint' ]
67
68
69
70      # Connect to the right arm trajectory action server
71      rospy.loginfo('Waiting for right arm trajectory controller
72      ...')
73
74      arm_client = actionlib.SimpleActionClient('/piernas/
75          piernas_controller/follow_joint_trajectory',
76          FollowJointTrajectoryAction)
77      #/piernas/piernas_controller/follow_joint_trajectory
78      arm_client.wait_for_server()
79
80
81      rospy.loginfo('...connected.')
82
83      hacia_delante = JointTrajectory()
84      hacia_delante.joint_names = piernas_joints
85
86      # piernas_goalF0 = [0, -0.3, -0.4, 0.8, -0.3, -0.4,
87      #                      0, -0.4, -0.3, 0.6, +0.4, -0.3]
88      # hacia_delante.points.append(JointTrajectoryPoint())
89      # hacia_delante.points[0].positions = piernas_goalF0
90      # hacia_delante.points[0].velocities = [0.0 for k in
91      #                                         piernas_goalF0]
92      # hacia_delante.points[0].accelerations = [0.0 for k in
93      #                                         piernas_goalF0]
94      # hacia_delante.points[0].time_from_start = rospy.Duration
95      #                                         (2)
96      ind_de_inicio = 100
97      piernas_goalF01 = [giro * porcentaje * art1_der['
98          senialCompleta'][0, ind_de_inicio], 0.4 * porcentaje *
99          art2_der['senialCompleta'][0, ind_de_inicio],
100         porcentaje1 * art3_der['senialCompleta'][0, ind_de_inicio
101     ] ],

```

```

87      porcentaje1 *art4_der[ 'senalCompleta
          '][0 , ind_de_inicio] , 0.4*
          porcentaje*art5_der[ 'senalCompleta
          '][0 , ind_de_inicio], porcentaje16*
          art6_der[ 'senalCompleta '][0 ,
          ind_de_inicio] +des6 ,
88      giro* porcentaje*art1_izq[ '
          senialCompleta '][0 , ind_de_inicio] ,
          0.4* porcentaje*art2_izq[ '
          senialCompleta '][0 , ind_de_inicio] ,
          porcentaje1*art3_izq[ 'senalCompleta
          '][0 , ind_de_inicio] ,
          porcentaje1 *art4_izq[ 'senalCompleta
          '][0 , ind_de_inicio] , 0.4*
          porcentaje*art5_izq[ 'senalCompleta
          '][0 , ind_de_inicio], porcentaje16*
          art6_izq[ 'senalCompleta '][0 ,
          ind_de_inicio] +des6 ]
89      hacia_delante . points . append( JointTrajectoryPoint ())
90      hacia_delante . points [0] . positions = piernas_goalF01
91      hacia_delante . points [0] . velocities = [0.0 for k in
92          piernas_goalF01]
93      hacia_delante . points [0] . accelerations = [0.0 for k in
94          piernas_goalF01]
95      hacia_delante . points [0] . time_from_start = rospy . Duration
          (1.5)
96
97      ind = 1
98      salto =4
99      for i in range(ind_de_inicio , int(art1_der[ 'senalCompleta
          ']. shape [1]*0.08) , salto):
100         # print ind
101         # print int(art1_der[ 'senalCompleta ']. shape [1]*0.2)
102         piernas_goalF = [ giro*porcentaje*art1_der[ '
          senialCompleta '][0 , i] , 0.4*porcentaje*art2_der[ '
          senialCompleta '][0 , i] , porcentaje1*art3_der[ '
          senialCompleta '][0 , i] ,
          porcentaje1 *art4_der[ 'senalCompleta
          '][0 , i] , 0.4*porcentaje*art5_der
          [ 'senalCompleta '][0 , i] ,
          porcentaje16*art6_der[ '

```

```

103         senialCompleta ][0 , i]+des6 ,
104         giro*porcentaje*art1_izq [
105         senialCompleta ][0 , i] , 0.4*
106         porcentaje*art2_izq [
107         senialCompleta ][0 , i] ,
108         porcentaje1*art3_izq [
109         senialCompleta ][0 , i] ,
110         porcentaje1 *art4_izq [ senialCompleta
111         ][0 , i] , 0.4* porcentaje*art5_izq
112         [ 'senialCompleta ][0 , i] ,
113         porcentaje16*art6_izq [
114         senialCompleta ][0 , i]+des6 ]
115
116
117
118     arm_trajectory = JointTrajectory()
119     arm_trajectory.joint_names = piernas_joints
120
121     porcentajeg = porcentaje
122     girod = 1 # 1
123     long_atras = 0.9
124     piernas_goalF01 = [-girod*porcentajeg * art1_der [
125         senialCompleta ][0 , int((art1_der[ 'senialCompleta '].
126         shape[1] - 1)*0.93)] ,
127         porcentajeg*0.6 * art2_der [
128         senialCompleta ][0 , int((art1_der[ 'senialCompleta '].
129         shape[1] - 1)*0.93)]

```

```

126           ] ,
127   long_atras*porcentajeg*art3_der[ '
128     senialCompleta'][0 , int((art1_der[ '
129       senialCompleta'].shape[1] - 1)*0.93)
130     ] ,
131   long_atras*porcentajeg*art4_der[ '
132     senialCompleta'][0 , int((art1_der[ '
133       senialCompleta'].shape[1] - 1)*0.93)
134     ] + 0.015 ,
135   porcentajeg*0.6 * art5_der[ '
136     senialCompleta'][0 , int((art1_der[ '
137       senialCompleta'].shape[1] - 1)*0.93)
138     ] - 0.03 ,
139   -girod*porcentajeg * art1_izq[ '
140     senialCompleta'][0 , int((art1_der[ '
141       senialCompleta'].shape[1] - 1)*0.93)
142     ] ,
143   porcentajeg*0.6 * art2_izq[ '
144     senialCompleta'][0 , int((art1_der[ '
145       senialCompleta'].shape[1] - 1)*0.93)
146     ] ,
147   long_atras*porcentajeg*art3_izq[ '
148     senialCompleta'][0 , int((art1_der[ '
149       senialCompleta'].shape[1] - 1)*0.93)
150     ] ,
151   long_atras*porcentajeg*art4_izq[ '
152     senialCompleta'][0 , int((art1_der[ '
153       senialCompleta'].shape[1] - 1)*0.93)
154     ] ,
155   porcentajeg*0.6 * art5_izq[ '
156     senialCompleta'][0 , int((art1_der[ '
157       senialCompleta'].shape[1] - 1)*0.93)
158     ] - 0.03 ,
159   long_atras*porcentajeg*art6_izq[ '
160     senialCompleta'][0 , int((art1_der[ '
161       senialCompleta'].shape[1] - 1)*0.93)
162     ] - 0.03]

```

```

136     arm_trajectory.points.append(JointTrajectoryPoint())
137     arm_trajectory.points[0].positions = piernas_goalF01
138     # arm_trajectory.points[0].velocities = [0.0 for k in
139     #     piernas_goalF01]
140     # arm_trajectory.points[0].accelerations = [0.0 for k in
141     #     piernas_goalF01]
142     arm_trajectory.points[0].time_from_start = rospy.Duration
143         (1)
144
145     ind = 1;
146     salto = 5;
147     for i in range(int((art1_der['senialCompleta'].shape[1] -
148     1)*0.93), int(art1_der['senialCompleta'].shape[1] *
0.89), #0.93
149             -salto):
150         # print i
151         # print int(art1_der['senialCompleta'].shape[1]*0.2)
152         piernas_goalF = [-girod*porcentajeg * art1_der['
153             senialCompleta'][0, i], 0.6 *porcentajeg* art2_der[
154                 'senialCompleta'][0, i], long_atras*porcentajeg*
155                 art3_der['senialCompleta'][0, i],
156                     long_atras*porcentajeg*art4_der['
157                     senialCompleta'][0, i], 0.6 *
158                     porcentajeg* art5_der['
159                     senialCompleta'][0, i] + 0.015,
160                     long_atras*porcentajeg*art6_der['
161                     senialCompleta'][0, i] - 0.03,
162                     -girod*porcentajeg * art1_izq['
163                     senialCompleta'][0, i], 0.6 *
164                     porcentajeg* art2_izq['
165                     senialCompleta'][0, i], long_atras*
166                     porcentajeg*art3_izq['
167                     senialCompleta'][0, i],
168                     long_atras*porcentajeg*art4_izq['
169                     senialCompleta'][0, i], 0.6 *
170                     porcentajeg* art5_izq['
171                     senialCompleta'][0, i] - 0.03,
172                     long_atras*porcentajeg*art6_izq['
173                     senialCompleta'][0, i] - 0.03]
174
175     arm_trajectory.points.append(JointTrajectoryPoint())
176     arm_trajectory.points[ind].positions = piernas_goalF

```

```

154         arm_trajectory.points[ind].velocities = [0.0 for j in
155             piernas_goalF]
156         arm_trajectory.points[ind].accelerations = [0.0 for j
157             in piernas_goalF]
158     # art1_der[ 'tiempo_completo '][0 , i]
159     # 0.01 * (ind + 1)
160     arm_trajectory.points[ind].time_from_start = rospy.
161         Duration(
162             1.1 + (art1_der[ 'tiempo_completo '][0 , int((
163                 art1_der[ 'senalCompleta '].shape[1] - 1)*0.93)
164                 - i]) *0.4)
165     # print 1+art1_der[ 'tiempo_completo '][0 , i]
166     ind += 1
167     # print art1_der[ 'senalCompleta '][0 , i]
168
169
170     reposo = JointTrajectory()
171     reposo.joint_names = piernas_joints
172     piernas_goalF01 = [0,0,0,0,0,0,0,0,0,0,0,0]
173     reposo.points.append(JointTrajectoryPoint())
174     reposo.points[0].positions = piernas_goalF01
175     reposo.points[0].velocities = [0.0 for k in
176         piernas_goalF01]
177     reposo.points[0].accelerations = [0.0 for k in
178         piernas_goalF01]
179     reposo.points[0].time_from_start = rospy.Duration(2)
180
181
182     # Create an empty trajectory goal
183     piernas_goal = FollowJointTrajectoryGoal()
184     piernas_goal_atras = FollowJointTrajectoryGoal()
185     piernas_reposo = FollowJointTrajectoryGoal()
186     # Set the trajectory component to the goal trajectory
187     # created above
188     piernas_goal.trajectory = hacia_delante
189     piernas_goal_atras.trajectory = arm_trajectory
190     piernas_reposo.trajectory = reposo
191     # piernas_goal.trajectory = arm_trajectory
192
193
194     # Specify zero tolerance for the execution time
195     piernas_goal.goal_time_tolerance = rospy.Duration(0.001)

```

```

187     piernas_goal_atras.goal_time_tolerance = rospy.Duration
188         (0.001)
189
190     # Send the goal to the action server
191     arm_client.send_goal(piernas_goal)
192     arm_client.wait_for_result()
193     arm_client.send_goal(piernas_goal_atras)
194     arm_client.wait_for_result()
195     arm_client.send_goal(piernas_goal)
196     arm_client.wait_for_result()
197     arm_client.send_goal(piernas_goal_atras)
198     arm_client.wait_for_result()
199     arm_client.send_goal(piernas_goal)
200     arm_client.wait_for_result()
201     arm_client.send_goal(piernas_goal_atras)
202     arm_client.wait_for_result()
203     arm_client.send_goal(piernas_goal)
204     arm_client.wait_for_result()
205     arm_client.send_goal(piernas_goal_atras)
206     arm_client.wait_for_result()
207
208     # arm_client.send_goal(piernas_goal)
209     print arm_client.get_result()
210     arm_client.send_goal(piernas_reposo)
211
212     rospy.loginfo('... done')
213
214 if __name__ == '__main__':
215     try:
216         TrajectoryDemo()
217     except rospy.ROSInterruptException:
218         pass

```

C.4. Giros

```

1 #!/usr/bin/env python
2
3 import actionlib
4 import rospy

```

```

5
6 from control_msgs.msg import FollowJointTrajectoryAction ,
   FollowJointTrajectoryGoal
7 from trajectory_msgs.msg import JointTrajectory ,
   JointTrajectoryPoint
8 import urdf_parser_py.urdf
9 import urdf_parser_py.xml_reflection
10 import kdl_parser_py.urdf
11 import PyKDL
12 import copy
13 from angles import normalize
14
15 import pickle
16
17 class TrajectoryDemo():
18     def __init__(self):
19
20         pi4 = 0.7853
21
22         filename = '/home/kaiser/WS_ROS/catkin_ws/src/urdf_serp/
   urdf/mi_robot_xacro4_cambio.urdf'
23
24         robot = urdf_parser_py.urdf.URDF.from_xml_file(filename)
25
26         (ok, tree) = kdl_parser_py.urdf.treeFromFile(filename)
27         cadena_der_up_down = tree.getChain("base_link", "
   pie_der_link")
28         cadena_der_down_up = tree.getChain("pie_der_link", "
   base_link")
29         cadena_izq_up_down = tree.getChain("base_link", "
   pie_izq_link")
30         cadena_izq_down_up = tree.getChain("pie_izq_link", "
   base_link")
31
32         print cadena_der_up_down.getNrOfSegments()
33         fksolver_der_up_down = PyKDL.ChainFkSolverPos_recursive(
   cadena_der_up_down)
34         fksolver_der_down_up = PyKDL.ChainFkSolverPos_recursive(
   cadena_der_down_up)
35         fksolver_izq_up_down = PyKDL.ChainFkSolverPos_recursive(
   cadena_izq_up_down)

```

```

36     fksolver_izq_down_up = PyKDL.ChainFkSolverPos_recursive(
37         cadena_izq_down_up)
38
38     vik_der_up_down = PyKDL.ChainIkSolverVel_pinv(
39         cadena_der_up_down)
39     ik_der_up_down = PyKDL.ChainIkSolverPos_NR(
40         cadena_der_up_down, fksolver_der_up_down,
41         vik_der_up_down)
41
41     vik_der_down_up = PyKDL.ChainIkSolverVel_pinv(
42         cadena_der_down_up)
42     ik_der_down_up = PyKDL.ChainIkSolverPos_NR(
43         cadena_der_down_up, fksolver_der_down_up,
44         vik_der_down_up)
44
44     vik_izq_up_down = PyKDL.ChainIkSolverVel_pinv(
45         cadena_izq_up_down)
45     ik_izq_up_down = PyKDL.ChainIkSolverPos_NR(
46         cadena_izq_up_down, fksolver_izq_up_down,
47         vik_izq_up_down)
47
47     vik_izq_down_up = PyKDL.ChainIkSolverVel_pinv(
48         cadena_izq_down_up)
48     ik_izq_down_up = PyKDL.ChainIkSolverPos_NR(
49         cadena_izq_down_up, fksolver_izq_down_up,
50         vik_izq_down_up)
50
50     nj_izq = cadena_der_up_down.getNrOfJoints()
51     posicionInicial_der_up_down = PyKDL.JntArray(nj_izq)
52     posicionInicial_der_up_down[0] = 0
53     posicionInicial_der_up_down[1] = 0.3
54     posicionInicial_der_up_down[2] = -0.3
55     posicionInicial_der_up_down[3] = 0.6
56     posicionInicial_der_up_down[4] = -0.3
57     posicionInicial_der_up_down[5] = -0.3
58
59     nj_izq = cadena_izq_up_down.getNrOfJoints()
60     posicionInicial_izq_up_down = PyKDL.JntArray(nj_izq)
61     posicionInicial_izq_up_down[0] = 0
62     posicionInicial_izq_up_down[1] = 0.3
63     posicionInicial_izq_up_down[2] = -0.3

```

```

64     posicionInicial_izq_up_down [3] = 0.6
65     posicionInicial_izq_up_down [4] = -0.3
66     posicionInicial_izq_up_down [5] = -0.3
67
68     nj_izq = cadena_der_down_up.getNrOfJoints()
69     posicionInicial_der_down_up = PyKDL.JntArray(nj_izq)
70     posicionInicial_der_down_up [5] = 0
71     posicionInicial_der_down_up [4] = 0.3
72     posicionInicial_der_down_up [3] = -0.3
73     posicionInicial_der_down_up [2] = 0.6
74     posicionInicial_der_down_up [1] = -0.3
75     posicionInicial_der_down_up [0] = -0.3
76
77     nj_izq = cadena_izq_down_up.getNrOfJoints()
78     posicionInicial_izq_down_up = PyKDL.JntArray(nj_izq)
79     posicionInicial_izq_down_up [5] = 0
80     posicionInicial_izq_down_up [4] = 0.3
81     posicionInicial_izq_down_up [3] = -0.3
82     posicionInicial_izq_down_up [2] = 0.6
83     posicionInicial_izq_down_up [1] = -0.3
84     posicionInicial_izq_down_up [0] = -0.3
85     print "Forward kinematics"
86     finalFrame_izq_up_down = PyKDL.Frame()
87     finalFrame_izq_down_up = PyKDL.Frame()
88     finalFrame_der_up_down = PyKDL.Frame()
89     finalFrame_der_down_up = PyKDL.Frame()
90
91
92     print "Rotational Matrix of the final Frame: "
93     print finalFrame_izq_up_down.M
94     print "End-effector position: ", finalFrame_izq_up_down.p
95
96     rospy.init_node('trajectory_demo')
97
98     # Set to True to move back to the starting configurations
99     reset = rospy.get_param('~reset', False)
100
101    # Set to False to wait for arm to finish before moving
102        head
103        sync = rospy.get_param('~sync', True)

```

```

104     # Which joints define the arm?
105     piernas_joints = [ 'cylinder_blue_box1_der_joint' , ,
106                         'cylinder_blue1_box2_der_joint' , ,
107                         'cylinder_blue_box2_der_joint' ,
108                         'cylinder_blue_box4_der_joint' , ,
109                         'cylinder_blue1_box6_der_joint' , ,
110                         'cylinder_blue_box6_der_joint' ,
111                         'cylinder_blue_box1_izq_joint' , ,
112                         'cylinder_blue1_box2_izq_joint' , ,
113                         'cylinder_blue_box2_izq_joint' ,
114                         'cylinder_blue_box4_izq_joint' , ,
115                         'cylinder_blue1_box6_izq_joint' , ,
116                         'cylinder_blue_box6_izq_joint' ]
117
117
118     piernas_goal0  = [ 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 ,
119                           0.0 , 0.0 , 0.0 , 0.0 ]
120
121     #1111111111111111111111111111111111111111111111111111111111111111
122     print "Inverse Kinematics"
123     fkssolver_izq_up_down.JntToCart(posicionInicial_izq_up_down
124                                         , finalFrame_izq_up_down)
125     fkssolver_izq_down_up.JntToCart(posicionInicial_izq_down_up
126                                         , finalFrame_izq_down_up)
127     q_init_izq_up_down = posicionInicial_izq_up_down # initial angles
128     desiredFrame = finalFrame_izq_up_down
129     desiredFrame.p[0] = finalFrame_izq_up_down.p[0]
130     desiredFrame.p[1] = finalFrame_izq_up_down.p[1]
131     desiredFrame.p[2] = finalFrame_izq_up_down.p[2]
132     print "Desired Position: ", desiredFrame.p
133     q_out_izq_up_down = PyKDL.JntArray(cadena_izq_up_down.
134                                         getNrOfJoints())
135     ik_izq_up_down.CartToJnt(q_init_izq_up_down, desiredFrame,
136                               q_out_izq_up_down)
137     print "Output angles in rads: ", q_out_izq_up_down
138
139
140
141     print "Inverse Kinematics"
142     fkssolver_der_up_down.JntToCart(posicionInicial_der_up_down
143                                         , finalFrame_der_up_down)

```

```

130     fksolver_der_down_up.JntToCart(posicionInicial_der_down_up
131         , finalFrame_der_down_up)
132     q_init_der_up_down = posicionInicial_der_up_down # initial angles
133     # desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))
134     desiredFrame = finalFrame_der_up_down
135     desiredFrame.p[0] = finalFrame_der_up_down.p[0]
136     desiredFrame.p[1] = finalFrame_der_up_down.p[1]
137     desiredFrame.p[2] = finalFrame_der_up_down.p[2]+0.02 #0.02
138     print "Desired Position: ", desiredFrame.p
139     q_out_der_up_down = PyKDL.JntArray(cadena_der_up_down.
140         getNrOfJoints())
141     ik_der_up_down.CartToJnt(q_init_der_up_down, desiredFrame,
142         q_out_der_up_down)
143     print "Output angles in rads: ", q_out_der_up_down
144
145     piernas_goal0[6] = q_out_izq_up_down[0]
146     piernas_goal0[7] = q_out_izq_up_down[1]
147     piernas_goal0[8] = q_out_izq_up_down[2]
148     piernas_goal0[9] = q_out_izq_up_down[3]
149     piernas_goal0[10] = q_out_izq_up_down[4]
150     piernas_goal0[11] = q_out_izq_up_down[5]
151
152     piernas_goal0[0] = q_out_der_up_down[0]
153     piernas_goal0[1] = q_out_der_up_down[1]
154     piernas_goal0[2] = q_out_der_up_down[2]
155     piernas_goal0[3] = q_out_der_up_down[3]
156     piernas_goal0[4] = q_out_der_up_down[4]
157     piernas_goal0[5] = q_out_der_up_down[5]
158
159     #piernas_goal12 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
160     piernas_goal12 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
161
162     piernas_goal12[6] = q_out_izq_up_down[0]-2*pi4
163     piernas_goal12[7] = q_out_izq_up_down[1]
164     piernas_goal12[8] = q_out_izq_up_down[2]
165     piernas_goal12[9] = q_out_izq_up_down[3]
166     piernas_goal12[10] = q_out_izq_up_down[4]
167     piernas_goal12[11] = q_out_izq_up_down[5]

```

```

166
167      piernas_goal12 [0] = 0
168      piernas_goal12 [1] = 0
169      piernas_goal12 [2] = -0.7
170      piernas_goal12 [3] = 1.4
171      piernas_goal12 [4] = 0
172      piernas_goal12 [5] = -0.7
173
174 ##########
175
176      piernas_goal12 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
177          0.0, 0.0, 0.0]
178
179      piernas_goal12 [6] = q_out_izq_up_down[0]-2*pi4
180      piernas_goal12 [7] = q_out_izq_up_down[1]-0.3-0.3
181      piernas_goal12 [8] = q_out_izq_up_down[2]-0.3
182      piernas_goal12 [9] = q_out_izq_up_down[3]+0.6
183      piernas_goal12 [10] = q_out_izq_up_down[4]+0.3+0.3
184      piernas_goal12 [11] = q_out_izq_up_down[5] -0.3
185
186      piernas_goal12 [0] = 0##-pi4
187      piernas_goal12 [1] = -0.25
188      piernas_goal12 [2] = -0.3
189      piernas_goal12 [3] = 0.6
190      piernas_goal12 [4] = 0.25
191      piernas_goal12 [5] = -0.3
192
193 ##########
194      piernas_goal3 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
195          0.0, 0.0, 0.0]
196      piernas_goal3 [6] = q_out_izq_up_down [0] +0.0
197      piernas_goal3 [7] = q_out_izq_up_down [1] - 0.3
198      piernas_goal3 [8] = q_out_izq_up_down [2] - 0.3
199      piernas_goal3 [9] = q_out_izq_up_down [3] + 0.6
200      piernas_goal3 [10] = q_out_izq_up_down [4] + 0.3
201      piernas_goal3 [11] = q_out_izq_up_down [5] - 0.3
202
203      piernas_goal3 [0] = -2*pi4
204      piernas_goal3 [1] = -0.25
205      piernas_goal3 [2] = -0.3
206      piernas_goal3 [3] = 0.6

```

```

205      piernas_goal3[4] = 0.25
206      piernas_goal3[5] = -0.3
207
208      #####
209      piernas_goal4 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
210          0.0, 0.0, 0.0]
211      piernas_goal4[6] = q_out_izq_up_down[0] +0.0
212      piernas_goal4[7] = q_out_izq_up_down[1] - 0.3 +0.3
213      piernas_goal4[8] = q_out_izq_up_down[2] - 0.3 +0.3
214      piernas_goal4[9] = q_out_izq_up_down[3] + 0.6 -0.6
215      piernas_goal4[10] = q_out_izq_up_down[4] + 0.3 -0.3
216      piernas_goal4[11] = q_out_izq_up_down[5] - 0.3 +0.3
217
218      piernas_goal4[0] = -2*pi4
219      piernas_goal4[1] = -0.25
220      piernas_goal4[2] = -0.7
221      piernas_goal4[3] = 1.4
222      piernas_goal4[4] = 0.25
223      piernas_goal4[5] = -0.7
224
225
226      # Connect to the right arm trajectory action server
227      rospy.loginfo('Waiting for right arm trajectory controller
... ')
228
229      arm_client = actionlib.SimpleActionClient('/piernas/
piernas_controller/follow_joint_trajectory',
FollowJointTrajectoryAction)
230      #/piernas/piernas_controller/follow_joint_trajectory
231      arm_client.wait_for_server()
232
233      rospy.loginfo('... connected.')
234
235
236
237      # Create a single-point arm trajectory with the
piernas_goal as the end-point
238      arm_trajectory = JointTrajectory()
239      arm_trajectory.joint_names = piernas_joints
240      arm_trajectory.points.append(JointTrajectoryPoint())

```

```

241     arm_trajectory.points[0].positions = piernas_goal0
242     arm_trajectory.points[0].velocities = [0.0 for i in
243         piernas_joints]
244     arm_trajectory.points[0].accelerations = [0.0 for i in
245         piernas_joints]
246     arm_trajectory.points[0].time_from_start = rospy.Duration
247         (3.0)
248     arm_trajectory.points.append(JointTrajectoryPoint())
249     arm_trajectory.points[1].positions = piernas_goal12
250     arm_trajectory.points[1].velocities = [0.0 for i in
251         piernas_joints]
252     arm_trajectory.points[1].accelerations = [0.0 for i in
253         piernas_joints]
254     arm_trajectory.points[1].time_from_start = rospy.Duration
255         (6.0)
256     arm_trajectory.points.append(JointTrajectoryPoint())
257     arm_trajectory.points[2].positions = piernas_goal2
258     arm_trajectory.points[2].velocities = [0.0 for i in
259         piernas_joints]
260     arm_trajectory.points[2].accelerations = [0.0 for i in
261         piernas_joints]
262     arm_trajectory.points[2].time_from_start = rospy.Duration
263         (9.0)
264     arm_trajectory.points.append(JointTrajectoryPoint())
265     arm_trajectory.points[3].positions = piernas_goal3
266     arm_trajectory.points[3].velocities = [0.0 for i in

```

```

267     arm_trajectory.points[5].velocities = [0.0 for i in
268         piernas_joints]
269     arm_trajectory.points[5].accelerations = [0.0 for i in
270         piernas_joints]
271     arm_trajectory.points[5].time_from_start = rospy.Duration
272         (18.0)
273     arm_trajectory.points.append(JointTrajectoryPoint())
274     arm_trajectory.points[6].positions = piernas_goal2
275     arm_trajectory.points[6].velocities = [0.0 for i in
276         piernas_joints]
277     arm_trajectory.points[6].accelerations = [0.0 for i in
278         piernas_joints]
279     arm_trajectory.points[6].time_from_start = rospy.Duration
280         (21.0)

281     arm_trajectory.points.append(JointTrajectoryPoint())
282     arm_trajectory.points[7].positions = piernas_goal3
283     arm_trajectory.points[7].velocities = [0.0 for i in
284         piernas_joints]
285     arm_trajectory.points[7].accelerations = [0.0 for i in
286         piernas_joints]
287     arm_trajectory.points[7].time_from_start = rospy.Duration
288         (24)
289     arm_trajectory.points.append(JointTrajectoryPoint())
290     arm_trajectory.points[8].positions = piernas_goal4
291     arm_trajectory.points[8].velocities = [0.0 for i in
292         piernas_joints]
293     arm_trajectory.points[8].accelerations = [0.0 for i in
294         piernas_joints]
295     arm_trajectory.points[8].time_from_start = rospy.Duration
296         (27)
297     arm_trajectory.points.append(JointTrajectoryPoint())
298     arm_trajectory.points[9].positions = piernas_goal12
299     arm_trajectory.points[9].velocities = [0.0 for i in
300         piernas_joints]
301     arm_trajectory.points[9].accelerations = [0.0 for i in
302         piernas_joints]
303     arm_trajectory.points[9].time_from_start = rospy.Duration
304         (30.0)
305     arm_trajectory.points.append(JointTrajectoryPoint())
306     arm_trajectory.points[10].positions = piernas_goal2

```

```

293     arm_trajectory.points[10].velocities = [0.0 for i in
294         piernas_joints]
295     arm_trajectory.points[10].accelerations = [0.0 for i in
296         piernas_joints]
297     arm_trajectory.points[10].time_from_start = rospy.Duration
298         (33.0)
299     arm_trajectory.points.append(JointTrajectoryPoint())
300     arm_trajectory.points[11].positions = piernas_goal3
301     arm_trajectory.points[11].velocities = [0.0 for i in
302         piernas_joints]
303     arm_trajectory.points[11].accelerations = [0.0 for i in
304         piernas_joints]
305     arm_trajectory.points[11].time_from_start = rospy.Duration
306         (36.0)
307     arm_trajectory.points.append(JointTrajectoryPoint())
308     arm_trajectory.points[12].positions = piernas_goal4
309     arm_trajectory.points[12].velocities = [0.0 for i in
310         piernas_joints]
311     arm_trajectory.points[12].accelerations = [0.0 for i in
312         piernas_joints]
313     arm_trajectory.points[12].time_from_start = rospy.Duration
314         (39.0)
315     arm_trajectory.points.append(JointTrajectoryPoint())
316     arm_trajectory.points[13].positions = piernas_goal12
317     arm_trajectory.points[13].velocities = [0.0 for i in
318         piernas_joints]
319     arm_trajectory.points[13].accelerations = [0.0 for i in
320         piernas_joints]
321     arm_trajectory.points[13].time_from_start = rospy.Duration
322         (42.0)
323     '''arm_trajectory.points.append(JointTrajectoryPoint())
324     arm_trajectory.points[14].positions = piernas_goal4
325     arm_trajectory.points[14].velocities = [0.0 for i in
326         piernas_joints]
327     arm_trajectory.points[14].accelerations = [0.0 for i in
328         piernas_joints]
329     arm_trajectory.points[14].time_from_start = rospy.Duration
330         (32.0)'''
331     # Send the trajectory to the arm action server
332     rospy.loginfo('Moving the arm to goal position... ')
333

```

```
319
320
321     # Create an empty trajectory goal
322     piernas_goal = FollowJointTrajectoryGoal()
323
324     # Set the trajectory component to the goal trajectory
325     # created above
326     piernas_goal.trajectory = arm_trajectory
327
328     # Specify zero tolerance for the execution time
329     piernas_goal.goal_time_tolerance = rospy.Duration(0.01)
330
331     # Send the goal to the action server
332     arm_client.send_goal(piernas_goal)
333
334     if not sync:
335         # Wait for up to 5 seconds for the motion to complete
336         arm_client.wait_for_result(rospy.Duration(5.0))
337
338         arm_client.wait_for_result()
339         print arm_client.get_result()
340
341         rospy.loginfo('... done')
342
343 if __name__ == '__main__':
344     try:
345         TrajectoryDemo()
346     except rospy.ROSInterruptException:
347         pass
```

C.5. Reposo

```
1 #!/usr/bin/env python
2
3 import actionlib
4 import rospy
5
6 from control_msgs.msg import FollowJointTrajectoryAction,
    FollowJointTrajectoryGoal
```

```

7 from trajectory_msgs.msg import JointTrajectory ,
   JointTrajectoryPoint
8 import urdf_parser_py.urdf
9 import urdf_parser_py.xml_reflection
10 import kdl_parser_py.urdf
11 import PyKDL
12
13 class TrajectoryDemo():
14     def __init__(self):
15         filename = '/home/kaiser/WS_ROS/catkin_ws/src/urdf_serp/
16                         urdf/urdf_exportado3.urdf'
17         robot = urdf_parser_py.urdf.URDF.from_xml_file(filename)
18
19         (ok, tree) = kdl_parser_py.urdf.treeFromFile(filename)
20         cadena_der_up_down = tree.getChain("base_link", "pie_der_link")
21         cadena_der_down_up = tree.getChain("pie_der_link", "base_link")
22         cadena_izq_up_down = tree.getChain("base_link", "pie_izq_link")
23         cadena_izq_down_up = tree.getChain("pie_izq_link", "base_link")
24
25         print cadena_der_up_down.getNrOfSegments()
26         fksolver_der_up_down = PyKDL.ChainFkSolverPos_recursive(
27             cadena_der_up_down)
28         fksolver_der_down_up = PyKDL.ChainFkSolverPos_recursive(
29             cadena_der_down_up)
30         fksolver_izq_up_down = PyKDL.ChainFkSolverPos_recursive(
31             cadena_izq_up_down)
32         fksolver_izq_down_up = PyKDL.ChainFkSolverPos_recursive(
33             cadena_izq_down_up)
34
35         vik_der_up_down = PyKDL.ChainIkSolverVel_pinv(
36             cadena_der_up_down)
37         ik_der_up_down = PyKDL.ChainIkSolverPos_NR(
38             cadena_der_up_down, fksolver_der_up_down,
39             vik_der_up_down)
40
41         vik_der_down_up = PyKDL.ChainIkSolverVel_pinv(
42             cadena_der_down_up)

```

```

34     ik_der_down_up = PyKDL.ChainIkSolverPos_NR(
35         cadena_der_down_up, fksolver_der_down_up,
36         vik_der_down_up)
37
38     vik_izq_up_down = PyKDL.ChainIkSolverVel_pinv(
39         cadena_izq_up_down)
40     ik_izq_up_down = PyKDL.ChainIkSolverPos_NR(
41         cadena_izq_up_down, fksolver_izq_up_down,
42         vik_izq_up_down)
43
44     vik_izq_down_up = PyKDL.ChainIkSolverVel_pinv(
45         cadena_izq_down_up)
46     ik_izq_down_up = PyKDL.ChainIkSolverPos_NR(
47         cadena_izq_down_up, fksolver_izq_down_up,
48         vik_izq_down_up)
49
50
51     nj_izq = cadena_der_up_down.getNrOfJoints()
52     posicionInicial_der_up_down = PyKDL.JntArray(nj_izq)
53     posicionInicial_der_up_down[0] = 0.3
54     posicionInicial_der_up_down[1] = -0.3
55     posicionInicial_der_up_down[2] = 0
56     posicionInicial_der_up_down[3] = 0.6
57     posicionInicial_der_up_down[4] = -0.3
58     posicionInicial_der_up_down[5] = -0.3
59
60     nj_izq = cadena_izq_up_down.getNrOfJoints()
61     posicionInicial_izq_up_down = PyKDL.JntArray(nj_izq)
62     posicionInicial_izq_up_down[0] = 0.3
63     posicionInicial_izq_up_down[1] = -0.3
64     posicionInicial_izq_up_down[2] = 0
65     posicionInicial_izq_up_down[3] = 0.6
66     posicionInicial_izq_up_down[4] = -0.3

```

```

67     posicionInicial_der_down_up[0] = -0.3
68
69     nj_izq = cadena_izq_down_up.getNrOfJoints()
70     posicionInicial_izq_down_up = PyKDL.JntArray(nj_izq)
71     posicionInicial_izq_down_up[5] = 0.3
72     posicionInicial_izq_down_up[4] = -0.3
73     posicionInicial_izq_down_up[3] = 0
74     posicionInicial_izq_down_up[2] = 0.6
75     posicionInicial_izq_down_up[1] = -0.3
76     posicionInicial_izq_down_up[0] = -0.3
77     print "Forward kinematics"
78     finalFrame_izq_up_down = PyKDL.Frame()
79     finalFrame_izq_down_up = PyKDL.Frame()
80     finalFrame_der_up_down = PyKDL.Frame()
81     finalFrame_der_down_up = PyKDL.Frame()
82
83     fkSolver_izq_up_down.JntToCart(posicionInicial_izq_up_down
84                                     , finalFrame_izq_up_down)
85     print "Rotational Matrix of the final Frame: "
86     print finalFrame_izq_up_down.M
87     print "End-effector position: ", finalFrame_izq_up_down.p
88
89
90     # Set to True to move back to the starting configurations
91     reset = rospy.get_param('~reset', False)
92
93     # Set to False to wait for arm to finish before moving
94     head
95     sync = rospy.get_param('~sync', True)
96
97     # Which joints define the arm?
98     piernas_joints = [ 'cilinder_blue1_box1_der_joint' ,
99                         'cilinder_blue_box1_der_joint' ,
                         'cilinder_blue_box2_der_joint' ,
                         'cilinder_blue_box4_der_joint' ,
                         'cilinder_blue1_box6_der_joint' ,
                         'cilinder_blue_box6_der_joint' ,
                         'cilinder_blue1_box1_izq_joint' ,
                         'cilinder_blue_box1_izq_joint' ,
                         'cilinder_blue_box2_izq_joint' ,

```

```

100          'cylinder_blue_box4_izq_joint' , ,
101          'cylinder_blue1_box6_izq_joint' , ,
102          'cylinder_blue_box6_izq_joint' ]
103
104      if reset:
105          # Set the arm back to the resting position
106          arm_goal = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
107
108      else:
109          # Set a goal configuration for the arm
110          arm_goal = [0.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
111
112          print "Inverse Kinematics"
113          q_init = posicionInicial_izq_up_down # initial angles
114          #desiredFrame = PyKDL.Frame(PyKDL.Vector(0.5, 0.4, 1))
115          desiredFrame = finalFrame_izq_up_down
116          desiredFrame.p[0] = finalFrame_izq_up_down.p[0]
117          desiredFrame.p[1] = finalFrame_izq_up_down.p[1]
118          desiredFrame.p[2] = finalFrame_izq_up_down.p[2]
119          print "Desired Position: ", desiredFrame.p
120          q_out = PyKDL.JntArray(6)
121          ik_izq_up_down.CartToJnt(q_init, desiredFrame, q_out)
122          print "Output angles in rads: ", q_out
123
124          #arm_goal[0] = q_out[0]
125          #arm_goal[1] = q_out[1]
126          #arm_goal[2] = q_out[2]
127          #arm_goal[3] = q_out[3]
128          #arm_goal[4] = q_out[4]
129          #arm_goal[5] = q_out[5]
130
131
132          # Connect to the right arm trajectory action server
133          rospy.loginfo('Waiting for right arm trajectory controller
... ')
134
135          arm_client = actionlib.SimpleActionClient('/piernas/
piernas_controller/follow_joint_trajectory',
FollowJointTrajectoryAction)
arm_client.wait_for_server()

```

```

136
137         rospy.loginfo('... connected.')
138
139
140     # Create a single-point arm trajectory with the arm_goal
141     # as the end-point
142     arm_trajectory = JointTrajectory()
143     arm_trajectory.joint_names = piernas_joints
144     arm_trajectory.points.append(JointTrajectoryPoint())
145     arm_trajectory.points[0].positions = arm_goal
146     arm_trajectory.points[0].velocities = [0.0 for i in
147         piernas_joints]
148     arm_trajectory.points[0].accelerations = [0.0 for i in
149         piernas_joints]
150     arm_trajectory.points[0].time_from_start = rospy.Duration
151         (2.0)
152
153     # Send the trajectory to the arm action server
154     rospy.loginfo('Moving the arm to goal position...')
155
156     # Create an empty trajectory goal
157     arm_goal = FollowJointTrajectoryGoal()
158
159     # Set the trajectory component to the goal trajectory
160     # created above
161     arm_goal.trajectory = arm_trajectory
162
163     # Specify zero tolerance for the execution time
164     arm_goal.goal_time_tolerance = rospy.Duration(0.0)
165
166     # Send the goal to the action server
167     arm_client.send_goal(arm_goal)
168
169     if not sync:
170         # Wait for up to 5 seconds for the motion to complete
171         arm_client.wait_for_result(rospy.Duration(5.0))
172
173
174     rospy.loginfo('... done')
175
176
177 if __name__ == '__main__':

```

```
172     try :
173         TrajectoryDemo()
174     except rospy.ROSInterruptException :
175         pass
```

C.6. Envío de comandos por I2C a los módulos de las articulaciones

```
1 #!/usr/bin/env python
2
3 import sys
4 sys.path.append('/root/mraa/build/src/python')
5 import mraa
6 import math
7 import ctypes
8 import time
9
10 x = mraa.I2c(0)
11
12 class CONTROLADOR_PIERNAS():
13
14     def __init__(self):
15
16         self.direcciones = [0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0
17                           x09, 0xa, 0xb, 0xc, 0xd, 0xe]
18         self.posiciones = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
19
20         self.lmites_motor_1_d = [0, 1020]
21         self.lmites_motor_2_d = [282, 0]
22         self.lmites_motor_3_d = [450, 900]
23         self.lmites_motor_4_d = [480, 760]
24         self.lmites_motor_5_d = [350, 670]
25         self.lmites_motor_6_d = [200, 600]
26
27         self.lmites_motor_1_i = [0, 1020]
28         self.lmites_motor_2_i = [282, 0]
29         self.lmites_motor_3_i = [450, 900]
30         self.lmites_motor_4_i = [480, 760]
31         self.lmites_motor_5_i = [350, 670]
```

```
31     self.lmites_motor_6_i = [200, 600]
32
33     self.archivo_1 = open('datos/zero.txt', 'r')
34     self.archivo_2 = open('datos/zero.txt', 'r')
35     self.archivo_3 = open('datos/motor_3_i.txt', 'r')
36     self.archivo_4 = open('datos/motor_4_i.txt', 'r')
37     self.archivo_5 = open('datos/zero.txt', 'r')
38     self.archivo_6 = open('datos/motor_6_i.txt', 'r')
39
40     self.datos_archivo_1 = []
41     self.datos_archivo_2 = []
42     self.datos_archivo_3 = []
43     self.datos_archivo_4 = []
44     self.datos_archivo_5 = []
45     self.datos_archivo_6 = []
46
47     for linea in self.archivo_1:
48
49         self.datos_archivo_1.append(int(linea.split("\n", 1)
50                                         [0]))
51
52     for linea in self.archivo_2:
53
54         self.datos_archivo_2.append(int(linea.split("\n", 1)
55                                         [0]))
56
57     for linea in self.archivo_3:
58
59         self.datos_archivo_3.append(int(linea.split("\n", 1)
60                                         [0]))
61
62     for linea in self.archivo_4:
63
64         self.datos_archivo_4.append(int(linea.split("\n", 1)
65                                         [0]))
66
```

```

67         for linea in self.archivo_6:
68
69             self.datos_archivo_6.append( int( linea.split( "\n" , 1)
70                                         [0]) )
71
72             self.archivo_1.close()
73             self.archivo_2.close()
74             self.archivo_3.close()
75             self.archivo_4.close()
76             self.archivo_5.close()
77             self.archivo_6.close()
78
78             self.archivo_7 = open( 'datos/zero.txt' , 'r' )
79             self.archivo_8 = open( 'datos/zero.txt' , 'r' )
80             self.archivo_9 = open( 'datos/motor_3_d.txt' , 'r' )
81             self.archivo_10 = open( 'datos/motor_4_d.txt' , 'r' )
82             self.archivo_11 = open( 'datos/zero.txt' , 'r' )
83             self.archivo_12 = open( 'datos/motor_6_d.txt' , 'r' )
84
85             self.datos_archivo_7 = []
86             self.datos_archivo_8 = []
87             self.datos_archivo_9 = []
88             self.datos_archivo_10 = []
89             self.datos_archivo_11 = []
90             self.datos_archivo_12 = []
91
92         for linea in self.archivo_7:
93
94             self.datos_archivo_7.append( int( linea.split( "\n" , 1)
95                                         [0]) )
96
96         for linea in self.archivo_8:
97
98             self.datos_archivo_8.append( int( linea.split( "\n" , 1)
99                                         [0]) )
100
100         for linea in self.archivo_9:
101
102             self.datos_archivo_9.append( int( linea.split( "\n" , 1)
103                                         [0]) )

```

```
104         for linea in self.archivo_10:
105
106             self.datos_archivo_10.append(int(linea.split("\n", 1)
107                                         [0]))
108
109             for linea in self.archivo_11:
110
111                 self.datos_archivo_11.append(int(linea.split("\n", 1)
112                                         [0]))
113
114                 for linea in self.archivo_12:
115
116                     self.datos_archivo_12.append(int(linea.split("\n", 1)
117                                         [0]))
118
119                     self.archivo_7.close()
120                     self.archivo_8.close()
121                     self.archivo_9.close()
122                     self.archivo_10.close()
123                     self.archivo_11.close()
124                     self.archivo_12.close()
125
126
127             def obtener_posicion(self):
128
129                 for j in range(len(self.direcciones)):
130
131                     x.address(self.direcciones[j])
132                     x.writeByte(184)
133
134                     print "El motor en la posicion " + str(self.
135                         direcciones[j]) + " esta libre"
136
137                     self.posiciones[j] = 0
138
139                     raw_input()
140
141
142                     i = 0
143
144                     while i <= 3:
```

```
141             a = x.readByte()
142
143             self.posiciones[j] = self.posiciones[j] + a
144
145             i = i + 1
146
147             i = 0
148
149             return self.posiciones
150
151     def mandar_posicion(self, angulo, motor):
152
153         resto = angulo
154         vector_posicion = [0, 0, 0, 0]
155
156         x.address(self.direcciones[motor])
157         x.writeByte(182)
158
159         i = 0
160
161         while resto > 255:
162             resto = resto - 255
163             vector_posicion[i] = 255
164             i = i + 1
165
166             vector_posicion[i] = resto
167
168             i = 0
169
170     print vector_posicion
171
172     for j in range(len(vector_posicion)):
173
174         x.writeByte(vector_posicion[j])
175
176         vector_posicion = [0, 0, 0, 0]
177
178
179 if __name__ == '__main__':
180
181     control_1 = CONTROLADOR_PIERNAS()
```

```
182
183     while 1:
184
185         print "0: Recibir posicion servo\n1: Mandar posicion servo
186             \n\nIntroduce la orden: "
187
188         var = raw_input()
189
190         if int(var) == 0:
191
192             print control_1.obtener_posicion()
193
194         if int(var) == 1:
195
196             for j in range(0, len(control_1.datos_archivo_1), 1):
197
198                 espera = 0.02
199                 espera2 = 0
200                 print '\n'
201
202                 control_1.mandar_posicion(control_1.
203                     datos_archivo_1[j], 0)
204                 time.sleep(espera)
205                 control_1.mandar_posicion(control_1.
206                     datos_archivo_1[j], 1)
207                 time.sleep(espera)
208                 control_1.mandar_posicion(control_1.
209                     datos_archivo_1[j], 2)
210                 time.sleep(espera)
211                 control_1.mandar_posicion(control_1.datos_archivo_1[j], 3)
212                 time.sleep(espera)
213                 control_1.mandar_posicion(control_1.datos_archivo_1[j], 4)
214                 time.sleep(espera)
215                 control_1.mandar_posicion(control_1.datos_archivo_1[j], 5)
216                 time.sleep(espera)
217
218
219                 control_1.mandar_posicion(control_1.
220                     datos_archivo_1[j], 6)
221                 time.sleep(espera)
222                 control_1.mandar_posicion(control_1.datos_archivo_1[j], 7)
223                 time.sleep(espera)
```

```
218     control_1.mandar_posicion(control_1.datos_archivo_1[j], 8)
219             time.sleep(espera)
220     control_1.mandar_posicion(control_1.datos_archivo_1[j], 9)
221             time.sleep(espera)
222     control_1.mandar_posicion(control_1.datos_archivo_1[j], 10)
223             time.sleep(espera)
224     control_1.mandar_posicion(control_1.datos_archivo_1[j], 11)
225             time.sleep(espera)
226
227     time.sleep(espera2)
```

D. Código en ROS del Nodo Servidor para el control de los movimientos de robots hexá-podos

D.1. Caminada adelante/atras

```
1 #!/usr/bin/env python
2
3 import actionlib
4 import rospy
5
6 from control_msgs.msg import FollowJointTrajectoryAction,
7     FollowJointTrajectoryGoal
8 from trajectory_msgs.msg import JointTrajectory,
9     JointTrajectoryPoint
10 import urdf_parser_py.urdf
11 import urdf_parser_py.xml_reflection
12 import kdl_parser_py.urdf
13 import PyKDL
14 import hrpsys.ModelLoader_idl
15 import copy
16 from angles import normalize
17 import pickle
18
19 class TrajectoryDemo():
20     def __init__(self):
```

```

19
20     filename = '/home/kaiser/WS_ROS/catkin_ws/src/urdf_serp/
21             urdf/mi_hexapodo_exp.urdf'
22
23     angulo_avance = +0.20 #rad
24     altura_pata = +0.03 #cm
25     # angulo_avance = 0 # +0.40 #rad
26     # altura_pata = 0 # -0.04 #cm
27
28
29     (ok, tree) = kdl_parser_py.urdf.treeFromFile(filename)
30     cadena_RR = tree.getChain("base_link", "tarsus_RR")
31     cadena_RM = tree.getChain("base_link", "tarsus_RM")
32     cadena_RF = tree.getChain("base_link", "tarsus_RF")
33     cadena_LR = tree.getChain("base_link", "tarsus_LR")
34     cadena_LM = tree.getChain("base_link", "tarsus_LM")
35     cadena_LF = tree.getChain("base_link", "tarsus_LF")
36
37     print cadena_RR.getNrOfSegments()
38     fksolver_RR = PyKDL.ChainFkSolverPos_recursive(cadena_RR)
39     fksolver_RM= PyKDL.ChainFkSolverPos_recursive(cadena_RM)
40     fksolver_RF = PyKDL.ChainFkSolverPos_recursive(cadena_RF)
41     fksolver_LR = PyKDL.ChainFkSolverPos_recursive(cadena_LR)
42     fksolver_LM = PyKDL.ChainFkSolverPos_recursive(cadena_LM)
43     fksolver_LF = PyKDL.ChainFkSolverPos_recursive(cadena_LF)
44
45     vik_RR = PyKDL.ChainIkSolverVel_pinv(cadena_RR)
46     ik_RR = PyKDL.ChainIkSolverPos_NR(cadena_RR, fksolver_RR,
47                                         vik_RR)
48
49     vik_RM = PyKDL.ChainIkSolverVel_pinv(cadena_RM)
50     ik_RM = PyKDL.ChainIkSolverPos_NR(cadena_RM, fksolver_RM,
51                                         vik_RM)
52
53     vik_RF = PyKDL.ChainIkSolverVel_pinv(cadena_RF)
54     ik_RF = PyKDL.ChainIkSolverPos_NR(cadena_RF, fksolver_RF,
55                                         vik_RF)
56
57     vik_LR = PyKDL.ChainIkSolverVel_pinv(cadena_LR)

```

```

55     ik_LR = PyKDL.ChainIkSolverPos_NR(cadena_LR, fksolver_LR,
56                                         vik_LR)
57
58     vik_LM = PyKDL.ChainIkSolverVel_pinv(cadena_LM)
59     ik_LM = PyKDL.ChainIkSolverPos_NR(cadena_LM, fksolver_LM,
60                                         vik_LM)
61
62
63     vik_LF = PyKDL.ChainIkSolverVel_pinv(cadena_LF)
64     ik_LF = PyKDL.ChainIkSolverPos_NR(cadena_LF, fksolver_LF,
65                                         vik_LF)
66
67     nj_izq = cadena_RR.getNrOfJoints()
68     posicionInicial_R = PyKDL.JntArray(nj_izq)
69     posicionInicial_R[0] = 0
70     posicionInicial_R[1] = 0
71     posicionInicial_R[2] = 0
72     posicionInicial_R[3] = 0
73
74     nj_izq = cadena_LR.getNrOfJoints()
75     posicionInicial_L = PyKDL.JntArray(nj_izq)
76     posicionInicial_L[0] = 0
77     posicionInicial_L[1] = 0
78     posicionInicial_L[2] = 0
79     posicionInicial_L[3] = 0
80
81     print "Forward kinematics"
82     finalFrame_R = PyKDL.Frame()
83     finalFrame_L = PyKDL.Frame()
84
85
86     # Set to True to move back to the starting configurations
87     reset = rospy.get_param('~reset', False)
88
89     # Set to False to wait for arm to finish before moving
90     # head
91     sync = rospy.get_param('~sync', True)
92
93     # Which joints define the arm?
94     piernas_joints_RR = ['coxa_joint_RR', 'femur_joint_RR',
95                          'tibia_joint_RR', 'tarsus_joint_RR']

```

```
91     piernas_joints_LM = [ 'coxa_joint_LM' , 'femur_joint_LM' , ,
92                             'tibia_joint_LM' , 'tarsus_joint_LM' ]
93
94     piernas_joints_RF = [ 'coxa_joint_RF' , 'femur_joint_RF' , ,
95                           'tibia_joint_RF' , 'tarsus_joint_RF' ]
96
97
98     piernas_joints_LR = [ 'coxa_joint_LR' , 'femur_joint_LR' , ,
99                             'tibia_joint_LR' , 'tarsus_joint_LR' ]
100    piernas_joints_RM = [ 'coxa_joint_RM' , 'femur_joint_RM' , ,
101                            'tibia_joint_RM' , 'tarsus_joint_RM' ]
102    piernas_joints_LF = [ 'coxa_joint_LF' , 'femur_joint_LF' , ,
103                          'tibia_joint_LF' , 'tarsus_joint_LF' ]
104
105    rr_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
106    lm_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
107    rf_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
108    rr_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
109    lm_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
110    rf_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
111    lr_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
112    rm_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
113    lf_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
114    lr_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
115    rm_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
116    lf_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
117
118 #1111111111111111111111111111111111111111111111111111
119 print "Inverse Kinematics"
120 fkSolver_RR.JntToCart(posicionInicial_R , finalFrame_R)
121 q_init_RR = posicionInicial_R # initial angles
122 desiredFrameRR = finalFrame_R
123 desiredFrameRR.p[0] = finalFrame_R.p[0]
124 desiredFrameRR.p[1] = finalFrame_R.p[1]
125 desiredFrameRR.p[2] = finalFrame_R.p[2]+altura_pata
126 print "Desired Position: " , desiredFrameRR.p
127 q_out_RR = PyKDL.JntArray(cadena_RR.getNrOfJoints())
128 ik_RR.CartToJnt(q_init_RR , desiredFrameRR , q_out_RR)
129 print "Output angles in rads: " , q_out_RR
130
131 rr_goal0[0] = q_out_RR[0]+angulo_avance
132 rr_goal0[1] = q_out_RR[1]
133 rr_goal0[2] = q_out_RR[2]
```

```
127     rr_goal0 [3] = q_out_RR[3]
128
129     print "Inverse Kinematics"
130     fkssolver_LM.JntToCart( posicionInicial_L , finalFrame_L )
131     q_init_LM = posicionInicial_L # initial angles
132     desiredFrameLM = finalFrame_L
133     desiredFrameLM.p[0] = finalFrame_L.p[0]
134     desiredFrameLM.p[1] = finalFrame_L.p[1]
135     desiredFrameLM.p[2] = finalFrame_L.p[2] + altura_pata
136     print "Desired Position: ", desiredFrameLM.p
137     q_out_LM = PyKDL.JntArray(cadena_LM.getNrOfJoints())
138     ik_LM.CartToJnt(q_init_LM, desiredFrameLM, q_out_LM)
139     print "Output angles in rads: ", q_out_LM
140
141
142     lm_goal0 [0] = q_out_LM[0]+angulo_avance
143     lm_goal0 [1] = q_out_LM[1]
144     lm_goal0 [2] = q_out_LM[2]
145     lm_goal0 [3] = q_out_LM[3]
146
147     print "Inverse Kinematics"
148     fkssolver_RF.JntToCart( posicionInicial_R , finalFrame_R )
149     q_init_RF = posicionInicial_R # initial angles
150     desiredFrameRF = finalFrame_R
151     desiredFrameRF.p[0] = finalFrame_R.p[0]
152     desiredFrameRF.p[1] = finalFrame_R.p[1]
153     desiredFrameRF.p[2] = finalFrame_R.p[2] + altura_pata
154     print "Desired Position: ", desiredFrameRF.p
155     q_out_RF = PyKDL.JntArray(cadena_RF.getNrOfJoints())
156     ik_RF.CartToJnt(q_init_RF, desiredFrameRF, q_out_RF)
157     print "Output angles in rads: ", q_out_RF
158
159     rf_goal0 [0] = q_out_RF[0]+angulo_avance
160     rf_goal0 [1] = q_out_RF[1]
161     rf_goal0 [2] = q_out_RF[2]
162     rf_goal0 [3] = q_out_RF[3]
163
164     # 22222222222222222222222222222222222222222222222222222
165     RR_actual = PyKDL.JntArray( nj_izq )
166     RR_actual[0] = rr_goal0[0]
167     RR_actual[1] = rr_goal0[1]
168     RR_actual[2] = rr_goal0[2]
```

```

168     RR_actual[3] = rr_goal0[3]
169     print "Inverse Kinematics"
170     fkSolver_RR.JntToCart(RR_actual, finalFrame_R)
171     q_init_RR = RR_actual # initial angles
172     desiredFrameRR = finalFrame_R
173     desiredFrameRR.p[0] = finalFrame_R.p[0]
174     desiredFrameRR.p[1] = finalFrame_R.p[1]
175     desiredFrameRR.p[2] = finalFrame_R.p[2] - altura_pata
176     print "Desired Position: ", desiredFrameRR.p
177     q_out_RR = PyKDL.JntArray(cadena_RR.getNrOfJoints())
178     ik_RR.CartToJnt(q_init_RR, desiredFrameRR, q_out_RR)
179     print "Output angles in rads: ", q_out_RR
180
181     rr_goal1[0] = q_out_RR[0]
182     rr_goal1[1] = q_out_RR[1]
183     rr_goal1[2] = q_out_RR[2]
184     rr_goal1[3] = q_out_RR[3]
185
186     LM_actual = PyKDL.JntArray(nj_izq)
187     LM_actual[0] = lm_goal0[0]
188     LM_actual[1] = lm_goal0[1]
189     LM_actual[2] = lm_goal0[2]
190     LM_actual[3] = lm_goal0[3]
191     print "Inverse Kinematics"
192     fkSolver_LM.JntToCart(LM_actual, finalFrame_L)
193     q_init_LM = LM_actual # initial angles
194     desiredFrameLM = finalFrame_L
195     desiredFrameLM.p[0] = finalFrame_L.p[0]
196     desiredFrameLM.p[1] = finalFrame_L.p[1]
197     desiredFrameLM.p[2] = finalFrame_L.p[2] - altura_pata
198     print "Desired Position: ", desiredFrameLM.p
199     q_out_LM = PyKDL.JntArray(cadena_LM.getNrOfJoints())
200     ik_LM.CartToJnt(q_init_LM, desiredFrameLM, q_out_LM)
201     print "Output angles in rads: ", q_out_LM
202
203     lm_goal1[0] = q_out_LM[0]
204     lm_goal1[1] = q_out_LM[1]
205     lm_goal1[2] = q_out_LM[2]
206     lm_goal1[3] = q_out_LM[3]
207
208     RF_actual = PyKDL.JntArray(nj_izq)

```

```

209      RF_actual[0] = rf_goal0[0]
210      RF_actual[1] = rf_goal0[1]
211      RF_actual[2] = rf_goal0[2]
212      RF_actual[3] = rf_goal0[3]
213      print "Inverse Kinematics"
214      fkSolver_RF.JntToCart(RF_actual, finalFrame_R)
215      q_init_RF = RF_actual # initial angles
216      desiredFrameRF = finalFrame_R
217      desiredFrameRF.p[0] = finalFrame_R.p[0]
218      desiredFrameRF.p[1] = finalFrame_R.p[1]
219      desiredFrameRF.p[2] = finalFrame_R.p[2] - altura_pata
220      print "Desired Position: ", desiredFrameRF.p
221      q_out_RF = PyKDL.JntArray(cadena_RF.getNrOfJoints())
222      ik_RF.CartToJnt(q_init_RF, desiredFrameRF, q_out_RF)
223      print "Output angles in rads: ", q_out_RF
224
225      rf_goal1[0] = q_out_RF[0]
226      rf_goal1[1] = q_out_RF[1]
227      rf_goal1[2] = q_out_RF[2]
228      rf_goal1[3] = q_out_RF[3]
229
230
231      # 111111111111111111111111111111111111111111111111111111111111111111
232      print "Inverse Kinematics"
233      fkSolver_LR.JntToCart(posicionInicial_L, finalFrame_L)
234      q_init_LR = posicionInicial_L # initial angles
235      desiredFrameLR = finalFrame_L
236      desiredFrameLR.p[0] = finalFrame_L.p[0]
237      desiredFrameLR.p[1] = finalFrame_L.p[1]
238      desiredFrameLR.p[2] = finalFrame_L.p[2] # + altura_pata
239      print "Desired Position: ", desiredFrameLR.p
240      q_out_LR = PyKDL.JntArray(cadena_LR.getNrOfJoints())
241      ik_LR.CartToJnt(q_init_LR, desiredFrameLR, q_out_LR)
242      print "Output angles in rads: ", q_out_LR
243
244      lr_goal0[0] = q_out_LR[0] - angulo_avance
245      lr_goal0[1] = q_out_LR[1]
246      lr_goal0[2] = q_out_LR[2]
247      lr_goal0[3] = q_out_LR[3]
248
249      print "Inverse Kinematics"

```

```

250     fksolver_RM.JntToCart(posicionInicial_R, finalFrame_R)
251     q_init_RM = posicionInicial_R # initial angles
252     desiredFrameRM = finalFrame_R
253     desiredFrameRM.p[0] = finalFrame_R.p[0]
254     desiredFrameRM.p[1] = finalFrame_R.p[1]
255     desiredFrameRM.p[2] = finalFrame_R.p[2] # + altura_pata
256     print "Desired Position: ", desiredFrameRM.p
257     q_out_RM = PyKDL.JntArray(cadena_RM.getNrOfJoints())
258     ik_RM.CartToJnt(q_init_RM, desiredFrameRM, q_out_RM)
259     print "Output angles in rads: ", q_out_RM
260
261
262
263
264
265
266     print "Inverse Kinematics"
267     fksolver_LF.JntToCart(posicionInicial_L, finalFrame_L)
268     q_init_LF = posicionInicial_L # initial angles
269     desiredFrameLF = finalFrame_L
270     desiredFrameLF.p[0] = finalFrame_L.p[0]
271     desiredFrameLF.p[1] = finalFrame_L.p[1]
272     desiredFrameLF.p[2] = finalFrame_L.p[2] # + altura_pata
273     print "Desired Position: ", desiredFrameLF.p
274     q_out_LF = PyKDL.JntArray(cadena_LF.getNrOfJoints())
275     ik_LF.CartToJnt(q_init_LF, desiredFrameLF, q_out_LF)
276     print "Output angles in rads: ", q_out_LF
277
278
279
280
281
282     # 222222222222222222222222222222222222222222222222222
283
284
285     LR_actual = PyKDL.JntArray(nj_izq)
286     LR_actual[0] = lr_goal0[0]
287     LR_actual[1] = lr_goal0[1]
288     LR_actual[2] = lr_goal0[2]
289     LR_actual[3] = lr_goal0[3]
290     print "Inverse Kinematics"

```

```

291     fksolver_LR.JntToCart(LR_actual, finalFrame_L)
292     q_init_LR = LR_actual # initial angles
293     print "Inverse Kinematics"
294     desiredFrameLR = finalFrame_L
295     desiredFrameLR.p[0] = finalFrame_L.p[0]
296     desiredFrameLR.p[1] = finalFrame_L.p[1]
297     desiredFrameLR.p[2] = finalFrame_L.p[2] + altura_pata
298     print "Desired Position: ", desiredFrameLR.p
299     q_out_LR = PyKDL.JntArray(cadena_LR.getNrOfJoints())
300     ik_LR.CartToJnt(q_init_LR, desiredFrameLR, q_out_LR)
301     print "Output angles in rads: ", q_out_LR
302
303     lr_goal1[0] = q_out_LR[0] #= angulo_avance
304     lr_goal1[1] = q_out_LR[1]
305     lr_goal1[2] = q_out_LR[2]
306     lr_goal1[3] = q_out_LR[3]
307
308     RM_actual = PyKDL.JntArray(nj_izq)
309     RM_actual[0] = rm_goal0[0]
310     RM_actual[1] = rm_goal0[1]
311     RM_actual[2] = rm_goal0[2]
312     RM_actual[3] = rm_goal0[3]
313     print "Inverse Kinematics"
314     fksolver_RM.JntToCart(RM_actual, finalFrame_R)
315     q_init_RM = RM_actual # initial angles
316     desiredFrameRM = finalFrame_R
317     desiredFrameRM.p[0] = finalFrame_R.p[0]
318     desiredFrameRM.p[1] = finalFrame_R.p[1]
319     desiredFrameRM.p[2] = finalFrame_R.p[2] + altura_pata
320     print "Desired Position: ", desiredFrameRM.p
321     q_out_RM = PyKDL.JntArray(cadena_RM.getNrOfJoints())
322     ik_RM.CartToJnt(q_init_RM, desiredFrameRM, q_out_RM)
323     print "Output angles in rads: ", q_out_RM
324
325     rm_goal1[0] = q_out_RM[0] #= angulo_avance
326     rm_goal1[1] = q_out_RM[1]
327     rm_goal1[2] = q_out_RM[2]
328     rm_goal1[3] = q_out_RM[3]
329
330     LF_actual = PyKDL.JntArray(nj_izq)
331     LF_actual[0] = lf_goal0[0]

```

```

332     LF_actual[1] = lf_goal0[1]
333     LF_actual[2] = lf_goal0[2]
334     LF_actual[3] = lf_goal0[3]
335     print "Inverse Kinematics"
336     fkSolver_LF.JntToCart(LF_actual, finalFrame_L)
337     q_init_LF = LF_actual # initial angles
338     desiredFrameLF = finalFrame_L
339     desiredFrameLF.p[0] = finalFrame_L.p[0]
340     desiredFrameLF.p[1] = finalFrame_L.p[1]
341     desiredFrameLF.p[2] = finalFrame_L.p[2] + altura_pata
342     print "Desired Position: ", desiredFrameLF.p
343     q_out_LF = PyKDL.JntArray(cadena_LF.getNrOfJoints())
344     ik_LF.CartToJnt(q_init_LF, desiredFrameLF, q_out_LF)
345     print "Output angles in rads: ", q_out_LF

346
347     lf_goal1[0] = q_out_LF[0] #= angulo_avance
348     lf_goal1[1] = q_out_LF[1]
349     lf_goal1[2] = q_out_LF[2]
350     lf_goal1[3] = q_out_LF[3]

351
352     # Connect to the right arm trajectory action server
353     rospy.loginfo('Waiting for right arm trajectory controller
...')

354
355     rr_client = actionlib.SimpleActionClient('/hexapodo/
pata_rr/follow_joint_trajectory',
FollowJointTrajectoryAction)
356     rr_client.wait_for_server()

357
358     lm_client = actionlib.SimpleActionClient('/hexapodo/
pata_lm/follow_joint_trajectory',
FollowJointTrajectoryAction)
359     lm_client.wait_for_server()

360
361     rf_client = actionlib.SimpleActionClient('/hexapodo/
pata_rf/follow_joint_trajectory',
FollowJointTrajectoryAction)
362     rf_client.wait_for_server()

363
364     lr_client = actionlib.SimpleActionClient('/hexapodo/
pata_lr/follow_joint_trajectory',

```

```
            FollowJointTrajectoryAction)
365      lr_client.wait_for_server()

366
367      rm_client = actionlib.SimpleActionClient('/hexapodo/
          pata_rm/follow_joint_trajectory',
          FollowJointTrajectoryAction)
368      rm_client.wait_for_server()

369
370      lf_client = actionlib.SimpleActionClient('/hexapodo/
          pata_lf/follow_joint_trajectory',
          FollowJointTrajectoryAction)
371      lf_client.wait_for_server()

372
373      rospy.loginfo('... connected.')

374
375      # Create a single-point arm trajectory with the
          piernas_goal as the end-point
376      rr_trajectory1 = JointTrajectory()
377      rr_trajectory1.joint_names = piernas_joints_RR
378      rr_trajectory1.points.append(JointTrajectoryPoint())
379      rr_trajectory1.points[0].positions = rr_goal0
380      rr_trajectory1.points[0].velocities = [0.0 for i in
          piernas_joints_RR]
381      rr_trajectory1.points[0].accelerations = [0.0 for i in
          piernas_joints_RR]
382      rr_trajectory1.points[0].time_from_start = rospy.Duration
          (0.25)
383      rr_trajectory1.points.append(JointTrajectoryPoint())
384      rr_trajectory1.points[1].positions = rr_goal1
385      rr_trajectory1.points[1].velocities = [0.0 for i in
          piernas_joints_RR]
386      rr_trajectory1.points[1].accelerations = [0.0 for i in
          piernas_joints_RR]
387      rr_trajectory1.points[1].time_from_start = rospy.Duration
          (0.5)
388      rr_trajectory1.points.append(JointTrajectoryPoint())
389      rr_trajectory1.points[2].positions = lr_goal1
390      rr_trajectory1.points[2].velocities = [0.0 for i in
          piernas_joints_RR]
391      rr_trajectory1.points[2].accelerations = [0.0 for i in
          piernas_joints_RR]
```

```

392     rr_trajectory1.points[2].time_from_start = rospy.Duration
393         (0.75)
394     rr_trajectory1.points.append(JointTrajectoryPoint())
395     rr_trajectory1.points[3].positions = lr_goal0
396     rr_trajectory1.points[3].velocities = [0.0 for i in
397         piernas_joints_RR]
398     rr_trajectory1.points[3].accelerations = [0.0 for i in
399         piernas_joints_RR]
400     rr_trajectory1.points[3].time_from_start = rospy.Duration
401         (1)

402
403     lm_trajectory1 = JointTrajectory()
404     lm_trajectory1.joint_names = piernas_joints_LM
405     lm_trajectory1.points.append(JointTrajectoryPoint())
406     lm_trajectory1.points[0].positions = lm_goal0#[0,0,0,0]
407     lm_trajectory1.points[0].velocities = [0.0 for i in
408         piernas_joints_LM]
409     lm_trajectory1.points[0].accelerations = [0.0 for i in
410         piernas_joints_LM]
411     lm_trajectory1.points[0].time_from_start = rospy.Duration
412         (0.25)
413     lm_trajectory1.points.append(JointTrajectoryPoint())
414     lm_trajectory1.points[1].positions = lm_goal1 # [0,0,0,0]
415     lm_trajectory1.points[1].velocities = [0.0 for i in
416         piernas_joints_LM]
417     lm_trajectory1.points[1].accelerations = [0.0 for i in
418         piernas_joints_LM]
419     lm_trajectory1.points[1].time_from_start = rospy.Duration
420         (0.5)
421     lm_trajectory1.points.append(JointTrajectoryPoint())
422     lm_trajectory1.points[2].positions = rm_goal1 # [0,0,0,0]
423     lm_trajectory1.points[2].velocities = [0.0 for i in
424         piernas_joints_LM]
425     lm_trajectory1.points[2].accelerations = [0.0 for i in
426         piernas_joints_LM]
427     lm_trajectory1.points[2].time_from_start = rospy.Duration
428         (0.75)
429     lm_trajectory1.points.append(JointTrajectoryPoint())
430     lm_trajectory1.points[3].positions = rm_goal0 # [0,0,0,0]
431     lm_trajectory1.points[3].velocities = [0.0 for i in
432         piernas_joints_LM]

```

```

419     lm_trajectory1.points[3].accelerations = [0.0 for i in
420         piernas_joints_LM]
421
422     rf_trajectory1 = JointTrajectory()
423     rf_trajectory1.joint_names = piernas_joints_RF
424     rf_trajectory1.points.append(JointTrajectoryPoint())
425     rf_trajectory1.points[0].positions = rf_goal0 # [0,0,0,0]
426     rf_trajectory1.points[0].velocities = [0.0 for i in
427         piernas_joints_RF]
428     rf_trajectory1.points[0].accelerations = [0.0 for i in
429         piernas_joints_RF]
430     rf_trajectory1.points[0].time_from_start = rospy.Duration
431         (0.25)
432     rf_trajectory1.points.append(JointTrajectoryPoint())
433     rf_trajectory1.points[1].positions = rf_goal1 # [0,0,0,0]
434     rf_trajectory1.points[1].velocities = [0.0 for i in
435         piernas_joints_RF]
436     rf_trajectory1.points[1].accelerations = [0.0 for i in
437         piernas_joints_RF]
438     rf_trajectory1.points[1].time_from_start = rospy.Duration
439         (0.5)
440     rf_trajectory1.points.append(JointTrajectoryPoint())
441     rf_trajectory1.points[2].positions = lf_goal1 # [0,0,0,0]
442     rf_trajectory1.points[2].velocities = [0.0 for i in
443         piernas_joints_RF]
444     rf_trajectory1.points[2].accelerations = [0.0 for i in
445         piernas_joints_RF]
446     rf_trajectory1.points[2].time_from_start = rospy.Duration
447         (0.75)
448     rf_trajectory1.points.append(JointTrajectoryPoint())
449     rf_trajectory1.points[3].positions = lf_goal0 # [0,0,0,0]
450     rf_trajectory1.points[3].velocities = [0.0 for i in
451         piernas_joints_RF]
452     rf_trajectory1.points[3].accelerations = [0.0 for i in
453         piernas_joints_RF]
454     rf_trajectory1.points[3].time_from_start = rospy.Duration
455         (1)
456
457     lr_trajectory1 = JointTrajectory()

```

```

446     lr_trajectory1.joint_names = piernas_joints_LR
447     lr_trajectory1.points.append(JointTrajectoryPoint())
448     lr_trajectory1.points[0].positions = lr_goal0#lr_goal0
449     lr_trajectory1.points[0].velocities = [0.0 for i in
450         piernas_joints_RR]
451     lr_trajectory1.points[0].accelerations = [0.0 for i in
452         piernas_joints_RR]
453     lr_trajectory1.points[0].time_from_start = rospy.Duration
454         (0.25)
455     lr_trajectory1.points.append(JointTrajectoryPoint())
456     lr_trajectory1.points[1].positions = lr_goal1
457     lr_trajectory1.points[1].velocities = [0.0 for i in
458         piernas_joints_RR]
459     lr_trajectory1.points[1].accelerations = [0.0 for i in
460         piernas_joints_RR]
461     lr_trajectory1.points[1].time_from_start = rospy.Duration
462         (0.5)
463     lr_trajectory1.points.append(JointTrajectoryPoint())
464     lr_trajectory1.points[2].positions = rr_goal1
465     lr_trajectory1.points[2].velocities = [0.0 for i in
466         piernas_joints_RR]
467     lr_trajectory1.points[2].accelerations = [0.0 for i in
468         piernas_joints_RR]
469     lr_trajectory1.points[2].time_from_start = rospy.Duration
470         (0.75)
471     lr_trajectory1.points.append(JointTrajectoryPoint())
472     lr_trajectory1.points[3].positions = rr_goal0 # lr_goal0
473     lr_trajectory1.points[3].velocities = [0.0 for i in
474         piernas_joints_RR]
475     lr_trajectory1.points[3].accelerations = [0.0 for i in
476         piernas_joints_RR]
477     lr_trajectory1.points[3].time_from_start = rospy.Duration
478         (1)

479
480     rm_trajectory1 = JointTrajectory()
481     rm_trajectory1.joint_names = piernas_joints_RM
482     rm_trajectory1.points.append(JointTrajectoryPoint())
483     rm_trajectory1.points[0].positions = rm_goal0#rm_goal0 #
484         [0,0,0,0]
485     rm_trajectory1.points[0].velocities = [0.0 for i in
486         piernas_joints_LM]

```

```

473     rm_trajectory1.points[0].accelerations = [0.0 for i in
474         piernas_joints_LM]
475     rm_trajectory1.points[0].time_from_start = rospy.Duration
476         (0.25)
477     rm_trajectory1.points.append(JointTrajectoryPoint())
478     rm_trajectory1.points[1].positions = rm_goal1 # [0,0,0,0]
479     rm_trajectory1.points[1].velocities = [0.0 for i in
480         piernas_joints_LM]
481     rm_trajectory1.points[1].accelerations = [0.0 for i in
482         piernas_joints_LM]
483     rm_trajectory1.points[1].time_from_start = rospy.Duration
484         (0.5)
485     rm_trajectory1.points.append(JointTrajectoryPoint())
486     rm_trajectory1.points[2].positions = lm_goal1 # [0,0,0,0]
487     rm_trajectory1.points[2].velocities = [0.0 for i in
488         piernas_joints_LM]
489     rm_trajectory1.points[2].accelerations = [0.0 for i in
490         piernas_joints_LM]
491     rm_trajectory1.points[2].time_from_start = rospy.Duration
492         (0.75)
493     rm_trajectory1.points.append(JointTrajectoryPoint())
494     rm_trajectory1.points[3].positions = lm_goal0 # rm_goal0
495         # [0,0,0,0]
496     rm_trajectory1.points[3].velocities = [0.0 for i in
497         piernas_joints_LM]
498     rm_trajectory1.points[3].accelerations = [0.0 for i in
499         piernas_joints_LM]
500     rm_trajectory1.points[3].time_from_start = rospy.Duration
501         (1)

502     lf_trajectory1 = JointTrajectory()
503     lf_trajectory1.joint_names = piernas_joints_LF
504     lf_trajectory1.points.append(JointTrajectoryPoint())
505     lf_trajectory1.points[0].positions = lf_goal0#lf_goal0 #
506         [0,0,0,0]
507     lf_trajectory1.points[0].velocities = [0.0 for i in
508         piernas_joints_RF]
509     lf_trajectory1.points[0].accelerations = [0.0 for i in
510         piernas_joints_RF]
511     lf_trajectory1.points[0].time_from_start = rospy.Duration
512         (0.25)

```

```

498     lf_trajectory1.points.append(JointTrajectoryPoint())
499     lf_trajectory1.points[1].positions = lf_goal1 # [0,0,0,0]
500     lf_trajectory1.points[1].velocities = [0.0 for i in
501         piernas_joints_RF]
502     lf_trajectory1.points[1].accelerations = [0.0 for i in
503         piernas_joints_RF]
504     lf_trajectory1.points[1].time_from_start = rospy.Duration
505         (0.5)
506     lf_trajectory1.points.append(JointTrajectoryPoint())
507     lf_trajectory1.points[2].positions = rf_goal1 # [0,0,0,0]
508     lf_trajectory1.points[2].velocities = [0.0 for i in
509         piernas_joints_RF]
510     lf_trajectory1.points[2].accelerations = [0.0 for i in
511         piernas_joints_RF]
512     lf_trajectory1.points[2].time_from_start = rospy.Duration
513         (0.75)
514     lf_trajectory1.points.append(JointTrajectoryPoint())
515     lf_trajectory1.points[3].positions = rf_goal0 # lf_goal0
516         # [0,0,0,0]
517     lf_trajectory1.points[3].velocities = [0.0 for i in
518         piernas_joints_RF]
519     lf_trajectory1.points[3].accelerations = [0.0 for i in
520         piernas_joints_RF]
521     lf_trajectory1.points[3].time_from_start = rospy.Duration
522         (1)

523
524     # Send the trajectory to the arm action server
525     rospy.loginfo('Moving the arm to goal position...')

526
527     # Create an empty trajectory goal
528     rr_goal = FollowJointTrajectoryGoal()
529     lm_goal = FollowJointTrajectoryGoal()
530     rf_goal = FollowJointTrajectoryGoal()

531
532     lr_goal = FollowJointTrajectoryGoal()
533     rm_goal = FollowJointTrajectoryGoal()
534     lf_goal = FollowJointTrajectoryGoal()
535     # Set the trajectory component to the goal trajectory
536     # created above

```

```
528     rr_goal.trajectory = rr_trajectory1
529     lm_goal.trajectory = lm_trajectory1
530     rf_goal.trajectory = rf_trajectory1
531     lr_goal.trajectory = lr_trajectory1
532     rm_goal.trajectory = rm_trajectory1
533     lf_goal.trajectory = lf_trajectory1
534     # Specify zero tolerance for the execution time
535     rr_goal.goal_time_tolerance = rospy.Duration(0.01)
536     lm_goal.goal_time_tolerance = rospy.Duration(0.01)
537     rf_goal.goal_time_tolerance = rospy.Duration(0.01)
538     lr_goal.goal_time_tolerance = rospy.Duration(0.01)
539     rm_goal.goal_time_tolerance = rospy.Duration(0.01)
540     lf_goal.goal_time_tolerance = rospy.Duration(0.01)
541     # Send the goal to the action server
542     rr_client.send_goal(rr_goal)
543     lm_client.send_goal(lm_goal)
544     rf_client.send_goal(rf_goal)
545     #rr_client.wait_for_result(rospy.Duration(5.0))
546     print 'Resultado recibido'
547     lr_client.send_goal(lr_goal)
548     rm_client.send_goal(rm_goal)
549     lf_client.send_goal(lf_goal)
550
551     rr_client.wait_for_result()
552     rr_client.send_goal(rr_goal)
553     lm_client.send_goal(lm_goal)
554     rf_client.send_goal(rf_goal)
555     # rr_client.wait_for_result(rospy.Duration(5.0))
556     print 'Resultado recibido'
557     lr_client.send_goal(lr_goal)
558     rm_client.send_goal(rm_goal)
559     lf_client.send_goal(lf_goal)
560
561     rr_client.wait_for_result()
562     rr_client.send_goal(rr_goal)
563     lm_client.send_goal(lm_goal)
564     rf_client.send_goal(rf_goal)
565     # rr_client.wait_for_result(rospy.Duration(5.0))
566     print 'Resultado recibido'
567     lr_client.send_goal(lr_goal)
568     rm_client.send_goal(rm_goal)
```

```
569     lf_client.send_goal(lf_goal)
570
571     rr_client.wait_for_result()
572     rr_client.send_goal(rr_goal)
573     lm_client.send_goal(lm_goal)
574     rf_client.send_goal(rf_goal)
575     # rr_client.wait_for_result(rospy.Duration(5.0))
576     print 'Resultado recibido'
577     lr_client.send_goal(lr_goal)
578     rm_client.send_goal(rm_goal)
579     lf_client.send_goal(lf_goal)
580
581     rr_client.wait_for_result()
582     rr_client.send_goal(rr_goal)
583     lm_client.send_goal(lm_goal)
584     rf_client.send_goal(rf_goal)
585     # rr_client.wait_for_result(rospy.Duration(5.0))
586     print 'Resultado recibido'
587     lr_client.send_goal(lr_goal)
588     rm_client.send_goal(rm_goal)
589     lf_client.send_goal(lf_goal)
590
591     rr_client.wait_for_result()
592     rr_client.send_goal(rr_goal)
593     lm_client.send_goal(lm_goal)
594     rf_client.send_goal(rf_goal)
595     # rr_client.wait_for_result(rospy.Duration(5.0))
596     print 'Resultado recibido'
597     lr_client.send_goal(lr_goal)
598     rm_client.send_goal(rm_goal)
599     lf_client.send_goal(lf_goal)
600
601     rr_client.wait_for_result()
602     rr_client.send_goal(rr_goal)
603     lm_client.send_goal(lm_goal)
604     rf_client.send_goal(rf_goal)
605     # rr_client.wait_for_result(rospy.Duration(5.0))
606     print 'Resultado recibido'
607     lr_client.send_goal(lr_goal)
608     rm_client.send_goal(rm_goal)
609     lf_client.send_goal(lf_goal)
```

```
610
611     rr_client.wait_for_result()
612     rr_client.send_goal(rr_goal)
613     lm_client.send_goal(lm_goal)
614     rf_client.send_goal(rf_goal)
615     # rr_client.wait_for_result(rospy.Duration(5.0))
616     print 'Resultado recibido'
617     lr_client.send_goal(lr_goal)
618     rm_client.send_goal(rm_goal)
619     lf_client.send_goal(lf_goal)
620
621     rr_client.wait_for_result()
622     rr_client.send_goal(rr_goal)
623     lm_client.send_goal(lm_goal)
624     rf_client.send_goal(rf_goal)
625     # rr_client.wait_for_result(rospy.Duration(5.0))
626     print 'Resultado recibido'
627     lr_client.send_goal(lr_goal)
628     rm_client.send_goal(rm_goal)
629     lf_client.send_goal(lf_goal)
630
631     rr_client.wait_for_result()
632     rr_client.send_goal(rr_goal)
633     lm_client.send_goal(lm_goal)
634     rf_client.send_goal(rf_goal)
635     # rr_client.wait_for_result(rospy.Duration(5.0))
636     print 'Resultado recibido'
637     lr_client.send_goal(lr_goal)
638     rm_client.send_goal(rm_goal)
639     lf_client.send_goal(lf_goal)
640     #lr_client.wait_for_result(rospy.Duration(5.0))
641     #rr_client.send_goal(rr_goal)
642     #lm_client.send_goal(lm_goal)
643     #rf_client.send_goal(rf_goal) ''
644
645     if not sync:
646         # Wait for up to 5 seconds for the motion to complete
647         rr_client.wait_for_result(rospy.Duration(5.0))
648
649     rr_client.wait_for_result()
650     print rr_client.get_result()
```

```
651
652
653         rospy.loginfo('... done')
654
655 if __name__ == '__main__':
656     try:
657         TrajectoryDemo()
658     except rospy.ROSInterruptException:
659         pass
```

D.2. Caminada lateral

```
1 #!/usr/bin/env python
2
3 from scipy.io import loadmat
4
5 import actionlib
6 import rospy
7 from control_msgs.msg import FollowJointTrajectoryAction,
8     FollowJointTrajectoryGoal
9 from trajectory_msgs.msg import JointTrajectory,
10    JointTrajectoryPoint
11
12 class TrajectoryDemo():
13     def __init__(self):
14
15         x = loadmat('hex_wave.mat')#hex_transversal hex_lateral2
16             hex_lateral_r hex_wave
17             # print x
18             pierna1 = x['pierna1']
19             pierna2 = x['pierna2']
20             pierna3 = x['pierna3']
21             pierna4 = x['pierna4']
22             pierna5 = x['pierna5']
23             pierna6 = x['pierna6']
24
25             rospy.init_node('trajectory_hex')
```

```

26     # Set to True to move back to the starting configurations
27     reset = rospy.get_param('~reset', False)
28
29     # Set to False to wait for arm to finish before moving
30     head
31     sync = rospy.get_param('~sync', True)
32
33     # Which joints define the arm?
34     piernas_joints_RR = ['coxa_joint_RR', 'femur_joint_RR', ,
35                           'tibia_joint_RR', 'tarsus_joint_RR']
36     piernas_joints_LM = ['coxa_joint_LM', 'femur_joint_LM', ,
37                           'tibia_joint_LM', 'tarsus_joint_LM']
38     piernas_joints_RF = ['coxa_joint_RF', 'femur_joint_RF', ,
39                           'tibia_joint_RF', 'tarsus_joint_RF']
40
41     piernas_joints_LR = ['coxa_joint_LR', 'femur_joint_LR', ,
42                           'tibia_joint_LR', 'tarsus_joint_LR']
43     piernas_joints_RM = ['coxa_joint_RM', 'femur_joint_RM', ,
44                           'tibia_joint_RM', 'tarsus_joint_RM']
45     piernas_joints_LF = ['coxa_joint_LF', 'femur_joint_LF', ,
46                           'tibia_joint_LF', 'tarsus_joint_LF']
47
48     rospy.loginfo('Waiting for right arm trajectory controller
49     ...')
50
51     rr_client = actionlib.SimpleActionClient('/hexapodo/
52                                               pata_rr/follow_joint_trajectory',
53                                               FollowJointTrajectoryAction
54                                               )
55     rr_client.wait_for_server()
56
57     lm_client = actionlib.SimpleActionClient('/hexapodo/
58                                               pata_lm/follow_joint_trajectory',
59                                               FollowJointTrajectoryAction
60                                               )
61     lm_client.wait_for_server()
62
63     rf_client = actionlib.SimpleActionClient('/hexapodo/
64                                               pata_rf/follow_joint_trajectory',
65                                               FollowJointTrajectoryAction
66                                               )

```

```

53         rf_client.wait_for_server()
54
55         lr_client = actionlib.SimpleActionClient('/hexapodo/
56                                         pata_lr/follow_joint_trajectory',
56                                         FollowJointTrajectoryAction
57                                         )
58
59         rm_client = actionlib.SimpleActionClient('/hexapodo/
60                                         pata_rm/follow_joint_trajectory',
60                                         FollowJointTrajectoryAction
61                                         )
62
63         lf_client = actionlib.SimpleActionClient('/hexapodo/
64                                         pata_lf/follow_joint_trajectory',
64                                         FollowJointTrajectoryAction
65                                         )
66
67         lf_client.wait_for_server()
68
69         rospy.loginfo('... connected.')
70
71         rr_trajectory1 = JointTrajectory()
72         rr_trajectory1.joint_names = piernas_joints_RR
73         lm_trajectory1 = JointTrajectory()
74         lm_trajectory1.joint_names = piernas_joints_LM
75         rf_trajectory1 = JointTrajectory()
76         rf_trajectory1.joint_names = piernas_joints_RF
77         lr_trajectory1 = JointTrajectory()
78         lr_trajectory1.joint_names = piernas_joints_LR
79         rm_trajectory1 = JointTrajectory()
80         rm_trajectory1.joint_names = piernas_joints_RM
81         lf_trajectory1 = JointTrajectory()
82         lf_trajectory1.joint_names = piernas_joints_LF
83
84         # piernas_goalF0 = [0, -0.3, -0.4, 0.8, -0.3, -0.4,
85         #                     0, -0.4, -0.3, 0.6, +0.4, -0.3]
86         # arm_trajectory.points.append(JointTrajectoryPoint())
87         # arm_trajectory.points[0].positions = piernas_goalF0
88         # arm_trajectory.points[0].velocities = [0.0 for k in
89         #                                         piernas_goalF0]

```

```

87      # arm_trajectory.points[0].accelerations = [0.0 for k in
88          piernas_goalF0]
89
90      # arm_trajectory.points[0].time_from_start = rospy.
91          Duration(2)
92
93
94      piernasLF_goalF01 = [ pierna6[0, 0], pierna6[1, 0], pierna6
95          [2, 0], -0.4]#0.35
96      lf_trajectory1.points.append(JointTrajectoryPoint())
97      lf_trajectory1.points[0].positions = piernasLF_goalF01
98      lf_trajectory1.points[0].time_from_start = rospy.Duration
99          (1)
100
101     piernasLM_goalF01 = [ pierna5[0, 0], pierna5[1, 0], pierna5
102         [2, 0], -0.4]#0.35
103     lm_trajectory1.points.append(JointTrajectoryPoint())
104     lm_trajectory1.points[0].positions = piernasLM_goalF01
105     lm_trajectory1.points[0].time_from_start = rospy.Duration
106         (1)
107
108     piernasLR_goalF01 = [ pierna4[0, 0], pierna4[1, 0], pierna4
109         [2, 0], -0.4]#0.35
110     lr_trajectory1.points.append(JointTrajectoryPoint())
111     lr_trajectory1.points[0].positions = piernasLR_goalF01
112     lr_trajectory1.points[0].time_from_start = rospy.Duration
113         (1)
114

```

```

115      piernasRR_goalF01 = [ pierna1[0, 0], -pierna1[1, 0], -
116          pierna1[2, 0], 0.4]#0.35
117      rr_trajectory1.points.append(JointTrajectoryPoint())
118      rr_trajectory1.points[0].positions = piernasRR_goalF01
119      rr_trajectory1.points[0].time_from_start = rospy.Duration
120          (1)
121
122
123      ind = 1
124      salto = 5
125      t_actual = 1
126      for i in range(1, int(pierna1.shape[1]*0.25), salto):
127          piernasLF_goalF01 = [pierna6[0, i], pierna6[1, i],
128              pierna6[2, i], -0.4]#0.35
129          lf_trajectory1.points.append(JointTrajectoryPoint())
130          lf_trajectory1.points[ind].positions =
131              piernasLF_goalF01
132          lf_trajectory1.points[0].velocities = [0.0 for k in
133              piernasLF_goalF01]
134          lf_trajectory1.points[0].accelerations = [0.0 for k in
135              piernasLF_goalF01]
136
137          piernasLM_goalF01 = [pierna5[0, i], pierna5[1, i],
138              pierna5[2, i], -0.4]#0.35
139          lm_trajectory1.points.append(JointTrajectoryPoint())
140          lm_trajectory1.points[ind].positions =
141              piernasLM_goalF01
142          lm_trajectory1.points[0].velocities = [0.0 for k in
143              piernasLM_goalF01]
144
145          piernasLR_goalF01 = [pierna4[0, i], pierna4[1, i],
146              pierna4[2, i], -0.4]#0.35
147          lr_trajectory1.points.append(JointTrajectoryPoint())
148          lr_trajectory1.points[ind].positions =
149              piernasLR_goalF01
150          lr_trajectory1.points[0].velocities = [0.0 for k in
151              piernasLR_goalF01]

```

```

143         lr_trajectory1.points[0].accelerations = [0.0 for k in
144                                         piernasLR_goalF01]
145
145     piernasRF_goalF01 = [pierna3[0, i], -pierna3[1, i], -
146                           pierna3[2, i], 0.4]#0.35
146     rf_trajectory1.points.append(JointTrajectoryPoint())
147     rf_trajectory1.points[ind].positions =
148         piernasRF_goalF01
148     rf_trajectory1.points[0].velocities = [0.0 for k in
149                                         piernasRF_goalF01]
149     rf_trajectory1.points[0].accelerations = [0.0 for k in
150                                         piernasRF_goalF01]
150
151     piernasRM_goalF01 = [pierna2[0, i], -pierna2[1, 0], -
152                           pierna2[2, i], 0.4]#0.35
152     rm_trajectory1.points.append(JointTrajectoryPoint())
153     rm_trajectory1.points[ind].positions =
154         piernasRM_goalF01
154     rm_trajectory1.points[0].velocities = [0.0 for k in
155                                         piernasRM_goalF01]
155     rm_trajectory1.points[0].accelerations = [0.0 for k in
156                                         piernasRM_goalF01]
156
157     piernasRR_goalF01 = [pierna1[0, i], -pierna1[1, 0], -
158                           pierna1[2, i], 0.4]#0.35
158     rr_trajectory1.points.append(JointTrajectoryPoint())
159     rr_trajectory1.points[ind].positions =
160         piernasRR_goalF01
160     rr_trajectory1.points[0].velocities = [0.0 for k in
161                                         piernasRR_goalF01]
161     rr_trajectory1.points[0].accelerations = [0.0 for k in
162                                         piernasRR_goalF01]
162     # print ind
163     # print int(art1_der['senialCompleta'].shape[1]*0.2)
164     # arm_trajectory.points[ind].velocities = [0.0 for j
164       in piernas_goalF]
165     # arm_trajectory.points[ind].accelerations = [0.0 for
165       j in piernas_goalF]
166     # art1_der['tiempo_completo'][0, i]
167     # 0.01 * (ind + 1)
168     t_actual+=0.01

```

```

169         lf_trajectory1.points[ind].time_from_start = rospy.
170             Duration(t_actual)
171         lm_trajectory1.points[ind].time_from_start = rospy.
172             Duration(t_actual)
173         lr_trajectory1.points[ind].time_from_start = rospy.
174             Duration(t_actual)
175         rf_trajectory1.points[ind].time_from_start = rospy.
176             Duration(t_actual)
177         rm_trajectory1.points[ind].time_from_start = rospy.
178             Duration(t_actual)
179         rr_trajectory1.points[ind].time_from_start = rospy.
180             Duration(t_actual)
181         # print 1+art1_der[ 'tiempo_completo '][0 , i]
182         ind+=1
183         # print art1_der[ 'senalCompleta '][0 , i]
184
185         # Create an empty trajectory goal
186         rosify.loginfo('Moving the arm to goal position... ')
187         print piernal.shape[1]
188
189         # Create an empty trajectory goal
190         rr_goal = FollowJointTrajectoryGoal()
191         lm_goal = FollowJointTrajectoryGoal()
192         rf_goal = FollowJointTrajectoryGoal()
193
194         lr_goal = FollowJointTrajectoryGoal()
195         rm_goal = FollowJointTrajectoryGoal()
196         lf_goal = FollowJointTrajectoryGoal()
197
198         # Set the trajectory component to the goal trajectory
199         # created above
200         rr_goal.trajectory = rr_trajectory1
201         lm_goal.trajectory = lm_trajectory1
202         rf_goal.trajectory = rf_trajectory1
203         lr_goal.trajectory = lr_trajectory1
204         rm_goal.trajectory = rm_trajectory1
205         lf_goal.trajectory = lf_trajectory1
206
207         # Specify zero tolerance for the execution time
208         rr_goal.goal_time_tolerance = rospy.Duration(0.01)
209         lm_goal.goal_time_tolerance = rospy.Duration(0.01)
210         rf_goal.goal_time_tolerance = rospy.Duration(0.01)

```

```
203     lr_goal.goal_time_tolerance = rospy.Duration(0.01)
204     rm_goal.goal_time_tolerance = rospy.Duration(0.01)
205     lf_goal.goal_time_tolerance = rospy.Duration(0.01)
206
207     # Send the goal to the action server
208     rr_client.send_goal(rr_goal)
209     lm_client.send_goal(lm_goal)
210     rf_client.send_goal(rf_goal)
211
212
213     lr_client.send_goal(lr_goal)
214     rm_client.send_goal(rm_goal)
215     lf_client.send_goal(lf_goal)
216     # rr_client.wait_for_result(rospy.Duration(5.0))
217     rr_client.wait_for_result()
218
219     # rr_client.send_goal(rr_goal)
220     # lm_client.send_goal(lm_goal)
221     # rf_client.send_goal(rf_goal)
222     #
223     # lr_client.send_goal(lr_goal)
224     # rm_client.send_goal(rm_goal)
225     # lf_client.send_goal(lf_goal)
226     # rr_client.wait_for_result(rospy.Duration(5.0))
227     # rr_client.wait_for_result()
228     #
229     # lr_client.send_goal(lr_goal)
230     # rm_client.send_goal(rm_goal)
231     # lf_client.send_goal(lf_goal)
232     # rr_client.wait_for_result(rospy.Duration(5.0))
233     # rr_client.wait_for_result()
234     #
235     # rr_client.send_goal(rr_goal)
236     # lm_client.send_goal(lm_goal)
237     # rf_client.send_goal(rf_goal)
238     #
239     # lr_client.send_goal(lr_goal)
240     # rm_client.send_goal(rm_goal)
241     # lf_client.send_goal(lf_goal)
242     # rr_client.wait_for_result(rospy.Duration(5.0))
243     # rr_client.wait_for_result()
```

```
244     #''
245
246     rospy.loginfo('... done')
247
248 if __name__ == '__main__':
249     try:
250         TrajectoryDemo()
251     except rospy.ROSInterruptException:
252         pass
```

D.3. Giros

```
1 #!/usr/bin/env python
2
3 import actionlib
4 import rospy
5
6 from control_msgs.msg import FollowJointTrajectoryAction,
7     FollowJointTrajectoryGoal
8 from trajectory_msgs.msg import JointTrajectory,
9     JointTrajectoryPoint
10 import urdf_parser_py.urdf
11 import urdf_parser_py.xml_reflection
12 import kdl_parser_py.urdf
13 import PyKDL
14 import hrpsys.ModelLoader_idl
15 import copy
16 from angles import normalize
17
18 import pickle
19
20
21 class TrajectoryDemo():
22     def __init__(self):
23
24         filename = '/home/kaiser/WS_ROS/catkin_ws/src/urdf_serp/
25             urdf/mi_hexapodo_exp.urdf'
26
27         angulo_avance = +0.20 #rad
28         altura_pata = +0.03 #cm
29         # angulo_avance = 0 # +0.40 #rad
```

```

26      # altura_pata = 0  # -0.04 #cm
27
28      robot = urdf_parser_py.urdf.URDF.from_xml_file(filename)
29
30      (ok, tree) = kdl_parser_py.urdf.treeFromFile(filename)
31      cadena_RR = tree.getChain("base_link", "tarsus_RR")
32      cadena_RM = tree.getChain("base_link", "tarsus_RM")
33      cadena_RF = tree.getChain("base_link", "tarsus_RF")
34      cadena_LR = tree.getChain("base_link", "tarsus_LR")
35      cadena_LM = tree.getChain("base_link", "tarsus_LM")
36      cadena_LF = tree.getChain("base_link", "tarsus_LF")
37
38      print cadena_RR.getNrOfSegments()
39      fksolver_RR = PyKDL.ChainFkSolverPos_recursive(cadena_RR)
40      fksolver_RM= PyKDL.ChainFkSolverPos_recursive(cadena_RM)
41      fksolver_RF = PyKDL.ChainFkSolverPos_recursive(cadena_RF)
42      fksolver_LR = PyKDL.ChainFkSolverPos_recursive(cadena_LR)
43      fksolver_LM = PyKDL.ChainFkSolverPos_recursive(cadena_LM)
44      fksolver_LF = PyKDL.ChainFkSolverPos_recursive(cadena_LF)
45
46      vik_RR = PyKDL.ChainIkSolverVel_pinv(cadena_RR)
47      ik_RR = PyKDL.ChainIkSolverPos_NR(cadena_RR, fksolver_RR,
48                                         vik_RR)
49
50      vik_RM = PyKDL.ChainIkSolverVel_pinv(cadena_RM)
51      ik_RM = PyKDL.ChainIkSolverPos_NR(cadena_RM, fksolver_RM,
52                                         vik_RM)
53
54      vik_RF = PyKDL.ChainIkSolverVel_pinv(cadena_RF)
55      ik_RF = PyKDL.ChainIkSolverPos_NR(cadena_RF, fksolver_RF,
56                                         vik_RF)
57
58      vik_LR = PyKDL.ChainIkSolverVel_pinv(cadena_LR)
59      ik_LR = PyKDL.ChainIkSolverPos_NR(cadena_LR, fksolver_LR,
60                                         vik_LR)
61
62      vik_LM = PyKDL.ChainIkSolverVel_pinv(cadena_LM)
63      ik_LM = PyKDL.ChainIkSolverPos_NR(cadena_LM, fksolver_LM,
64                                         vik_LM)
65
66      vik_LF = PyKDL.ChainIkSolverVel_pinv(cadena_LF)

```

```

62     ik_LF = PyKDL.ChainIkSolverPos_NR(cadena_LF, fksolver_LF,
63                                         vik_LF)
64
65     nj_izq = cadena_RR.getNrOfJoints()
66     posicionInicial_R = PyKDL.JntArray(nj_izq)
67     posicionInicial_R[0] = 0
68     posicionInicial_R[1] = 0
69     posicionInicial_R[2] = 0
70     posicionInicial_R[3] = 0
71
72     nj_izq = cadena_LR.getNrOfJoints()
73     posicionInicial_L = PyKDL.JntArray(nj_izq)
74     posicionInicial_L[0] = 0
75     posicionInicial_L[1] = 0
76     posicionInicial_L[2] = 0
77     posicionInicial_L[3] = 0
78
79     print "Forward kinematics"
80     finalFrame_R = PyKDL.Frame()
81     finalFrame_L = PyKDL.Frame()
82
83     rospy.init_node('trajectory_demo')
84
85     # Set to True to move back to the starting configurations
86     reset = rospy.get_param('~reset', False)
87
88     # Set to False to wait for arm to finish before moving
89     # head
90     sync = rospy.get_param('~sync', True)
91
92     # Which joints define the arm?
93     piernas_joints_RR = ['coxa_joint_RR', 'femur_joint_RR', ,
94                           'tibia_joint_RR', 'tarsus_joint_RR']
95     piernas_joints_LM = ['coxa_joint_LM', 'femur_joint_LM', ,
96                           'tibia_joint_LM', 'tarsus_joint_LM']
97     piernas_joints_RF = ['coxa_joint_RF', 'femur_joint_RF', ,
98                           'tibia_joint_RF', 'tarsus_joint_RF']
99
100    piernas_joints_LR = ['coxa_joint_LR', 'femur_joint_LR', ,
101                           'tibia_joint_LR', 'tarsus_joint_LR']

```

```

96     piernas_joints_RM = [ 'coxa_joint_RM' , 'femur_joint_RM' , ,
97         'tibia_joint_RM' , 'tarsus_joint_RM' ]
98     piernas_joints_LF = [ 'coxa_joint_LF' , 'femur_joint_LF' , ,
99         'tibia_joint_LF' , 'tarsus_joint_LF' ]
100
101     rr_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
102     lm_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
103     rf_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
104     rr_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
105     lm_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
106     rf_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
107     lr_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
108     rm_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
109     lf_goal0 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
110     lr_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
111     rm_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
112     lf_goal1 = [ 0.0 , 0.0 , 0.0 , 0.0 ]
113
114     #1111111111111111111111111111111111111111111111111111111111111111
115     print "Inverse Kinematics"
116     fksolver_RR.JntToCart( posicionInicial_R , finalFrame_R )
117     q_init_RR = posicionInicial_R # initial angles
118     desiredFrameRR = finalFrame_R
119     desiredFrameRR.p[0] = finalFrame_R.p[0]
120     desiredFrameRR.p[1] = finalFrame_R.p[1]
121     desiredFrameRR.p[2] = finalFrame_R.p[2]+altura_pata
122     print "Desired Position: ", desiredFrameRR.p
123     q_out_RR = PyKDL.JntArray(cadena_RR.getNrOfJoints())
124     ik_RR.CartToJnt(q_init_RR , desiredFrameRR , q_out_RR)
125     print "Output angles in rads: " , q_out_RR
126
127     rr_goal0[0] = q_out_RR[0]+angulo_avance
128     rr_goal0[1] = q_out_RR[1]
129     rr_goal0[2] = q_out_RR[2]
130     rr_goal0[3] = q_out_RR[3]
131
132     print "Inverse Kinematics"
133     fksolver_LM.JntToCart( posicionInicial_L , finalFrame_L )
134     q_init_LM = posicionInicial_L # initial angles
135     desiredFrameLM = finalFrame_L
136     desiredFrameLM.p[0] = finalFrame_L.p[0]

```

```

135     desiredFrameLM.p[1] = finalFrame_L.p[1]
136     desiredFrameLM.p[2] = finalFrame_L.p[2] + altura_pata
137     print "Desired Position: ", desiredFrameLM.p
138     q_out_LM = PyKDL.JntArray(cadena_LM.getNrOfJoints())
139     ik_LM.CartToJnt(q_init_LM, desiredFrameLM, q_out_LM)
140     print "Output angles in rads: ", q_out_LM
141
142     lm_goal0[0] = q_out_LM[0]-angulo_avance
143     lm_goal0[1] = q_out_LM[1]
144     lm_goal0[2] = q_out_LM[2]
145     lm_goal0[3] = q_out_LM[3]
146
147     print "Inverse Kinematics"
148     fkssolver_RF.JntToCart(posicionInicial_R, finalFrame_R)
149     q_init_RF = posicionInicial_R # initial angles
150     desiredFrameRF = finalFrame_R
151     desiredFrameRF.p[0] = finalFrame_R.p[0]
152     desiredFrameRF.p[1] = finalFrame_R.p[1]
153     desiredFrameRF.p[2] = finalFrame_R.p[2] + altura_pata
154     print "Desired Position: ", desiredFrameRF.p
155     q_out_RF = PyKDL.JntArray(cadena_RF.getNrOfJoints())
156     ik_RF.CartToJnt(q_init_RF, desiredFrameRF, q_out_RF)
157     print "Output angles in rads: ", q_out_RF
158
159     rf_goal0[0] = q_out_RF[0]+angulo_avance
160     rf_goal0[1] = q_out_RF[1]
161     rf_goal0[2] = q_out_RF[2]
162     rf_goal0[3] = q_out_RF[3]
163
164     # 2222222222222222222222222222222222222222222222222222
165     RR_actual = PyKDL.JntArray(nj_izq)
166     RR_actual[0] = rr_goal0[0]
167     RR_actual[1] = rr_goal0[1]
168     RR_actual[2] = rr_goal0[2]
169     RR_actual[3] = rr_goal0[3]
170     print "Inverse Kinematics"
171     fkssolver_RR.JntToCart(RR_actual, finalFrame_R)
172     q_init_RR = RR_actual # initial angles
173     desiredFrameRR = finalFrame_R
174     desiredFrameRR.p[0] = finalFrame_R.p[0]
175     desiredFrameRR.p[1] = finalFrame_R.p[1]

```

```

176     desiredFrameRR.p[2] = finalFrame_R.p[2] - altura_pata
177     print "Desired Position: ", desiredFrameRR.p
178     q_out_RR = PyKDL.JntArray(cadena_RR.getNrOfJoints())
179     ik_RR.CartToJnt(q_init_RR, desiredFrameRR, q_out_RR)
180     print "Output angles in rads: ", q_out_RR
181
182     rr_goal1[0] = q_out_RR[0]
183     rr_goal1[1] = q_out_RR[1]
184     rr_goal1[2] = q_out_RR[2]
185     rr_goal1[3] = q_out_RR[3]
186
187     rr_goal2 = rr_goal1[:]
188     rr_goal2[0] = -angulo_avance
189
190     LM_actual = PyKDL.JntArray(nj_izq)
191     LM_actual[0] = lm_goal0[0]
192     LM_actual[1] = lm_goal0[1]
193     LM_actual[2] = lm_goal0[2]
194     LM_actual[3] = lm_goal0[3]
195     print "Inverse Kinematics"
196     fksolver_LM.JntToCart(LM_actual, finalFrame_L)
197     q_init_LM = LM_actual # initial angles
198     desiredFrameLM = finalFrame_L
199     desiredFrameLM.p[0] = finalFrame_L.p[0]
200     desiredFrameLM.p[1] = finalFrame_L.p[1]
201     desiredFrameLM.p[2] = finalFrame_L.p[2] - altura_pata
202     print "Desired Position: ", desiredFrameLM.p
203     q_out_LM = PyKDL.JntArray(cadena_LM.getNrOfJoints())
204     ik_LM.CartToJnt(q_init_LM, desiredFrameLM, q_out_LM)
205     print "Output angles in rads: ", q_out_LM
206
207     lm_goal1[0] = q_out_LM[0]
208     lm_goal1[1] = q_out_LM[1]
209     lm_goal1[2] = q_out_LM[2]
210     lm_goal1[3] = q_out_LM[3]
211
212     lm_goal2 = lm_goal1[:]
213     lm_goal2[0] = angulo_avance
214
215     RF_actual = PyKDL.JntArray(nj_izq)
216     RF_actual[0] = rf_goal0[0]

```

```

217     RF_actual[1] = rf_goal0[1]
218     RF_actual[2] = rf_goal0[2]
219     RF_actual[3] = rf_goal0[3]
220     print "Inverse Kinematics"
221     fkSolver_RF.JntToCart(RF_actual, finalFrame_R)
222     q_init_RF = RF_actual # initial angles
223     desiredFrameRF = finalFrame_R
224     desiredFrameRF.p[0] = finalFrame_R.p[0]
225     desiredFrameRF.p[1] = finalFrame_R.p[1]
226     desiredFrameRF.p[2] = finalFrame_R.p[2] - altura_pata
227     print "Desired Position: ", desiredFrameRF.p
228     q_out_RF = PyKDL.JntArray(cadena_RF.getNrOfJoints())
229     ik_RF.CartToJnt(q_init_RF, desiredFrameRF, q_out_RF)
230     print "Output angles in rads: ", q_out_RF

231
232     rf_goal1[0] = q_out_RF[0]
233     rf_goal1[1] = q_out_RF[1]
234     rf_goal1[2] = q_out_RF[2]
235     rf_goal1[3] = q_out_RF[3]

236
237     rf_goal2 = rf_goal1 [:]
238     rf_goal2[0] = -angulo_avance

239
240     # 111111111111111111111111111111111111111111111111111111111111111111
241     print "Inverse Kinematics"
242     fkSolver_LR.JntToCart(posicionInicial_L, finalFrame_L)
243     q_init_LR = posicionInicial_L # initial angles
244     desiredFrameLR = finalFrame_L
245     desiredFrameLR.p[0] = finalFrame_L.p[0]
246     desiredFrameLR.p[1] = finalFrame_L.p[1]
247     desiredFrameLR.p[2] = finalFrame_L.p[2] # + altura_pata
248     print "Desired Position: ", desiredFrameLR.p
249     q_out_LR = PyKDL.JntArray(cadena_LR.getNrOfJoints())
250     ik_LR.CartToJnt(q_init_LR, desiredFrameLR, q_out_LR)
251     print "Output angles in rads: ", q_out_LR

252
253     lr_goal0[0] = q_out_LR[0] + angulo_avance
254     lr_goal0[1] = q_out_LR[1]
255     lr_goal0[2] = q_out_LR[2]
256     lr_goal0[3] = q_out_LR[3]

```

```

258     print "Inverse Kinematics"
259     fksolver_RM.JntToCart( posicionInicial_R , finalFrame_R )
260     q_init_RM = posicionInicial_R # initial angles
261     desiredFrameRM = finalFrame_R
262     desiredFrameRM.p[0] = finalFrame_R.p[0]
263     desiredFrameRM.p[1] = finalFrame_R.p[1]
264     desiredFrameRM.p[2] = finalFrame_R.p[2] # + altura_pata
265     print "Desired Position: ", desiredFrameRM.p
266     q_out_RM = PyKDL.JntArray(cadena_RM.getNrOfJoints())
267     ik_RM.CartToJnt(q_init_RM, desiredFrameRM, q_out_RM)
268     print "Output angles in rads: ", q_out_RM
269
270     rm_goal0[0] = q_out_RM[0] - angulo_avance
271     rm_goal0[1] = q_out_RM[1]
272     rm_goal0[2] = q_out_RM[2]
273     rm_goal0[3] = q_out_RM[3]
274
275     print "Inverse Kinematics"
276     fksolver_LF.JntToCart( posicionInicial_L , finalFrame_L )
277     q_init_LF = posicionInicial_L # initial angles
278     desiredFrameLF = finalFrame_L
279     desiredFrameLF.p[0] = finalFrame_L.p[0]
280     desiredFrameLF.p[1] = finalFrame_L.p[1]
281     desiredFrameLF.p[2] = finalFrame_L.p[2] # + altura_pata
282     print "Desired Position: ", desiredFrameLF.p
283     q_out_LF = PyKDL.JntArray(cadena_LF.getNrOfJoints())
284     ik_LF.CartToJnt(q_init_LF, desiredFrameLF, q_out_LF)
285     print "Output angles in rads: ", q_out_LF
286
287     lf_goal0[0] = q_out_LF[0] + angulo_avance
288     lf_goal0[1] = q_out_LF[1]
289     lf_goal0[2] = q_out_LF[2]
290     lf_goal0[3] = q_out_LF[3]
291
292     # 222222222222222222222222222222222222222222222222222
293
294     LR_actual = PyKDL.JntArray(nj_izq)
295     LR_actual[0] = lr_goal0[0]
296     LR_actual[1] = lr_goal0[1]
297     LR_actual[2] = lr_goal0[2]
298     LR_actual[3] = lr_goal0[3]

```

```

299     print "Inverse Kinematics"
300     fkssolver_LR.JntToCart(LR_actual, finalFrame_L)
301     q_init_LR = LR_actual # initial angles
302     print "Inverse Kinematics"
303     desiredFrameLR = finalFrame_L
304     desiredFrameLR.p[0] = finalFrame_L.p[0]
305     desiredFrameLR.p[1] = finalFrame_L.p[1]
306     desiredFrameLR.p[2] = finalFrame_L.p[2] + altura_pata
307     print "Desired Position: ", desiredFrameLR.p
308     q_out_LR = PyKDL.JntArray(cadena_LR.getNrOfJoints())
309     ik_LR.CartToJnt(q_init_LR, desiredFrameLR, q_out_LR)
310     print "Output angles in rads: ", q_out_LR
311
312     lr_goal1[0] = q_out_LR[0] #= angulo_avance
313     lr_goal1[1] = q_out_LR[1]
314     lr_goal1[2] = q_out_LR[2]
315     lr_goal1[3] = q_out_LR[3]
316
317     RM_actual = PyKDL.JntArray(nj_izq)
318     RM_actual[0] = rm_goal0[0]
319     RM_actual[1] = rm_goal0[1]
320     RM_actual[2] = rm_goal0[2]
321     RM_actual[3] = rm_goal0[3]
322     print "Inverse Kinematics"
323     fkssolver_RM.JntToCart(RM_actual, finalFrame_R)
324     q_init_RM = RM_actual # initial angles
325     desiredFrameRM = finalFrame_R
326     desiredFrameRM.p[0] = finalFrame_R.p[0]
327     desiredFrameRM.p[1] = finalFrame_R.p[1]
328     desiredFrameRM.p[2] = finalFrame_R.p[2] + altura_pata
329     print "Desired Position: ", desiredFrameRM.p
330     q_out_RM = PyKDL.JntArray(cadena_RM.getNrOfJoints())
331     ik_RM.CartToJnt(q_init_RM, desiredFrameRM, q_out_RM)
332     print "Output angles in rads: ", q_out_RM
333
334     rm_goal1[0] = q_out_RM[0] #= angulo_avance
335     rm_goal1[1] = q_out_RM[1]
336     rm_goal1[2] = q_out_RM[2]
337     rm_goal1[3] = q_out_RM[3]
338
339     LF_actual = PyKDL.JntArray(nj_izq)

```

```

340     LF_actual[0] = lf_goal0[0]
341     LF_actual[1] = lf_goal0[1]
342     LF_actual[2] = lf_goal0[2]
343     LF_actual[3] = lf_goal0[3]
344     print "Inverse Kinematics"
345     fkSolver_LF.JntToCart(LF_actual, finalFrame_L)
346     q_init_LF = LF_actual # initial angles
347     desiredFrameLF = finalFrame_L
348     desiredFrameLF.p[0] = finalFrame_L.p[0]
349     desiredFrameLF.p[1] = finalFrame_L.p[1]
350     desiredFrameLF.p[2] = finalFrame_L.p[2] + altura_pata
351     print "Desired Position: ", desiredFrameLF.p
352     q_out_LF = PyKDL.JntArray(cadena_LF.getNrOfJoints())
353     ik_LF.CartToJnt(q_init_LF, desiredFrameLF, q_out_LF)
354     print "Output angles in rads: ", q_out_LF
355
356     lf_goal1[0] = q_out_LF[0] #= angulo_avance
357     lf_goal1[1] = q_out_LF[1]
358     lf_goal1[2] = q_out_LF[2]
359     lf_goal1[3] = q_out_LF[3]
360
361     # Connect to the right arm trajectory action server
362     rospy.loginfo('Waiting for right arm trajectory controller
... ')
363
364     rr_client = actionlib.SimpleActionClient('/hexapodo/
pata_rr/follow_joint_trajectory',
365                                              FollowJointTrajectoryAction
)
366
367     rr_client.wait_for_server()
368
369     lm_client = actionlib.SimpleActionClient('/hexapodo/
pata_lm/follow_joint_trajectory',
370                                              FollowJointTrajectoryAction
)
371     lm_client.wait_for_server()
372
373     rf_client = actionlib.SimpleActionClient('/hexapodo/
pata_rf/follow_joint_trajectory',

```

```

374                                     FollowJointTrajectoryAction
375                                         )
376
377             lr_client = actionlib.SimpleActionClient('/hexapodo/
378                 pata_lr/follow_joint_trajectory',
379                                         FollowJointTrajectoryAction
380                                         )
381
382             rm_client = actionlib.SimpleActionClient('/hexapodo/
383                 pata_rm/follow_joint_trajectory',
384                                         FollowJointTrajectoryAction
385                                         )
386
387             lf_client = actionlib.SimpleActionClient('/hexapodo/
388                 pata_lf/follow_joint_trajectory',
389                                         FollowJointTrajectoryAction
390                                         )
391
392             lf_client.wait_for_server()
393
394             rospy.loginfo('...connected.')
395
396             # Create a single-point arm trajectory with the
397             # piernas_goal as the end-point
398             rr_trajectory1 = JointTrajectory()
399             rr_trajectory1.joint_names = piernas_joints_RR
400             rr_trajectory1.points.append(JointTrajectoryPoint())
401             rr_trajectory1.points[0].positions = rr_goal0
402             rr_trajectory1.points[0].velocities = [0.0 for i in
403                 piernas_joints_RR]
404             rr_trajectory1.points[0].accelerations = [0.0 for i in
405                 piernas_joints_RR]
406             rr_trajectory1.points[0].time_from_start = rospy.Duration
407                 (0.25)
408             rr_trajectory1.points.append(JointTrajectoryPoint())
409             rr_trajectory1.points[1].positions = rr_goal1
410             rr_trajectory1.points[1].velocities = [0.0 for i in
411                 piernas_joints_RR]

```

```

402     rr_trajectory1.points[1].accelerations = [0.0 for i in
403         piernas_joints_RR]
404     rr_trajectory1.points[1].time_from_start = rospy.Duration
405         (0.5)
406     rr_trajectory1.points.append(JointTrajectoryPoint())
407     rr_trajectory1.points[2].positions = rr_goal2
408     rr_trajectory1.points[2].velocities = [0.0 for i in
409         piernas_joints_RR]
410     rr_trajectory1.points[2].accelerations = [0.0 for i in
411         piernas_joints_RR]
412     rr_trajectory1.points[2].time_from_start = rospy.Duration
413         (0.75)
414     rr_trajectory1.points.append(JointTrajectoryPoint())
415     rr_trajectory1.points[3].positions = lr_goal0
416     rr_trajectory1.points[3].velocities = [0.0 for i in
417         piernas_joints_RR]
418     rr_trajectory1.points[3].accelerations = [0.0 for i in
419         piernas_joints_RR]
420     rr_trajectory1.points[3].time_from_start = rospy.Duration
421         (1.0)
422     #'''
423     lm_trajectory1 = JointTrajectory()
424     lm_trajectory1.joint_names = piernas_joints_LM
425     lm_trajectory1.points.append(JointTrajectoryPoint())
426     lm_trajectory1.points[0].positions = lm_goal0#[0,0,0,0]
427     lm_trajectory1.points[0].velocities = [0.0 for i in
428         piernas_joints_LM]
429     lm_trajectory1.points[0].accelerations = [0.0 for i in
430         piernas_joints_LM]
431     lm_trajectory1.points[0].time_from_start = rospy.Duration
432         (0.25)
433     lm_trajectory1.points.append(JointTrajectoryPoint())
434     lm_trajectory1.points[1].positions = lm_goal1 # [0,0,0,0]
435     lm_trajectory1.points[1].velocities = [0.0 for i in
436         piernas_joints_LM]
437     lm_trajectory1.points[1].accelerations = [0.0 for i in
438         piernas_joints_LM]
439     lm_trajectory1.points[1].time_from_start = rospy.Duration
440         (0.5)
441     lm_trajectory1.points.append(JointTrajectoryPoint())
442     lm_trajectory1.points[2].positions = lm_goal2 #[0,0,0,0]

```

```

429     lm_trajectory1.points[2].velocities = [0.0 for i in
430         piernas_joints_LM]
431     lm_trajectory1.points[2].accelerations = [0.0 for i in
432         piernas_joints_LM]
433     lm_trajectory1.points[2].time_from_start = rospy.Duration
434         (0.75)
435     lm_trajectory1.points.append(JointTrajectoryPoint())
436     lm_trajectory1.points[3].positions = rm_goal0 # [0,0,0,0]
437     lm_trajectory1.points[3].velocities = [0.0 for i in
438         piernas_joints_LM]
439     lm_trajectory1.points[3].accelerations = [0.0 for i in
440         piernas_joints_LM]
441     lm_trajectory1.points[3].time_from_start = rospy.Duration
442         (1.0)
443     #
444     rf_trajectory1 = JointTrajectory()
445     rf_trajectory1.joint_names = piernas_joints_RF
446     rf_trajectory1.points.append(JointTrajectoryPoint())
447     rf_trajectory1.points[0].positions = rf_goal0 # [0,0,0,0]
448     rf_trajectory1.points[0].velocities = [0.0 for i in
449         piernas_joints_RF]
450     rf_trajectory1.points[0].accelerations = [0.0 for i in
451         piernas_joints_RF]
452     rf_trajectory1.points[0].time_from_start = rospy.Duration
453         (0.25)
454     rf_trajectory1.points.append(JointTrajectoryPoint())
455     rf_trajectory1.points[1].positions = rf_goal1 # [0,0,0,0]
456     rf_trajectory1.points[1].velocities = [0.0 for i in
457         piernas_joints_RF]
458     rf_trajectory1.points[1].accelerations = [0.0 for i in
459         piernas_joints_RF]
460     rf_trajectory1.points[1].time_from_start = rospy.Duration
461         (0.5)
462     rf_trajectory1.points.append(JointTrajectoryPoint())
463     rf_trajectory1.points[2].positions = rf_goal2 # [0,0,0,0]
464     rf_trajectory1.points[2].velocities = [0.0 for i in
465         piernas_joints_RF]
466     rf_trajectory1.points[2].accelerations = [0.0 for i in
467         piernas_joints_RF]
468     rf_trajectory1.points[2].time_from_start = rospy.Duration
469         (0.75)

```

```

455     rf_trajectory1.points.append(JointTrajectoryPoint())
456     rf_trajectory1.points[3].positions = lf_goal0 # [0,0,0,0]
457     rf_trajectory1.points[3].velocities = [0.0 for i in
458         piernas_joints_RF]
459     rf_trajectory1.points[3].accelerations = [0.0 for i in
460         piernas_joints_RF]
461     rf_trajectory1.points[3].time_from_start = rospy.Duration
462         (1.0)
463     # ''
464     lr_trajectory1 = JointTrajectory()
465     lr_trajectory1.joint_names = piernas_joints_LR
466     lr_trajectory1.points.append(JointTrajectoryPoint())
467     lr_trajectory1.points[0].positions = lr_goal0#lr_goal0
468     lr_trajectory1.points[0].velocities = [0.0 for i in
469         piernas_joints_RR]
470     lr_trajectory1.points[0].accelerations = [0.0 for i in
471         piernas_joints_RR]
472     lr_trajectory1.points[0].time_from_start = rospy.Duration
473         (0.25)
474     lr_trajectory1.points.append(JointTrajectoryPoint())
475     lr_trajectory1.points[1].positions = lr_goal1
476     lr_trajectory1.points[1].velocities = [0.0 for i in
477         piernas_joints_RR]
478     lr_trajectory1.points[1].accelerations = [0.0 for i in
479         piernas_joints_RR]
480     lr_trajectory1.points[1].time_from_start = rospy.Duration
481         (0.5)
482     lr_trajectory1.points.append(JointTrajectoryPoint())
483     lr_trajectory1.points[2].positions = rr_goal2
484     lr_trajectory1.points[2].velocities = [0.0 for i in
485         piernas_joints_RR]
486     lr_trajectory1.points[2].accelerations = [0.0 for i in
487         piernas_joints_RR]
488     lr_trajectory1.points[2].time_from_start = rospy.Duration
489         (0.75)
490     lr_trajectory1.points.append(JointTrajectoryPoint())
491     lr_trajectory1.points[3].positions = rr_goal0 # lr_goal0
492     lr_trajectory1.points[3].velocities = [0.0 for i in
493         piernas_joints_RR]
494     lr_trajectory1.points[3].accelerations = [0.0 for i in
495         piernas_joints_RR]

```

```

482     lr_trajectory1.points[3].time_from_start = rospy.Duration
483         (1.0)
484     # ''
485     rm_trajectory1 = JointTrajectory()
486     rm_trajectory1.joint_names = piernas_joints_RM
487     rm_trajectory1.points.append(JointTrajectoryPoint())
488     rm_trajectory1.points[0].positions = rm_goal0#rm_goal0  #
489         [0,0,0,0]
490     rm_trajectory1.points[0].velocities = [0.0 for i in
491         piernas_joints_LM]
492     rm_trajectory1.points[0].accelerations = [0.0 for i in
493         piernas_joints_LM]
494     rm_trajectory1.points[0].time_from_start = rospy.Duration
495         (0.25)
496     rm_trajectory1.points.append(JointTrajectoryPoint())
497     rm_trajectory1.points[1].positions = rm_goal1 # [0,0,0,0]
498     rm_trajectory1.points[1].velocities = [0.0 for i in
499         piernas_joints_LM]
500     rm_trajectory1.points[1].accelerations = [0.0 for i in
501         piernas_joints_LM]
502     rm_trajectory1.points[1].time_from_start = rospy.Duration
503         (0.5)
504     rm_trajectory1.points.append(JointTrajectoryPoint())
505     rm_trajectory1.points[2].positions = lm_goal2 # [0,0,0,0]
506     rm_trajectory1.points[2].velocities = [0.0 for i in
507         piernas_joints_LM]
508     rm_trajectory1.points[2].accelerations = [0.0 for i in
509         piernas_joints_LM]
510     rm_trajectory1.points[2].time_from_start = rospy.Duration
511         (0.75)
512     rm_trajectory1.points.append(JointTrajectoryPoint())
513     rm_trajectory1.points[3].positions = lm_goal0 # rm_goal0
514         # [0,0,0,0]
515     rm_trajectory1.points[3].velocities = [0.0 for i in
516         piernas_joints_LM]
517     rm_trajectory1.points[3].accelerations = [0.0 for i in
518         piernas_joints_LM]
519     rm_trajectory1.points[3].time_from_start = rospy.Duration
520         (1.0)
521     # ''
522     lf_trajectory1 = JointTrajectory()

```

```

508     lf_trajectory1.joint_names = piernas_joints_LF
509     lf_trajectory1.points.append(JointTrajectoryPoint())
510     lf_trajectory1.points[0].positions = lf_goal0#lf_goal0  #
511         [0,0,0,0]
512     lf_trajectory1.points[0].velocities = [0.0 for i in
513         piernas_joints_RF]
514     lf_trajectory1.points[0].accelerations = [0.0 for i in
515         piernas_joints_RF]
516     lf_trajectory1.points[0].time_from_start = rospy.Duration
517         (0.25)
518     lf_trajectory1.points.append(JointTrajectoryPoint())
519     lf_trajectory1.points[1].positions = lf_goal1 # [0,0,0,0]
520     lf_trajectory1.points[1].velocities = [0.0 for i in
521         piernas_joints_RF]
522     lf_trajectory1.points[1].accelerations = [0.0 for i in
523         piernas_joints_RF]
524     lf_trajectory1.points.append(JointTrajectoryPoint())
525     lf_trajectory1.points[2].positions = rf_goal2 # [0,0,0,0]
526     lf_trajectory1.points[2].velocities = [0.0 for i in
527         piernas_joints_RF]
528     lf_trajectory1.points[2].accelerations = [0.0 for i in
529         piernas_joints_RF]
530     lf_trajectory1.points[2].time_from_start = rospy.Duration
531         (0.75)
532     lf_trajectory1.points.append(JointTrajectoryPoint())
533     lf_trajectory1.points[3].positions = rf_goal0 # lf_goal0
534         # [0,0,0,0]
535     lf_trajectory1.points[3].velocities = [0.0 for i in
536         piernas_joints_RF]
537     lf_trajectory1.points[3].accelerations = [0.0 for i in
538         piernas_joints_RF]
539     lf_trajectory1.points[3].time_from_start = rospy.Duration
540         (1.0)
541     # ''
542     # Send the trajectory to the arm action server
543     rospy.loginfo('Moving the arm to goal position... ')
544

```

```

535     # Create an empty trajectory goal
536     rr_goal = FollowJointTrajectoryGoal()
537     lm_goal = FollowJointTrajectoryGoal()
538     rf_goal = FollowJointTrajectoryGoal()
539
540     lr_goal = FollowJointTrajectoryGoal()
541     rm_goal = FollowJointTrajectoryGoal()
542     lf_goal = FollowJointTrajectoryGoal()
543     # Set the trajectory component to the goal trajectory
544     # created above
545     rr_goal.trajectory = rr_trajectory1
546     lm_goal.trajectory = lm_trajectory1
547     rf_goal.trajectory = rf_trajectory1
548     lr_goal.trajectory = lr_trajectory1
549     rm_goal.trajectory = rm_trajectory1
550     lf_goal.trajectory = lf_trajectory1
551     # Specify zero tolerance for the execution time
552     rr_goal.goal_time_tolerance = rospy.Duration(0.01)
553     lm_goal.goal_time_tolerance = rospy.Duration(0.01)
554     rf_goal.goal_time_tolerance = rospy.Duration(0.01)
555     lr_goal.goal_time_tolerance = rospy.Duration(0.01)
556     rm_goal.goal_time_tolerance = rospy.Duration(0.01)
557     lf_goal.goal_time_tolerance = rospy.Duration(0.01)
558     # Send the goal to the action server
559     rr_client.send_goal(rr_goal)
560     lm_client.send_goal(lm_goal)
561     rf_client.send_goal(rf_goal)
562     #rr_client.wait_for_result(rospy.Duration(5.0))
563     print 'Resultado recibido'
564     lr_client.send_goal(lr_goal)
565     rm_client.send_goal(rm_goal)
566     lf_client.send_goal(lf_goal)
567
568     rr_client.wait_for_result()
569     rr_client.send_goal(rr_goal)
570     lm_client.send_goal(lm_goal)
571     rf_client.send_goal(rf_goal)
572     # rr_client.wait_for_result(rospy.Duration(5.0))
573     print 'Resultado recibido'
574     lr_client.send_goal(lr_goal)
575     rm_client.send_goal(rm_goal)

```

```
575     lf_client.send_goal(lf_goal)
576
577     rr_client.wait_for_result()
578     rr_client.send_goal(rr_goal)
579     lm_client.send_goal(lm_goal)
580     rf_client.send_goal(rf_goal)
581 # rr_client.wait_for_result(rospy.Duration(5.0))
582     print 'Resultado recibido'
583     lr_client.send_goal(lr_goal)
584     rm_client.send_goal(rm_goal)
585     lf_client.send_goal(lf_goal)
586
587     rr_client.wait_for_result()
588     rr_client.send_goal(rr_goal)
589     lm_client.send_goal(lm_goal)
590     rf_client.send_goal(rf_goal)
591 # rr_client.wait_for_result(rospy.Duration(5.0))
592     print 'Resultado recibido'
593     lr_client.send_goal(lr_goal)
594     rm_client.send_goal(rm_goal)
595     lf_client.send_goal(lf_goal)
596
597     rr_client.wait_for_result()
598     rr_client.send_goal(rr_goal)
599     lm_client.send_goal(lm_goal)
600     rf_client.send_goal(rf_goal)
601 # rr_client.wait_for_result(rospy.Duration(5.0))
602     print 'Resultado recibido'
603     lr_client.send_goal(lr_goal)
604     rm_client.send_goal(rm_goal)
605     lf_client.send_goal(lf_goal)
606
607     rr_client.wait_for_result()
608     rr_client.send_goal(rr_goal)
609     lm_client.send_goal(lm_goal)
610     rf_client.send_goal(rf_goal)
611 # rr_client.wait_for_result(rospy.Duration(5.0))
612     print 'Resultado recibido'
613     lr_client.send_goal(lr_goal)
614     rm_client.send_goal(rm_goal)
615     lf_client.send_goal(lf_goal)
```

```
616
617     rr_client.wait_for_result()
618     rr_client.send_goal(rr_goal)
619     lm_client.send_goal(lm_goal)
620     rf_client.send_goal(rf_goal)
621     # rr_client.wait_for_result(rospy.Duration(5.0))
622     print 'Resultado recibido'
623     lr_client.send_goal(lr_goal)
624     rm_client.send_goal(rm_goal)
625     lf_client.send_goal(lf_goal)
626
627     rr_client.wait_for_result()
628     rr_client.send_goal(rr_goal)
629     lm_client.send_goal(lm_goal)
630     rf_client.send_goal(rf_goal)
631     # rr_client.wait_for_result(rospy.Duration(5.0))
632     print 'Resultado recibido'
633     lr_client.send_goal(lr_goal)
634     rm_client.send_goal(rm_goal)
635     lf_client.send_goal(lf_goal)
636
637     rr_client.wait_for_result()
638     rr_client.send_goal(rr_goal)
639     lm_client.send_goal(lm_goal)
640     rf_client.send_goal(rf_goal)
641     # rr_client.wait_for_result(rospy.Duration(5.0))
642     print 'Resultado recibido'
643     lr_client.send_goal(lr_goal)
644     rm_client.send_goal(rm_goal)
645     lf_client.send_goal(lf_goal)
646
647     rr_client.wait_for_result()
648     rr_client.send_goal(rr_goal)
649     lm_client.send_goal(lm_goal)
650     rf_client.send_goal(rf_goal)
651     # rr_client.wait_for_result(rospy.Duration(5.0))
652     print 'Resultado recibido'
653     lr_client.send_goal(lr_goal)
654     rm_client.send_goal(rm_goal)
655     lf_client.send_goal(lf_goal)
656     #lr_client.wait_for_result(rospy.Duration(5.0))
```

```
657     #rr_client.send_goal(rr_goal)
658     #lm_client.send_goal(lm_goal)
659     #rf_client.send_goal(rf_goal) ''
660
661     if not sync:
662         # Wait for up to 5 seconds for the motion to complete
663         rr_client.wait_for_result(rospy.Duration(5.0))
664
665         rr_client.wait_for_result()
666         print rr_client.get_result()
667
668
669         rospy.loginfo('... done')
670
671 if __name__ == '__main__':
672     try:
673         TrajectoryDemo()
674     except rospy.ROSInterruptException:
675         pass
```

Referencias