

# 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 E.4.5 IMPLEMENTACIÓN EN UNA HERRAMIENTA SOFTWARE.

---

### RESUMEN DEL DOCUMENTO

En este entregable se detalla la implementación de la herramienta software definida en el entregable anterior. Esta herramienta es capaz de generar un robot en base a unas especificaciones definidas por el usuario.

---

19 de enero de 2017

# Índice

<b>1. Introducción</b>	<b>1</b>
<b>2. Aplicación de usuario de diseño automático de robots</b>	<b>2</b>
2.1. Elección de un robot . . . . .	3
2.1.1. Algunos aspectos acerca de pySUMO . . . . .	3
2.1.2. Interfaz de usuario . . . . .	4
2.1.3. Explicación del código . . . . .	6
2.1.4. Ejemplo de uso . . . . .	10
2.2. Parametrización de un robot . . . . .	13
2.2.1. Manipulador: robindArm . . . . .	15
2.3. Robot con ruedas . . . . .	25
<b>3. Programas para el estudio dinámico del robot</b>	<b>27</b>
3.1. Introducción y uso de MATLAB . . . . .	27
3.2. Uso de Robotics Toolbox . . . . .	28
3.2.1. Funciones básicas . . . . .	28
3.2.2. Métodos para la definición de robots . . . . .	31
3.2.3. Consideraciones para la definición de robots . . . . .	32
3.3. Robot manipulador . . . . .	34
3.4. Cálculo automático de las dimensiones de los eslabones . . . . .	36
3.4.1. Primera estrategia . . . . .	37
3.4.2. Segunda estrategia . . . . .	37
3.5. Optimización de los eslabones . . . . .	40
3.6. Caracterización automática de las propiedades físicas de los eslabones	40
3.7. Cálculo del par en las articulaciones . . . . .	43
3.7.1. Cálculo del par estático . . . . .	44
3.7.2. Cálculo del par dinámico . . . . .	46
3.8. Programa final para el diseño automático de manipuladores de 3 grados de libertad . . . . .	49
<b>4. Anexo A. Código de la aplicación en Python</b>	<b>53</b>
4.1. App principal . . . . .	53
4.2. Código ventana principal . . . . .	64

4.3.	Ventanas auxiliares . . . . .	66
4.3.1.	Manipulador . . . . .	66
4.3.2.	Robot móvil . . . . .	70
4.4.	Generación STL . . . . .	72
4.4.1.	Eslabones Manipulador . . . . .	72
4.4.2.	Ruedas robot móvil . . . . .	74
4.4.3.	Xacro Robot móvil . . . . .	76
4.5.	Parametrización URDF . . . . .	81
4.5.1.	Parametrizar Hexápodo . . . . .	81
4.6.	Xacro Hexápodo . . . . .	83
4.6.1.	Parametrizar Humanoide . . . . .	88
4.7.	Xacro Humanoide . . . . .	91
4.7.1.	Parametrizar Serpiente . . . . .	101
4.7.2.	Xacro Serpiente . . . . .	103
<b>5.</b>	<b>Anexo B. Programa final para el diseño automático de manipuladores de 3 grados de libertad. Código MATLAB</b>	<b>112</b>

# 1. Introducción

Este documento forma parte de la documentación asociada al Paquete de Trabajo número 4 (PT4) del proyecto BOTBLOQ. El PT4 tiene como objetivo el diseño, la fabricación y el control de robots modulares. En los anteriores entregables se han realizado:

**E.4.1. Informe detallado del estado del arte.** Este documento contiene una clasificación de los robots modulares preexistentes y un análisis de los requisitos exigibles a una arquitectura robótica modular con el fin de definir las funcionalidades que debe reunir para que resulte satisfactoria según los objetivos del proyecto.

**E.4.2. Diseño de la arquitectura modular.** En este entregable se presentaron los módulos confeccionados para la implementación de estructuras robóticas, incluyendo características mecánicas, eléctricas, electrónicas y lógicas. En él también se describen los modos de generación permitidos (manual, predefinido y semi-automático), y se muestran ejemplos de robots configurables (predefinidos) con esta arquitectura.

**E.4.3. Proceso metodológico para el diseño de un robot modular.** Este entregable define las etapas a seguir para el diseño y programación de un robot en función de un conjunto de entradas.

**E.4.4. Definición del alcance de la herramienta de software.** Este entregable indica explícitamente cuáles son las posibilidades de la herramienta de software a realizar para ayuda al diseño de estructuras robóticas.

En este entregable se detallará la implementación de la herramienta software definida en el entregable anterior. Esta herramienta genera un robot a partir de unas especificaciones definidas por el usuario interactuando con la interfaz del programa.

## 2. Aplicación de usuario de diseño automático de robots

Programar un sistema capaz de diseñar robots por sí mismo es el principal objetivo de esta parte del proyecto Botbloq. Este capítulo, en el que se mostrará cómo se ha conseguido el objetivo, está dividido en dos partes: Elección del Robot y Parametrización del mismo. Por el momento, se ha diseñado un sistema simple que extrae información de la ontología realizando consultas permitiendo al usuario elegir un robot que haga lo que él quiere. Una vez el robot ha sido elegido, el proceso de parametrización comienza. En él, el usuario podrá modificar algunos de los parámetros del robot elegido dependiendo del uso específico que quiera hacer de él (tamaño, partes del robot, etc.). De esta manera, en la primera parte, el usuario elige un robot capaz de realizar la acción que él quiere (agarrar un objeto y desplazarlo, por ejemplo). Mientras que en la segunda parte se configuran los parámetros específicos del robot dependiendo de ciertos rasgos específicos de la acción a realizar (forma del objeto, distancia a la que se quiere desplazar, etc.).

La aplicación está escrita en Python y necesita la versión 3 porque pySUMO, herramienta utilizada para tratar con la ontología, está escrita en ese lenguaje. Las búsquedas se llevan a cabo usando el módulo `pysumo` (<https://github.com/pySUMO/pysumo>). La interfaz gráfica usa PySide. De forma resumida, el funcionamiento consiste en los siguientes pasos:

1. En primer lugar se selecciona la(s) tarea(s) que debe realizar el robot.
2. A continuación se solicita la búsqueda, que tiene tres pasos:
  - Primero se buscan las partes que necesita el robot para poder realizar las acciones seleccionadas.
  - A continuación se buscan los robots que contienen las partes necesarias.
  - Por último se comprueba si alguno de los robots está en todas las búsquedas, es decir, si puede realizar todas las acciones.
3. Tras determinar si hay uno o varios robots que pueden realizar las acciones, se pasa a la parametrización<sup>1</sup>:

<sup>1</sup>Actualmente la aplicación permite parametrizar un robot manipulador “robindArm” y un

- Se selecciona el robot a parametrizar y se pulsa el botón de parametrizar robot.
  - Se introducen los datos parametrizables necesarios para definir el robot.
4. Se genera la estructura física y los controladores del robot.

## 2.1. Elección de un robot

En este apartado se explica en profundidad la primera parte del proceso en la que se elegirá un robot capaz de realizar una acción determinada de entre las posibles.

### 2.1.1. Algunos aspectos acerca de pySUMO

Lo primero que debemos entender es que pySUMO fue desarrollado simplemente como IDE para ontologías escritas en SUO-KIF, por lo que las opciones que proporciona son limitadas (por ejemplo, pySUMO no tiene una función ya definida que realice consultas semánticas). En este apartado, se explicarán las dos únicas funciones (escritas en Python) que ha sido necesario crear para realizar consultas sobre la ontología.

Una vez escrita la ontología, todos sus axiomas quedan relacionados entre sí en forma de grafo. Se pueden tener tantas relaciones como se quieran, sin embargo, en SUMO las más relevantes son dos: subclase e instancia ('humano' es una subclase de 'animal' y 'Marta' podría ser una instancia de 'humano'). Como hemos dicho, pySUMO no permite realizar consultas pero sí que tiene funciones que, por ejemplo, permiten recorrer el grafo siguiendo las relaciones entre los axiomas. Dado un axioma y determinando el tipo de relación, es posible recorrer el grafo y obtener todos los axiomas que se relacionan con él. Un formato típico para almacenar y transmitir datos en ontologías escritas (p.ej. en notación RDF, *Resource Description Framework*) es usar *triples* con la estructura sujeto-predicado-objeto. En este caso, podemos utilizar este término para entender cómo recorreremos el grafo. Tendremos un axioma **sujeto** y otro **objeto** que se relacionarán a través del predicado, con pySUMO podremos recorrer el grafo en dos direcciones definidas según la siguiente notación:

robot con ruedas: tipo diferencial y tipo *skid-steer*.

- **Hacia arriba:** Partimos del **objeto** y obtenemos todos los axiomas **sujeto** que se relacionan con el **objeto** a través del **predicado**. Ejemplo: Imaginemos que queremos realizar una búsqueda *hacia arriba* de todos los axiomas relacionados a través del **predicado** 'subclase' con el axioma 'humano'. Con pySUMO obtendremos información necesaria para saber que 'humano' es una subclase de 'animal', que a su vez es subclase de 'organismo' y así hasta llegar al primer axioma (raíz) de nuestra ontología 'Entidad'. Por lo tanto, esto sería equivalente a buscar las 'superclases' de las que depende un axioma.
- **Hacia abajo:** Esta vez es al contrario, partimos del **sujeto** y obtenemos todos los axiomas **objeto** relacionados con el **sujeto** a través del **predicado**. Ejemplo: Supongamos que queremos realizar una búsqueda *hacia abajo* de todos los axiomas relacionados a través del **predicado** 'subclase' con el axioma 'humano'. Con pySUMO obtendremos información necesaria para saber que 'humano' se divide en varias subclases: 'hombre', 'mujer', 'humano adulto', 'joven', etc. Y estos axiomas a su vez pueden tener subclases. Por lo tanto, este tipo de consulta nos devuelve las subclases de un axioma y las subclases de sus subclases (y así hasta llegar al último axioma del grafo).

En nuestro sistema, se han implementado dos funciones que permiten recorrer el grafo y extraer la información que necesitemos, en una y otra dirección. Las hemos llamado *query* y *reversedQuery* según si la búsqueda es hacia abajo o hacia arriba.

### 2.1.2. Interfaz de usuario

Primero de todo se mostrará y se explicará el aspecto visual de la aplicación, para pasar luego a una prueba de concepto utilizando un ejemplo. En la Figura 1 se muestra el aspecto de la ventana principal de la aplicación en la que se pueden ver las partes en las que se divide (de la uno a la seis).

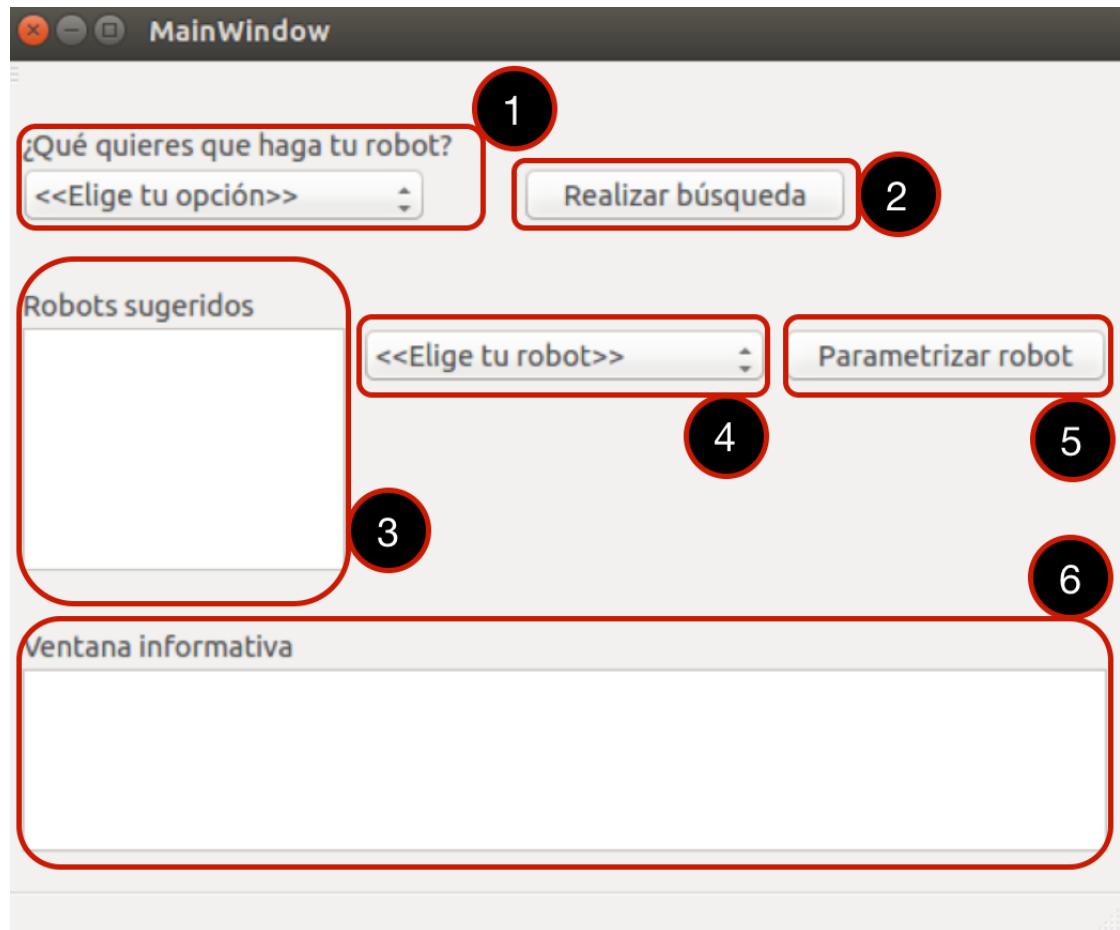


Figura 1: CORE app: Main Window.

1. Menú desplegable que permite al usuario elegir la acción que el robot deberá ser capaz de realizar.
2. Botón que, siempre que la acción haya sido elegida, debe ser pulsado para buscar robots capaces de realizarla.
3. Ventana que muestra los robots sugeridos por el sistema.
4. Menú desplegable en el que es posible elegir el robot que se desee de entre los sugeridos.
5. Botón que se deberá pulsar una vez el robot se ha elegido, será entonces cuando el proceso de parametrización comenzará, abriéndose una ventana

nueva que dependerá del robot elegido.

6. Ventana para mostrar información relevante al programador (útil para depurar).

### 2.1.3. Explicación del código

El argumento de entrada de la aplicación es una acción, el robot debe ser capaz de realizarla. Las acciones están definidas en la ontología de manera que sabemos qué partes estructurales son necesarias para que el robot realice dichas acciones (para ello se utiliza la relación 'structuralrequirement'). Por ejemplo, la acción de 'andar' necesita la parte estructural 'piernas', y para 'agarrar' se necesita una 'pinza'. Por lo tanto, por medio de consultas semánticas (utilizando la función 'reversedQuery' y el **predicado** 'structuralrequirement') conseguiremos saber qué partes estructurales son necesarias para desarrollar la acción elegida por el usuario. Una vez lo sabemos, volvemos a consultar la ontología para saber cuáles de los robots pre-definidos tienen en su estructura las partes necesarias (utilizando la función 'reversedQuery' y el **predicado** 'uses'). Todo robot que use, y por lo tanto tenga entre sus partes, las partes estructurales necesarias, serán capaces de realizar la acción que demanda el usuario y serán por ello propuestos como posibles opciones. Finalmente, el usuario deberá elegir uno entre los robots propuestos y comenzar el proceso de parametrización del mismo.

Tras la explicación realizada en la Sección 2.1.2 se puede inferir que el proceso de elección del robot estará dividido en dos pasos (relacionados con el evento producido al pulsar cada uno de los dos botones existentes). Concretamente, con la pulsación de cada uno de los botones: Realizar búsqueda y Parametrizar robot, se ejecutará una función en Python asociada a dicho evento: '*searchingStructuralRequirement*' y '*customizingRobot*', respectivamente. A continuación, se explicarán estos dos pasos utilizando una breve explicación en forma de texto. Todo el código del que se hablará puede encontrarse dentro del archivo 'CORE\_app.py'. Debe tenerse en cuenta que no todo el código que aparece en dicho archivo será explicado (se evitará la definición de algunas variables y funciones que se realiza al inicio).

Es interesante notar que para abrir la ventana con la interfaz de usuario es necesario ejecutar el archivo Python (utilizar versión 3 de Python) del que se ha

hablado en el párrafo anterior, 'CORE\_app.py'. Para ello, nos movemos desde la terminal hasta la ruta en la que se encuentre el archivo y escribimos lo siguiente:

```
python3 CORE_app.py
```

**1: Elección de la acción** El primer paso está asociado a la función '*searchingStructuralRequirement*', en ella se utilizará una variable de tipo diccionario definida al inicio del código, '*actions*'. Dado que la ontología está escrita en Inglés y nuestra aplicación en Español, era necesario establecer una relación entre las acciones en un idioma y otro. En el código de la función se sigue la siguiente secuencia de acciones:

1. Se almacena el texto actual del menú desplegable de las acciones (1 en la Figura 1) en una variable y después se obtiene su equivalente en Inglés (utilizando el diccionario citado antes).

```
actionBotbloq = self.RobotActions.currentText()  
actionOntology = actions[actionBotbloq]
```

2. Si el texto se corresponde con la opción vacía (opción por defecto) entonces se mostrará un mensaje de aviso por la pantalla y el proceso no continuará hasta que el usuario elija una acción. Si no se da esta condición, el proceso continuará.

```
if actionOntology == 'Sin elegir':  
    self.thereAreNOSomeFoundedRobots = True  
    self.searchResult.clear()  
    msg = QMessageBox()  
    msg.setIcon(QMessageBox.Warning)  
    msg.setText('Por favor, seleccione una acción antes  
de continuar.')
```

```
msg.exec_()
```

3. Se consulta en la ontología qué partes estructurales son necesarias para realizar la acción elegida por el usuario.

```
queriedGraph = reversedQuery('structuralrequirement',  
    actionOntology)
```

```
requirements = []  
for i in range(0, len(queriedGraph)):  
    requirements.insert(i, queriedGraph[i])
```

4. De manera complementaria (aunque en esta versión no se haga uso de esta información) se obtienen de la ontología otras acciones que también necesitan las partes estructurales encontradas. Esto se hace con vistas a complementar la información aportada por el usuario, proponiéndole otras acciones diferentes a la elegida.

```
complementaryAxioms = []  
for i in range(0,len(requirements)):  
    queriedGraph1 = query('structuralrequirement',  
        requirements[i])  
    complementaryAxioms = queriedGraph1[0:len(queriedGraph1)]
```

5. Se consulta la ontología para encontrar todos los robot que utilizan o tienen las partes estructurales necesarias. Se muestra el resultado por la ventana dirigida a ello (3 en la Figura 1).

```
x = len(requirements)
```

```
for i in range(0,x):  
  
    queriedGraphRobots = reversedQuery('uses',  
    requirements[i])  
  
    self.requiredRobots =  
        queriedGraphRobots[0:len(queriedGraphRobots)]  
  
    x1 = len(self.requiredRobots)  
  
    for j in range(0,x1):  
  
        self.suggestedRobots.append('%d %s' % (j,  
        self.requiredRobots[j]))
```

### Paso 2: Elección del robot e inicio del proceso de parametrización

Este paso se inicia con la ejecución de la función '*customizingRobot*' tras pulsar el botón 'Parametrizar robot' (5 en la Figura 1).

1. Dado que el método de elección del robot se realiza mediante un número identificador que puede ir del cero al nueve y que el número de robots sugeridos rara vez supere la decena. Se comprueba que el número asociado al robot elegido no es mayor que el número de robots sugeridos. En el caso de que lo fuera, saltaría una ventana de aviso y el proceso terminaría, si no es así, el proceso continúa.

```
numberChosenRobot = int(textChosenRobot)  
  
if numberChosenRobot >len(self.requiredRobots)-1:  
  
    msg = QMessageBox()  
  
    msg.setText('Por favor, seleccione un robot de entre los sugeridos  
antes de continuar. Recuerde elegir el número correctamente.')  
    msg.exec_()
```

2. Se abre una ventana de confirmación en la que se muestra el robot elegido y se le permite al usuario parar el proceso. Si el usuario decide continuar, se ejecutará una función asociada al robot elegido (que variará en función

del robot). Para ello se utiliza el diccionario '*definedRobots*', en el que hay punteros a funciones asociadas a los distintos robots. Estas funciones se ocuparán de abrir una ventana secundaria en la que poder parametrizar el robot en cuestión.

```
msg = QMessageBox()  
  
msg.setIcon(QMessageBox.Warning)  
  
msg.setText('El robot elegido es: %s. ¿Quieres comenzar el proceso  
de parametrización de este robot? Si prefieres elegir otro robot,  
cancela el proceso.' % self.requiredRobots[numberChosenRobot])  
  
msg.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)  
  
RetVal = msg.exec_()  
  
  
if RetVal == QMessageBox.Ok:  
  
  
    functionToCall = definedRobots[self.requiredRobots[numberChosenRobot]]  
  
    functionToCall(self)
```

#### 2.1.4. Ejemplo de uso

Llega el momento de ver un ejemplo de uso. Se mostrará utilizando imágenes, con el fin de que quede mucho más claro.

##### PASO 1 - Eligiendo una acción

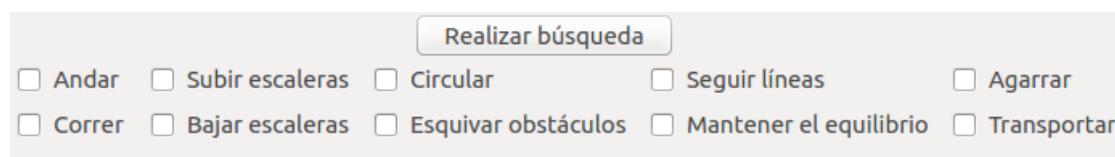


Figura 2: COREapp: Ejemplo - Eligiendo una acción.

**PASO 2 - Robots sugeridos** En la Figura 3 podemos ver robots capaces de subir escaleras.

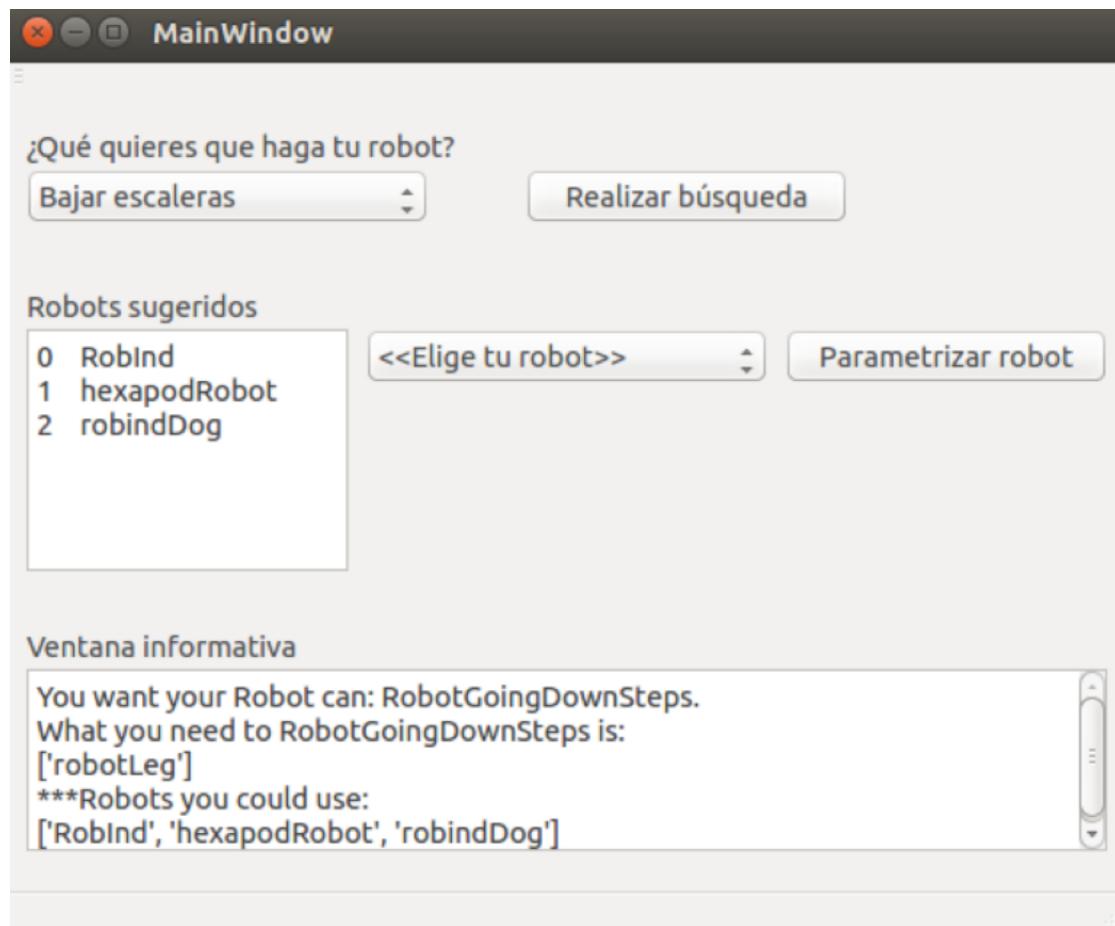


Figura 3: COREapp: Ejemplo - Robots sugeridos.

### PASO 3 - Eligiendo un robot

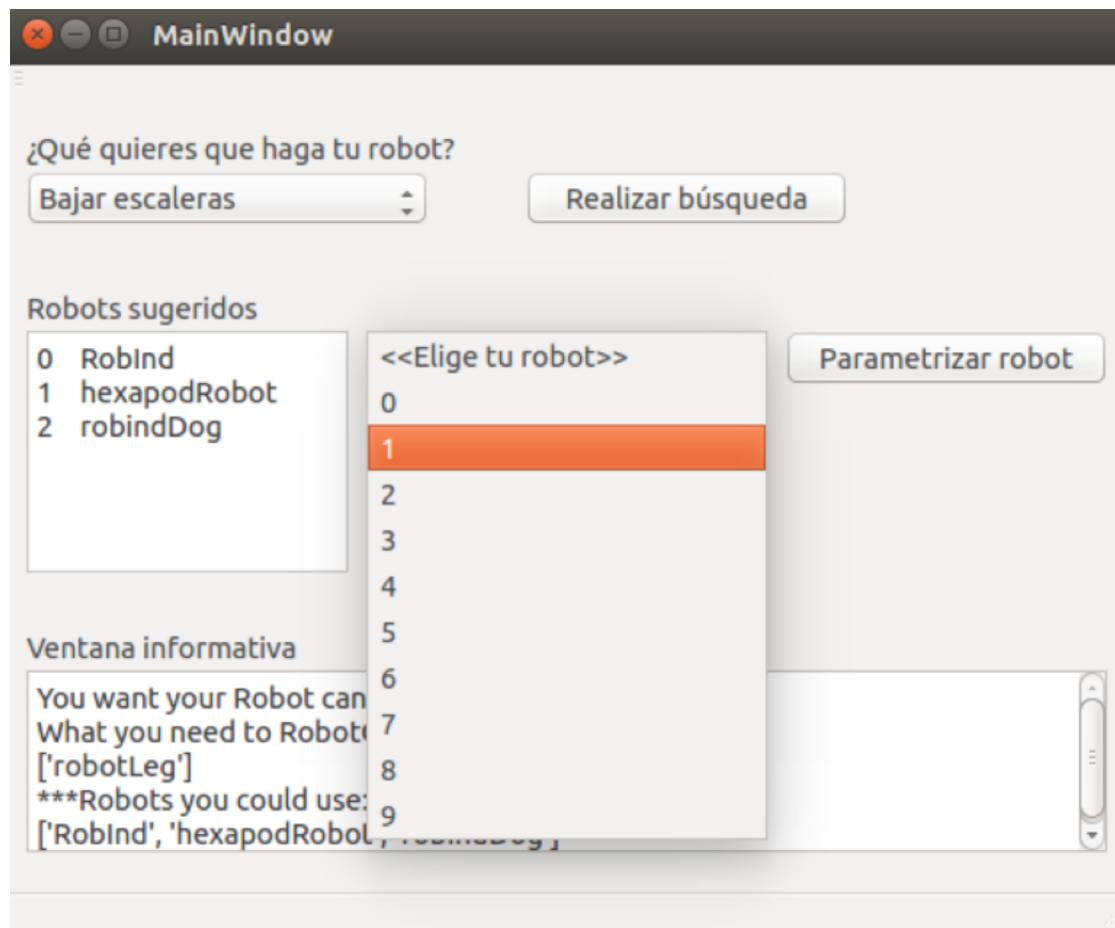


Figura 4: COREapp: Ejemplo - Eligiendo un robot.

#### PASO 4 - ¡Listos para parametrizar!

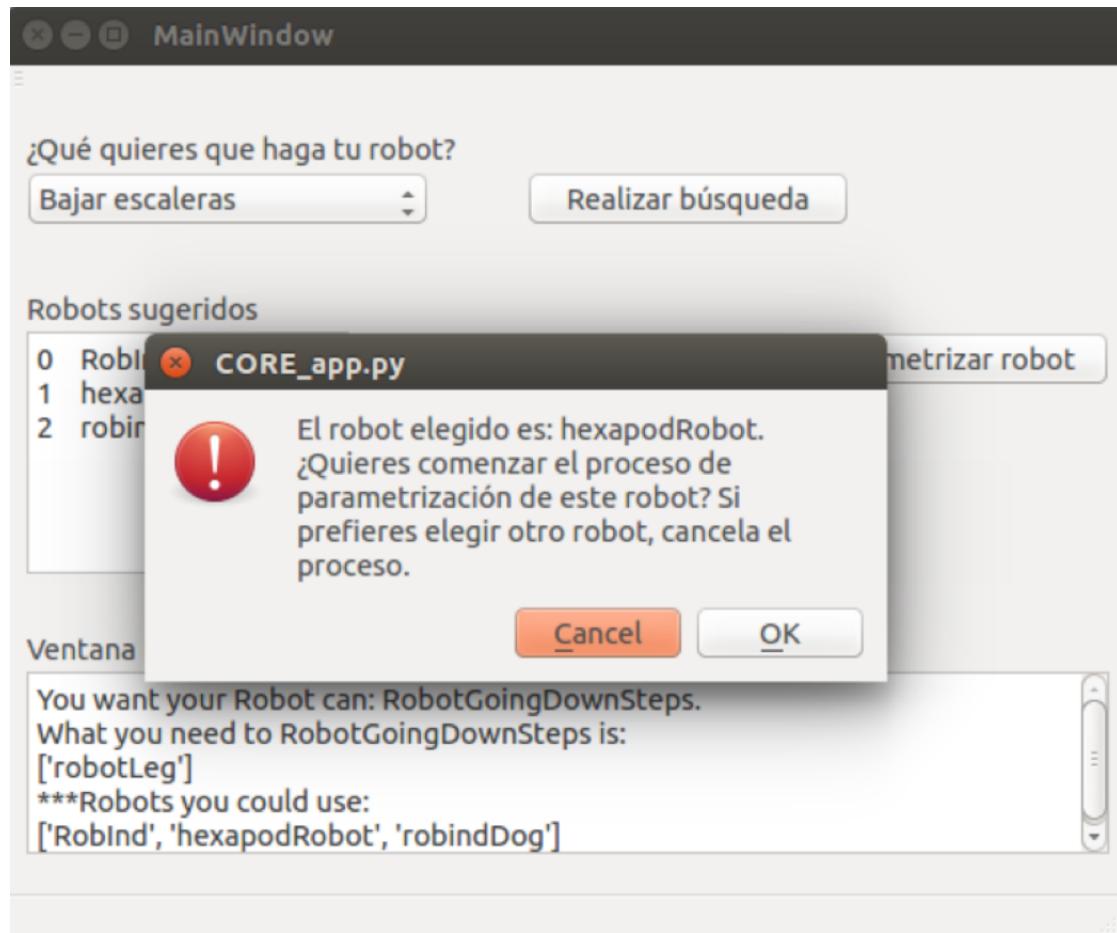


Figura 5: COREapp: Ejemplo - ¡Listos para parametrizar!.

## 2.2. Parametrización de un robot

Una vez el usuario ha elegido un robot es el momento de parametrizarlo. Cada robot (manipulador, hexápodo, humanoide, etc.) tendrá unos parámetros propios, es por ello que se ha decidido diseñar una sub-ventana específica para parametrizar cada uno de ellos. La aplicación abrirá esta ventana secundaria cuando el proceso de parametrización comience.

Hasta el momento sólo hemos hablado del diseño de una aplicación que permite interactuar con el usuario y extraer información de esta interacción. Ahora, cuando el proceso de parametrización del robot comienza, es el momento en el

que se utilizará esa información para diseñar el robot y definirlo adecuadamente para que sea utilizado en ROS (software utilizado para el control de los robots). A continuación se presenta la aplicación desarrollada, la cual permite cumplir con todos los objetivos propuestos en la memoria del proyecto, si bien está siendo ampliada y mejorada para añadir nuevas funcionalidades. Las partes de la aplicación desarrollada son las siguientes:

- El primer paso será diseñar la morfología del robot, se trata de realizar los cálculos matemáticos pertinentes utilizando la información inferida del usuario. En el futuro se pretende realizar este trabajo utilizando diversas herramientas incluídas en ROS. Sin embargo, dada su facilidad de uso, por el momento se opta por utilizar Matlab<sup>2</sup> y el conjunto de herramientas (*Robotics Toolbox*) orientadas a la robótica diseñado por Peter Corke<sup>3</sup>. Con Matlab se realizarán los cálculos matemáticos necesarios y se configurará la forma final del robot elegido por el usuario (este proceso variará dependiendo del robot elegido).
- Una vez diseñado el robot es necesario definirlo en el formato adecuado para que ROS pueda utilizarlo y podamos controlar el robot, concretamente, ese formato es **URDF**. Si bien la manera de obtener estos archivos es relativamente sencilla si lo hacemos manualmente (desde el *.stl* generado en un programa CAD), automatizar este proceso requiere el uso de otro tipo de archivos que ROS proporciona (*Xacro*). Con ellos, ROS nos permite parametrizar un archivo **URDF** previamente definido. De esta manera, definiremos archivos **URDF** para cada módulo de nuestro kit de robótica (pasivos y activos) y, generando un archivo *xacro* una vez se ha diseñado el robot, se podrá obtener el **URDF** del robot final.

Como ya se ha dicho, existirá una sub-ventana (y por lo tanto, un proceso) para cada robot pre-definido. En las siguientes secciones se analizará con detalle el proceso de parametrización de cada robot por separado, en el que se estudiarán las siguientes partes:

<sup>2</sup>Herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux.

<sup>3</sup><http://petercorke.com/Robotics\Toolbox.html>

- Sub-ventana en la que el usuario proporciona la información necesaria para configurar el robot en cuestión.
- Diseño del robot utilizando la información recibida.
- Definición del robot adecuada al formato requerido en ROS (**URDF**).
- Ejemplo de uso.

### 2.2.1. Manipulador: robindArm

En esta sección se muestran las distintas partes implicadas en el proceso de parametrización de un robot manipulador (brazo robótico). Es necesario tener instalado Matlab para realizar la parametrización ya que se usa la herramienta *Robotics Toolbox*, según se explica en la Sección 3. Cuando se carga la ventana de parametrización es necesario introducir los parámetros de relevancia, que podrían ser cuatro:

- Forma del objeto a manipular.
- Peso del objeto.
- Espacio de trabajo del robot (relacionado con la longitud del robot).
- Número de grados de libertad del robot, que variará de tres a seis dependiendo de si el usuario quiere que el robot pueda posicionar o si también es necesario orientar el objeto.

**Ventana para un Manipulador** Por el momento, esta ventana tiene cinco partes fundamentales (véase la Figura 6), aunque está sujeta a cambios y se debe seguir trabajando sobre ella, sin embargo, cuenta con control de fallos.

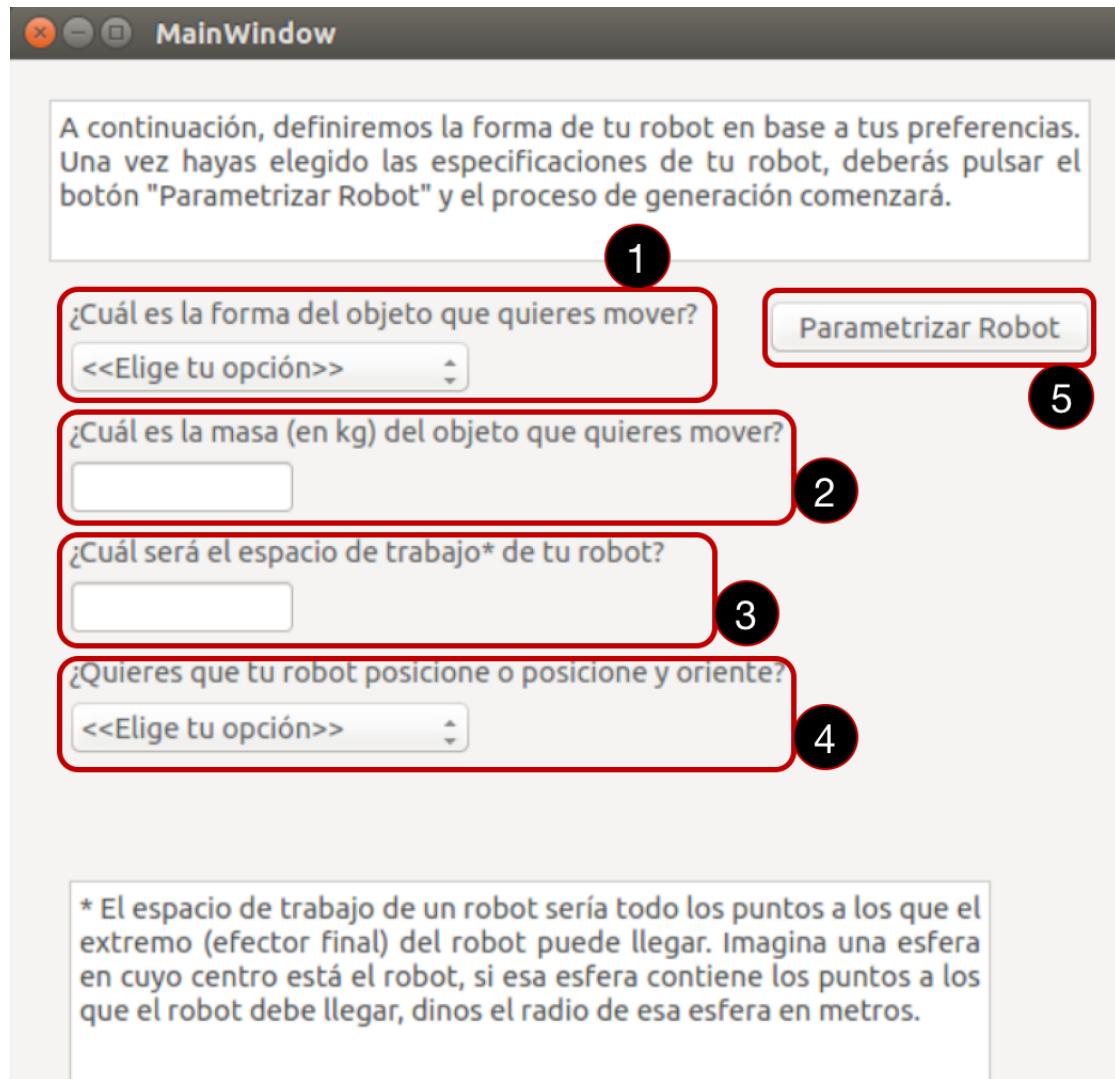


Figura 6: COREapp: Parametrización de un Manipulador - Ventana Principal

1. Menú desplegable que permite elegir la forma del objeto que manipulará el robot. En este menú se ofrecen unas opciones fijas, aunque se pueden ampliar.
2. Cuadro de texto en el que escribir la masa (en kg) del objeto a manipular.
3. Espacio de trabajo del robot o un dato de longitud máxima del brazo robótico. Es un concepto complejo para explicarlo a un niño, por lo que habría que adecuar su definición.

4. Menú desplegable que permite elegir el número de grados de libertad del robot (las opciones posibles de momento son sólo dos: tres o seis grados).
5. Botón para dar comienzo al proceso de parametrización.

Cuando se han llenado los campos correctamente se lanza el motor de Matlab que proporciona para usarse desde Python y se ejecuta la toolbox necesaria, este proceso es un poco lento ya que primero es necesario lanzar el motor de Matlab, después cargar la toolbox y por último realizar la optimización, este tiempo se reduciría mucho si no se usara Matlab.

En esta etapa ya se conocen todos los parámetros del robot: tipo de cada uno de los módulos y parámetros que deben tener los eslabones, la parte de Matlab devuelve un vector con los radios interno y externo, longitudes y radios de los redondeos internos y externos.

Se ha creado un código que a partir de estos parámetros generar los archivos stl de los eslabones, se usa un módulo de Python llamado **SolidPython**. Este módulo permite generar piezas en formato scad como la que aparece en la Fig 7 donde se ha creado a partir de los parámetros el trozo del eslabón de los redondeos.

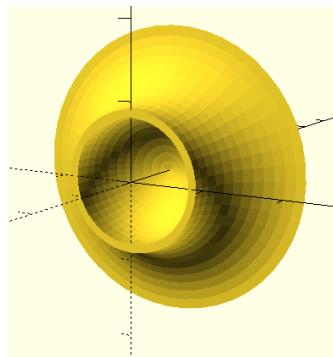


Figura 7: Visualización de los redondeos del eslabón generado.

Una vez que se ha generado el eslabón completo en scad (Fig 8), se usa un comando que proporciona el software OpenSCAD que permite exportar el archivo scad en formato stl.

Listing 1: Comando para exportar un archivo scad como stl.

```
1      openscad -o output_file.stl input_file.scad
```

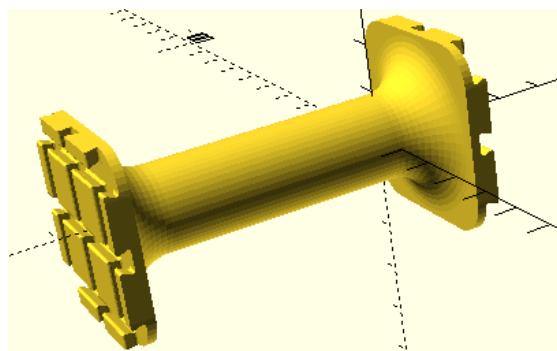


Figura 8: Eslabón completo scad generado.

La aplicación también genera un archivo en formato json donde se almacena toda la información del manipulador, tipo de motores y parámetros de cada eslabón. Este archivo puede ser muy útil para generar los eslabones de forma manual ya que editando el fichero JSON se puede cargar y generar las piezas.

**Ejemplo de uso** Llega el momento de ver un ejemplo. Se mostrará utilizando imágenes, con el fin de que quede mucho más claro.

#### PASO 1 - Eligiendo una acción

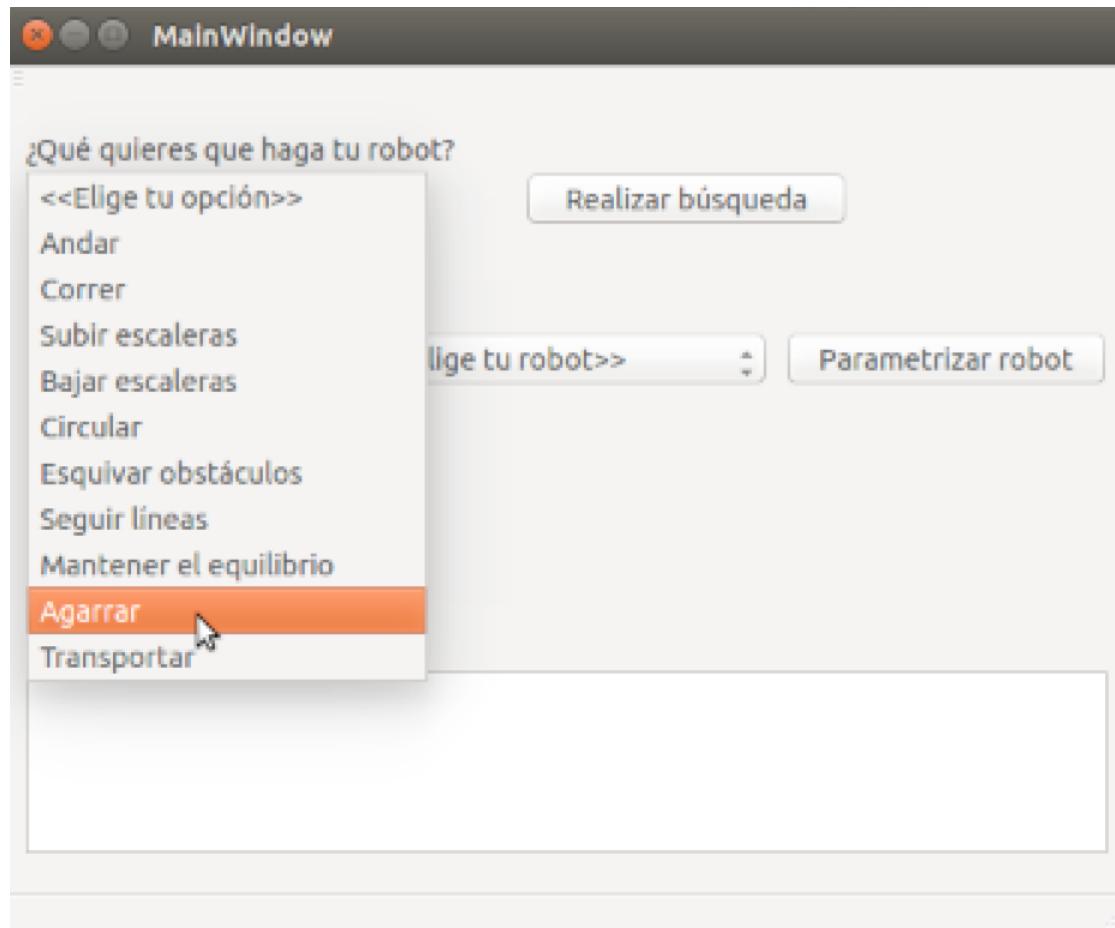


Figura 9: COREapp: Ejemplo Manipulador - Eligiendo la acción 'Agarrar'.

## PASO 2 - Eligiendo el robot

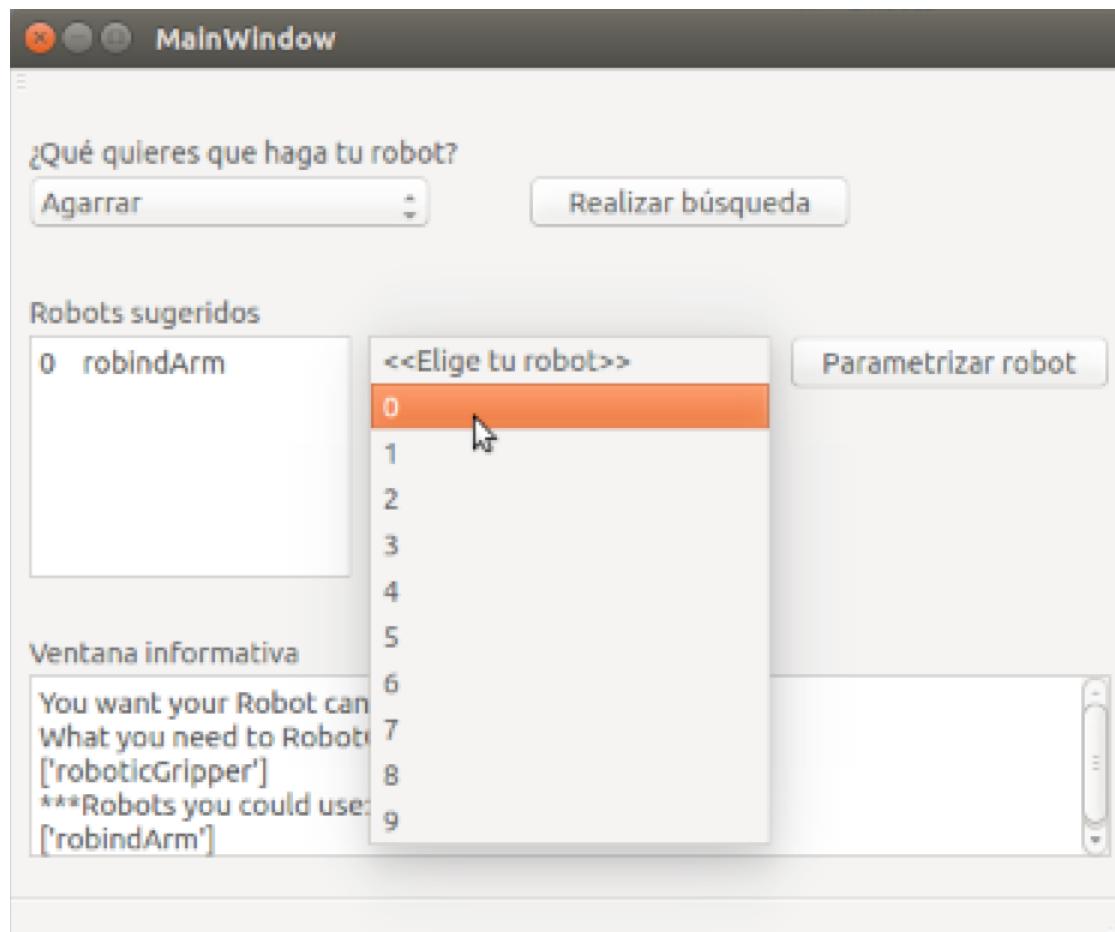


Figura 10: COREapp: Ejemplo Manipulador - eligiendo el robot 'robindArm'.

### PASO 3 - Comienza el proceso de parametrización

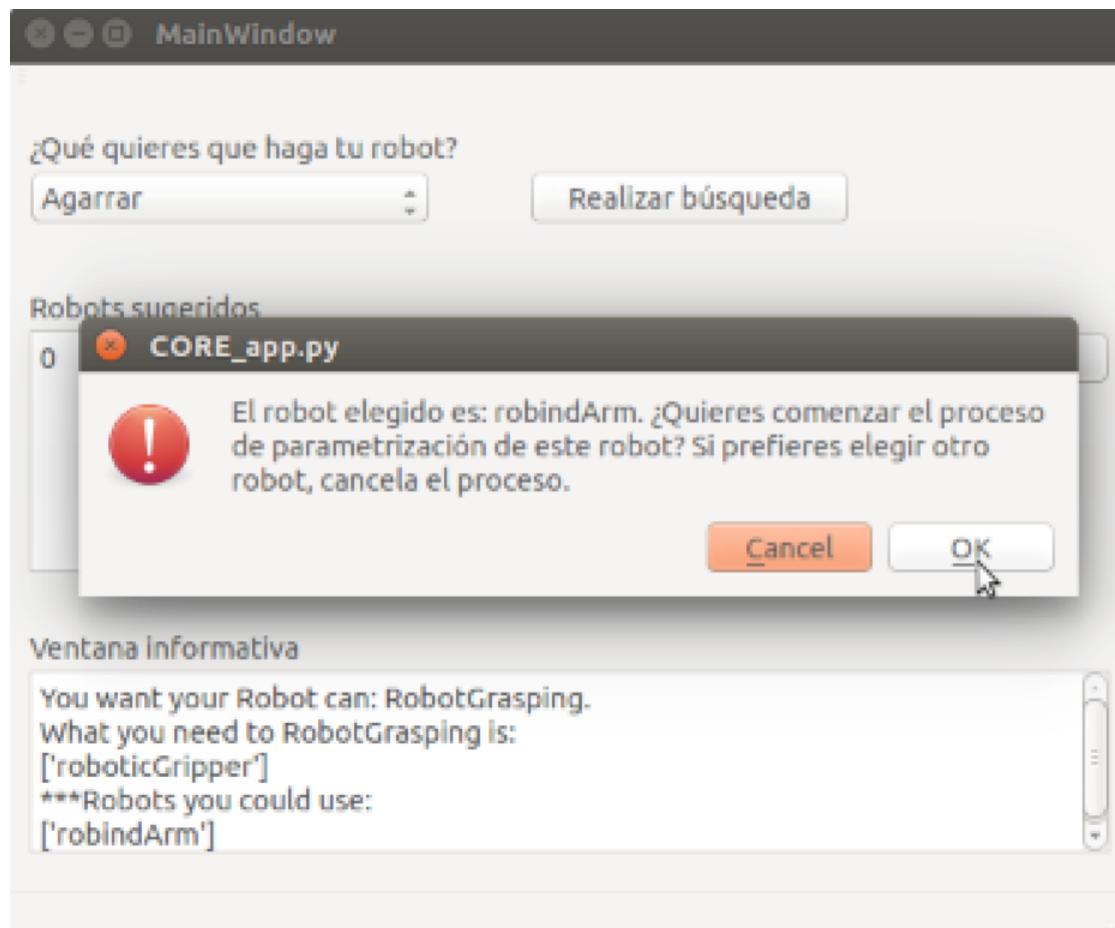


Figura 11: COREapp: Ejemplo Manipulador - Comienzo del proceso de parametrización, se abre la nueva ventana.

#### PASO 4 - Parametrizando el robot I

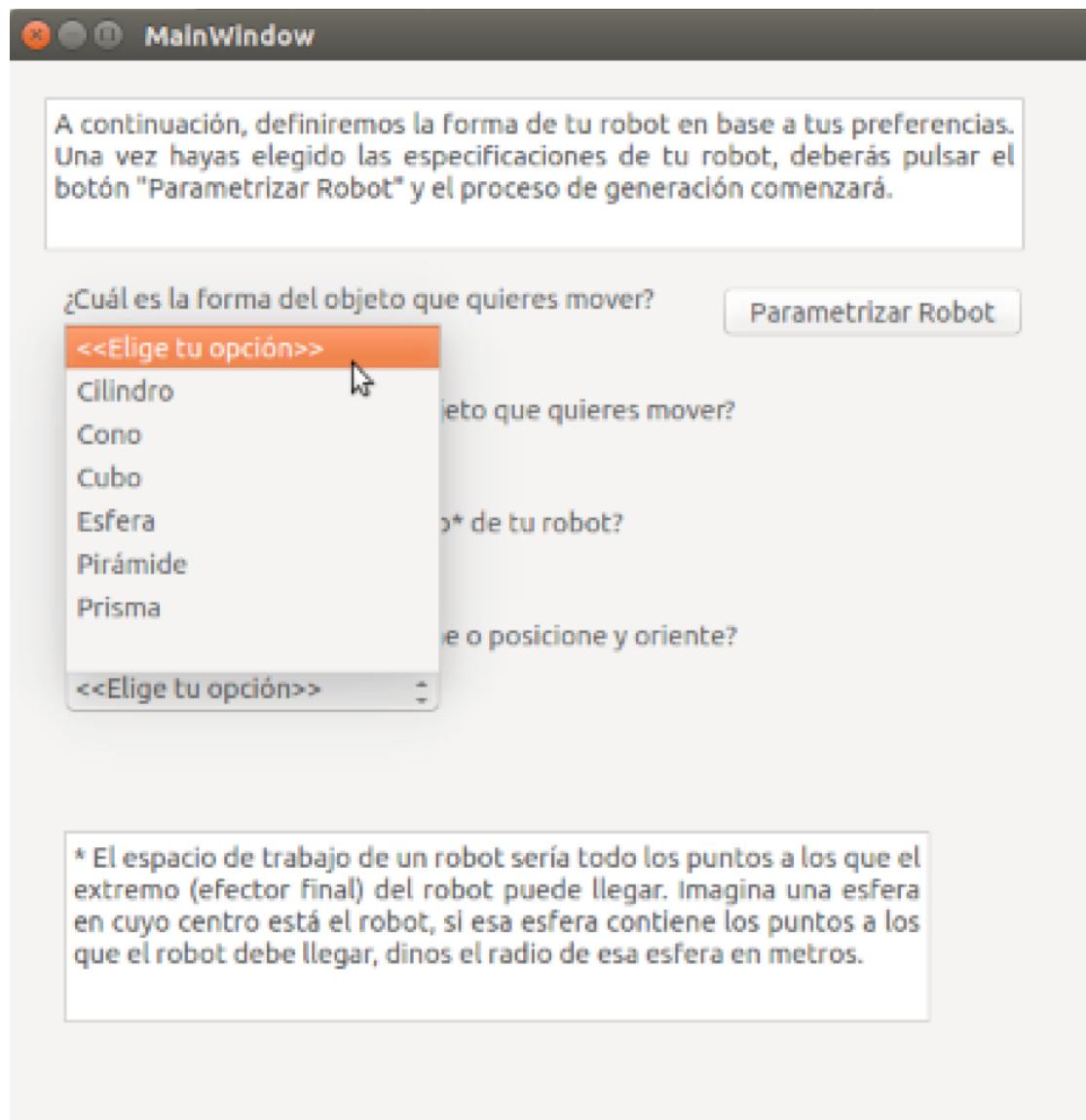


Figura 12: COREapp: Parametrizando el robot I: forma del objeto a manipular.

### PASO 5 - Parametrizando el robot II



Figura 13: COREapp: Parametrizando el robot II: elección del número de grados de libertad (tres o seis).

### PASO 6 - Parametrizando el robot III

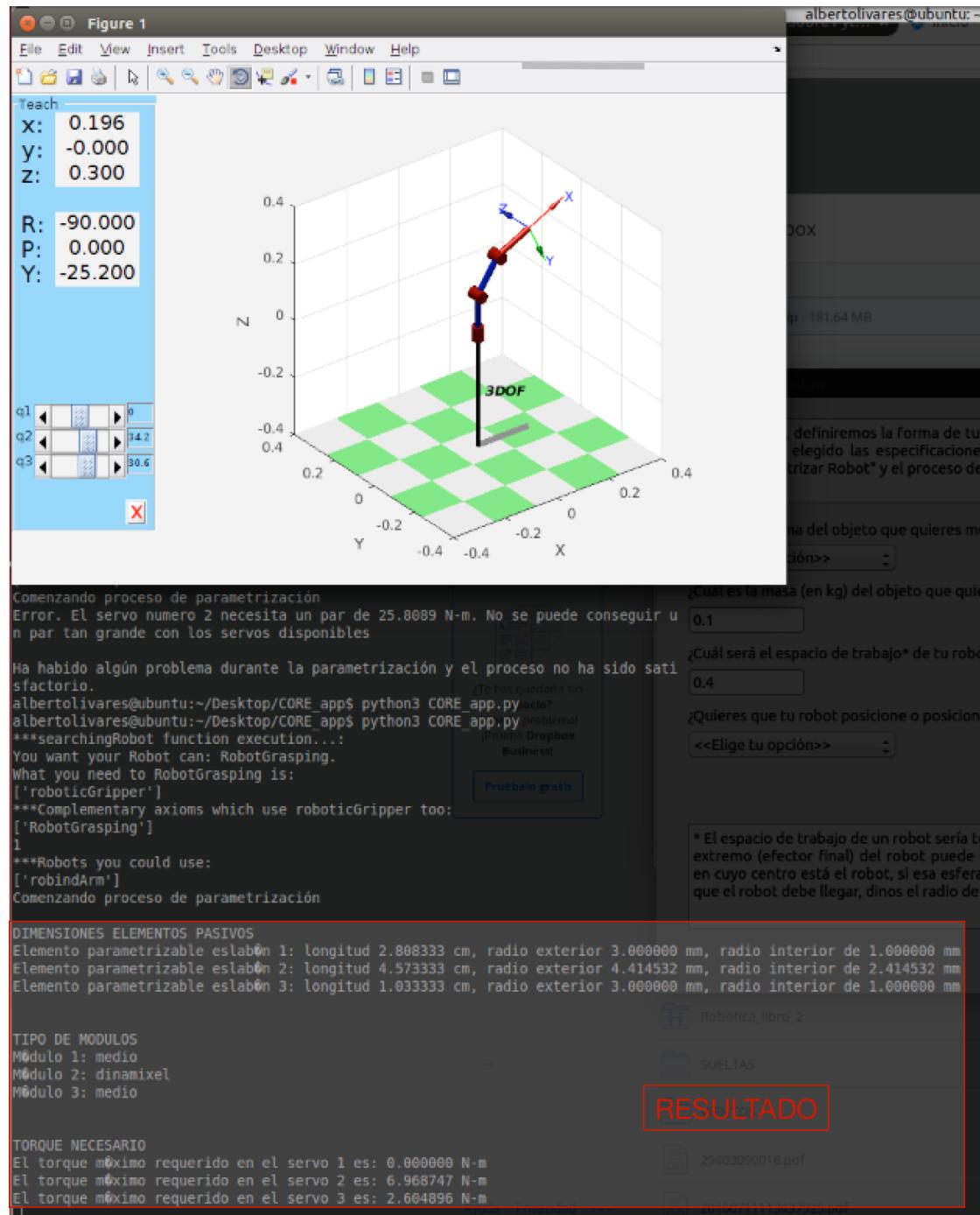


Figura 14: COREapp: Parametrizando el robot III: Robot totalmente diseñado para un espacio de trabajo de 0.4 m y un objeto de 0.1 kg.

## 2.3. Robot con ruedas

El procedimiento para parametrizar este robot es igual que el anterior. Cuando se desea parametrizar un robot de dos o cuatro ruedas se abre la ventana que se ve en la Fig 15.

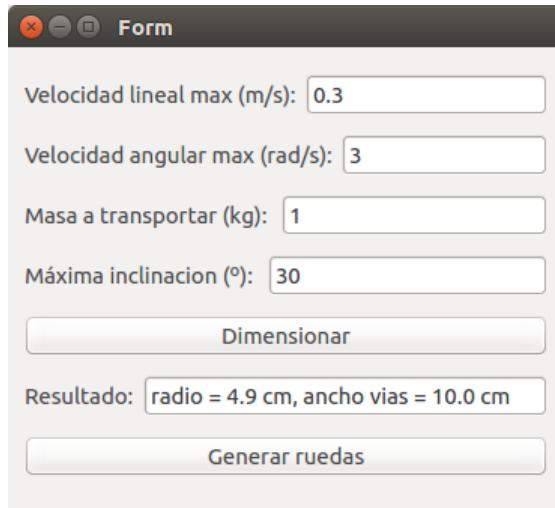


Figura 15: Ventana de parametrización de un robot móvil.

Según los valores introducidos se generan los parámetros para las ruedas y si es necesario introducir un separador entre los módulos para aumentar la distancia entre las ruedas. También según los parámetros se selecciona entre los dos tipos de motores y si el robot necesita tener dos o cuatro ruedas motrices.

Para la generación de los stl se realizan los mismo pasos que para el manipulador, en la Fig 16 se puede ver el archivo CAD de una rueda parametrizada.

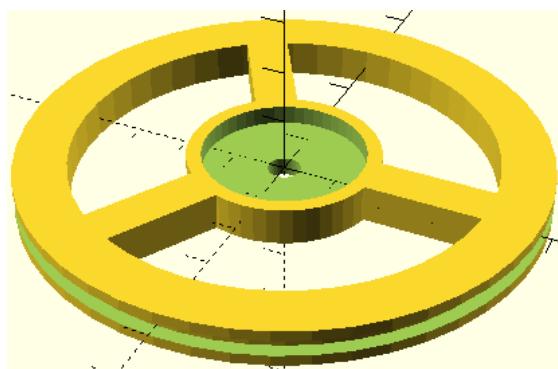


Figura 16: Visualización de una rueda parametrizada.

### 3. Programas para el estudio dinámico del robot

#### 3.1. Introducción y uso de MATLAB

El objetivo final de este estudio es la generación de robots manipuladores y cualquier otro tipo de robots, entendiendo por generación el cálculo de una serie de parámetros clave para su construcción y definición, como pueden ser la dimensión de los eslabones, tipo de motores a utilizar, etc. Esto se deberá realizar de manera automática, en función de unos parámetros de entrada que variarán según el tipo de robot.

Para llevar a cabo esto, es fundamental el uso de una serie de herramientas matemáticas que nos permitan el estudio geométrico, cinemático y dinámico de un robot.

Esta etapa de generación, que constituye un paso previo a la simulación y posterior construcción del robot, es de un carácter eminentemente más matemático. Por esto, se ha decidido utilizar el software MATLAB, desmarcando esta etapa de la fase de simulación, donde se puede emplear otro tipo de software como por ejemplo kdl o gazebo.

Dentro de MATLAB podemos encontrarnos con diversas opciones a la hora de trabajar con robots, como pueden ser las librerías ARTE [4], RoKiSim [1], HEMERO, etc... Nosotros nos vamos a decantar por la Robotic Toolbox de Peter Corke [2]. Esta librería se trata de la más extendida dentro del campo de la robótica, razón por la cual se puede encontrar una mayor cantidad de documentación y de mejor calidad. Contiene un gran número de funciones que permiten realizar un estudio muy versátil de los robots. En concreto, vamos a utilizar la versión 9.10, la más actualizada a día de hoy.

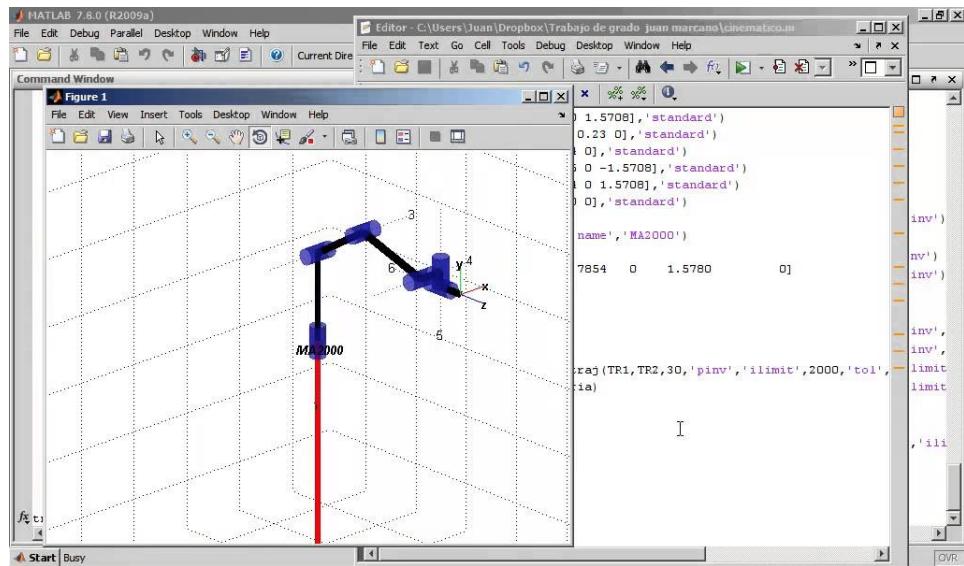


Figura 17: Interfaz de Robotics Toolbox

Hay que mencionar que esta librería también se puede encontrar escrita en Python, de manera que se puede utilizar de forma libre sin necesidad de recurrir a un software de pago como MATLAB. Sin embargo, se trata de una versión muy desactualizada y sin ningún tipo de soporte, y que tras pruebas demostró ser muy poco funcional, siendo prácticamente imposible trabajar con ella. Esto justifica el hecho de tener que usar MATLAB.

A pesar de esto, cabe destacar que las funciones realizadas en MATLAB se pueden llamar fácilmente a través de otras hechas en Python, lo cual resulta muy conveniente a la hora de efectuar las aplicaciones finales para el proyecto.

### 3.2. Uso de Robotics Toolbox

En esta sección se van a desarrollar algunos de los aspectos básicos a la hora de trabajar con la toolbox.

#### 3.2.1. Funciones básicas

Estas son algunas funciones que han resultado más útiles tanto para definir robots como para trabajar posteriormente con ellos. En la documentación de la

toolbox [3] se puede encontrar información mucho más detallada sobre estas y otras funciones.

- *startup\_rvc*. Es necesario ejecutar este comando cada vez que se inicia MATLAB para poder utilizar la toolbox
- *Revolute*. Definición de un eslabón del robot con articulación de rotación. Puede contener todos los parámetros físicos y geométricos del eslabón.

$$L(1) = \text{Revolute}('d', 0.29, 'a', 0, 'alpha', -pi/2, 'offset', 0, \dots)$$

- *SerialLink*. Crea un objeto robot en el que se incluyen todos sus parámetros, introduciendo una matriz de Denavit-Hartenberg o una concatenación de eslabones definidos con *Revolute*

$$R = \text{SerialLink}(DH, 'name', 'IRB 120')$$

- *plot*. Representación gráfica del robot en una posición articular  $Q$ .

$$R.plot(Q)$$

- *teach*. Representación gráfica del robot en la que además se incluye una botonera que permite modificar las posiciones articulares del mismo.

$$R.teach()$$

- *fkine*. Permite obtener la cinemática directa en una posición articular  $Q$ .

$$T=R.fkine(Q)$$

- *ikine*. Devuelve la cinemática inversa del robot, introduciendo una matriz  $T$  con la posición del efecto final del robot (matriz resultante de la cinemática directa).

$$R.ikine(T)$$

En el caso de un robot manipulador de menos de 6 grados de libertad el sistema es ‘infraactuado’, por lo que la cinemática inversa es más complicada de calcular. En estos casos es necesario incluir un vector  $Q$  con posiciones articulares que podrían ser solución, y un vector  $m$ , en el que se indiquen las dimensiones del espacio cartesiano que pueden ser ignoradas a la hora de encontrar la solución. Por ejemplo, en un robot de 3DOF, solo nos interesan los valores de posición  $x, y, z$ , y no los tres valores de orientación. En este caso:

$$m = [1 \ 1 \ 1 \ 0 \ 0 \ 0]$$

$$R.\text{ikine}(T, Q, m)$$

- *jacob0*. Matriz jacobiana (con coordenadas referenciadas a la base) en una posición articular  $Q$ .

$$J = R.\text{jacob0}(Q)$$

- *rne*. Cálculo de la dinámica inversa para una posición  $Q$ , velocidad  $Qd$  y aceleración  $Qdd$  mediante el método de Newton-Euler.

$$R.\text{rne} = (Q, Qd, Qdd)$$

Las siguientes funciones trabajan con el método de Newton-Euler. Son simplificaciones que se pueden utilizar para algunos casos concretos.

- *gravjac*. Devuelve el torque necesario para poder mantener el robot en estático en una posición articular  $Q$ .

$$R.\text{gravjac}(Q)$$

- *payload*. Añade una carga de masa  $m$  en la posición  $p$  del eje de coordenadas del efecto final.

$$R.\text{payload}(m, p)$$

- *pay*. Determina el torque extra generado en cada articulación cuando se ejerce un esfuerzo en el efecto final del robot. El esfuerzo se introduce como un vector columna  $w$ . También es necesario incluir el jacobiano  $J$

$$w = [0 \ 0 \ 1 \ 0 \ 0 \ 0]';$$

$$R.\text{pay}(w, J);$$

### 3.2.2. Métodos para la definición de robots

La toolbox permite elegir el grado de información con la que se define un robot. De esta manera, se proponen dos métodos para definirlos, uno sencillo y otro más complejo. Dependiendo de los análisis a los que se quiera someter al robot, el primer método puede ser suficiente.

**Definición mediante matriz de Denavit-Hartenberg** La matriz de Denavit-Hartenberg *DH* proporciona la información geométrica necesaria para definir un robot (longitud de los eslabones y orientación relativa entre los mismos). Este método de definición se puede calcular la cinemática directa, inversa, jacobiano, etc...

**Definición mediante matriz de Denavit-Hartenberg e información física de los eslabones** A parte de la información relativa a la matriz *DH*, también se puede incluir la información física de cada eslabón, de tal manera que se puedan realizar análisis dinámicos. Se permite introducir los siguientes parámetros de cada eslabón y articulación:

- *I*. Momento de inercia del eslabón con respecto al sistema de referencia del propio eslabón.
- *r*. Posición del centro de gravedad del eslabón con respecto al sistema de referencia del propio eslabón.
- *m*. Masa del eslabón.
- *Jm*. Inercia del motor.
- *G*. Relación de transmisión del motor.
- *B*. Rozamiento de la articulación.
- *Tc*. Rozamiento de Coulomb.
- *qlim*. Valores límites de la articulación.

Para poder trabajar con funciones que utilicen la información física del robot, los parámetros mínimos e indispensables a definir son *I*, *r* y *m*. El resto son opcionales.

### 3.2.3. Consideraciones para la definición de robots

Los análisis dinámicos son indispensables para el objeto de nuestro estudio, por lo tanto, va a ser necesario una definición completa de los robots sobre los que queramos trabajar.

El tener que proporcionar la información física de los robots supone una restricción importante, tanto para trabajar con robots ya existentes, como a la hora de definir nuestros propios robots.

El principal problema que podemos encontrar a la hora de buscar modelos de robots comerciales es que mientras que la matriz de Denavit-Hartenberg *DH* suele estar disponible en las especificaciones técnicas, la información física de los eslabones no se proporciona por cuestiones de confidencialidad. De esta manera, el espectro de robots comerciales disponibles se encuentra reducido a modelos creados a partir de información recabada por usuarios particulares. Por suerte, la propia toolbox de Peter Corke incluye robots predefinidos de manera completa, entre los que se encuentran algunos bastante ‘populares’ entre la comunidad de la robótica como son el *Puma 560*, *ABB IRB 140*, etc... lo cual nos permite realizar pruebas para llegar a comprender perfectamente el funcionamiento de la toolbox.

Si queremos definir nuestro propio robot, el primer paso es obviamente calcular la matriz *DH*. Una vez hecho debemos obtener su información física. La opción más sencilla para lograr esto es extraer estos datos desde el propio software de diseño, que en nuestro caso es SolidWorks. Con SolidWorks, podemos ensamblar cada uno de los eslabones que conforman el robot por separado, y obtener la posición del centro de masas, la matriz de inercias medidas desde el centro de masas y la masa. Estos datos se pueden encontrar en la herramienta de propiedades físicas (figura 3.2.3).

Para que estos datos sean correctos, es importante definir correctamente el material correspondiente. Las piezas impresas están hechas de PLA. La densidad de este material puede variar en función del fabricante, el color del mismo, etc. Por esto se decidió medir experimentalmente la densidad de distintas piezas impresas en PLA, obteniendo el valor de:

$$\rho_{PLA} = 1250 \text{kg/m}^3$$

Una vez calculados los datos de inercias, centros de masa y masa, se incluyen

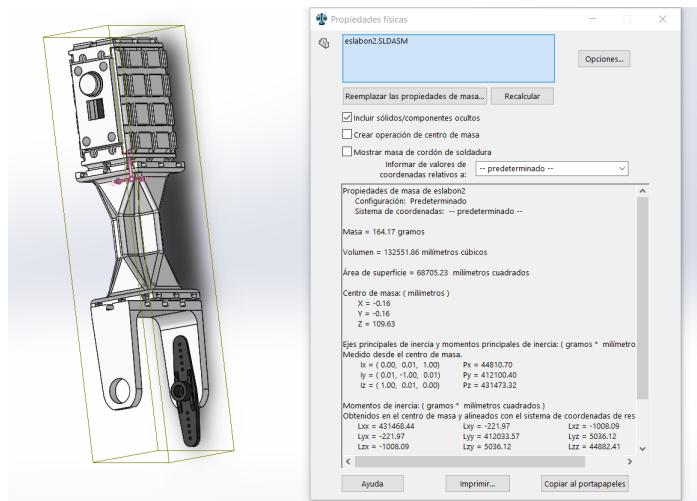


Figura 18: Herramienta propiedades físicas en SolidWorks

en la definición cada eslabón. Para definirlos, es necesario prestar atención a los sistemas de referencias. El sistema de coordenadas de cada eslabón está definido por la matriz *DH* del robot. Es necesario que el sistema de referencia del eslabón y el usado en SolidWorks a la hora de posicionar la pieza coincidan para una correcta definición del eslabón. En caso contrario, será necesario rotar la matriz de inercia y reajustar el vector que define la posición del centro de masa para que ambos sistemas de ejes coincidan.

Otro de los parámetros que se pueden definir es el límite de giro de las articulaciones. Estos están determinados por los servomotores utilizados, siendo el límite de  $[-90^\circ, 90^\circ]$  en caso de los servos de posición.

Se muestra la definición completa de un eslabón (figura 19) y la representación gráfica de un robot en MATLAB (figura 20).

```
L(1) = Revolute('d', 0.1725, 'a', 0, 'alpha', -pi/2, ...
'offset', 0, ...
'I', [304985.13, 218.29, 778.58; 218.29, 298502.65, -771.20; ...
778.58, -771.20, 33075.08]*10^(-9), ...
'r', [0, 0, 0.0565], ...
'm', 0.15386, ...
'qlim', [-90 90]*rad);
```

Figura 19: Definición de un eslabón en MATLAB

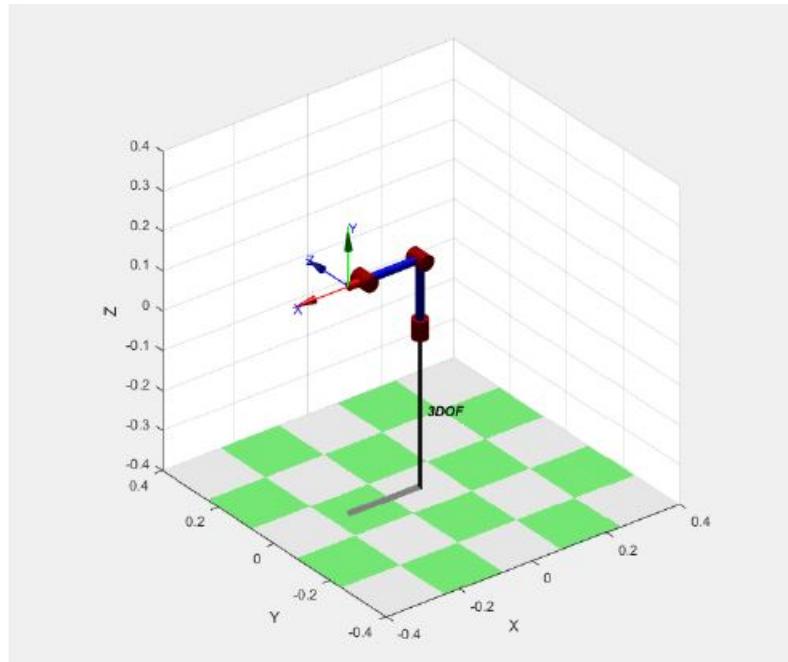


Figura 20: Representación gráfica de un robot en MATLAB

### 3.3. Robot manipulador

Los robots manipuladores, también denominados brazo robótico o robot antropomórfico por similitud con un brazo humano (figura 21). Su gran versatilidad los convierte en los robots más usados hoy en día en la industria.

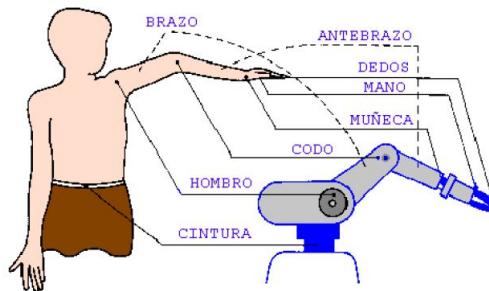


Figura 21: Similitud brazo humano-robot manipulador

Existen multitud de configuraciones de robots manipuladores, siendo las más

frecuentes las siguientes:

- 3 grados de libertad. Permite posicionamiento.
- 4 grados de libertad. Permite posicionamiento. El último grado de libertad es redundante, de manera que pueda facilitar la labor de posicionamiento.
- 6 grados de libertad. Permite posicionamiento y orientación.

Como representante de los robots manipuladores se ha decidido construir y parametrizar un robot de 3 grados de libertad. Este manipulador se ofrece en su configuración más típica y generalizada, cuya orientación relativa entre eslabones está definida por su matriz de *DH* (figura 22)

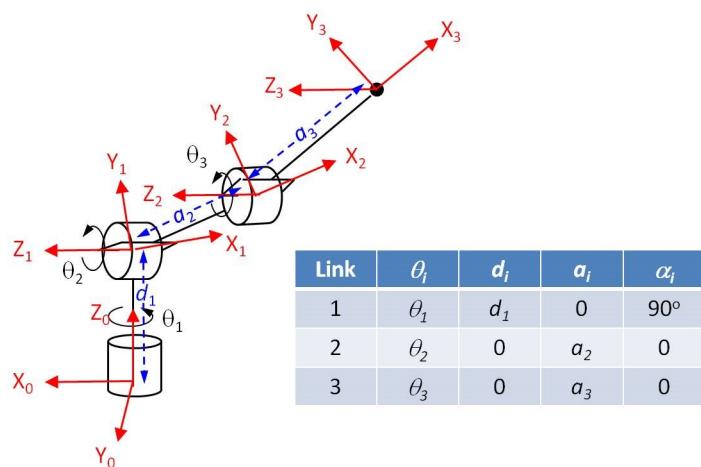


Figura 22: Configuración típica de un manipulador de 3DOF

Debido a la complejidad y a las limitaciones a las que estamos sometidos (principalmente la potencia de los servomotores de los que disponemos) no se han realizado manipuladores de más grados de libertad.

El ensamblaje del manipulador está conformado por los siguientes elementos:

- 3 módulos servo
- 3 eslabones de longitud variable y optimizables
- 2 elementos pasivos radio

- 1 elemento pasivo base
- 1 efecto final

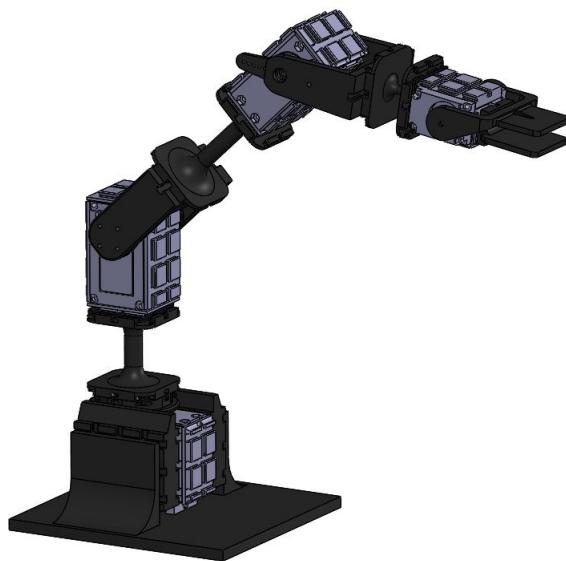


Figura 23: Ensamblaje manipulador de 3DOF

El objetivo del estudio es la generación automática de uno de estos manipuladores de 3 grados de libertad en función de dos parámetros de entrada, la masa que puede soportar el robot en su extremo y la longitud total del mismo.

Este proceso de generación automática consta fundamentalmente de una serie de pasos que se desarrollarán a continuación.

### 3.4. Cálculo automático de las dimensiones de los eslabones

La longitud total del robot (que en caso del manipulador corresponde con el espacio de trabajo del mismo) y está relacionado con los puntos donde puede posicionarse el efecto final del robot. Obviamente, la longitud total va a definir la longitud de los eslabones.

### 3.4.1. Primera estrategia

A la hora de trabajar con el posicionamiento de un robot, las funciones más interesantes son la cinemática directa y la inversa. La cinemática directa nos dice la posición del efecto final del robot para una determinada posición articular. La cinemática inversa permite calcular el valor articular de cada eslabón en función de la posición final del robot.

Utilizando esta herramientas, se plantea una estrategia para el cálculo de los eslabones. En función del campo de trabajo que queramos que tenga el robot se puede definir una coordenada  $(x, y, z)$  para el efecto final del robot. Es decir, podemos utilizar una matriz T (que contiene la posición del efecto final), y resolver la cinemática inversa para obtener como salida un conjunto de valores articulares Q, que hagan que el robot se encuentre en esta posición y que dependerán de las dimensiones del robot. Si tenemos las dimensiones del robot, podemos sacar fácilmente la longitud de los eslabones.

Este método implica declarar como variable simbólicas la longitud de los eslabones. Resolviendo el problema que se plantea se podría obtener un conjunto de valores de longitudes de eslabón que hagan que el robot tenga la longitud total deseada. El inconveniente es que la presencia de múltiples soluciones y el cálculo simbólico hacen el problema más complejo de resolver de lo deseado, por lo que vamos a descartar este planteamiento.

### 3.4.2. Segunda estrategia

Se opta por un método mucho más sencillo para determinar la longitud de los eslabones, pero igualmente válido. Se trata simplemente de introducir la longitud total del robot y hacer que cada uno de los eslabones mida lo mismo. Es decir, la longitud de cada eslabón será:

$$\text{Longitud eslabón} = \frac{\text{Longitud total robot}}{3} \quad (1)$$

Como ya hemos visto, cada uno de los eslabones está compuesto por una serie de elementos tanto de longitud fija como de longitud variable. Si conocemos la longitud total del eslabón y las longitudes de los elementos fijos, es muy sencillo calcular la longitud de los elementos parametrizables.

Para el manipulador, la distribución de los elementos fijos siempre es la misma (figura 24), de forma que tras tomar las medidas correspondiente siempre podemos calcular de la misma manera la longitud de los elementos parametrizables:

■ Primer eslabón

- $d_{11}$  = Longitud elemento pasivo base (fija)
  - $d_{12}$  = Parametrizable
  - $d_{13}$  = Dependiente el tipo de módulo a utilizar (fija)
- Longitud total eslabón =  $d_{11} + d_{12} + d_{13}$

■ Segundo eslabón

- $d_{21}$  = Longitud elemento pasivo radio (fija)
  - $d_{22}$  = Parametrizable
  - $d_{23}$  = Dependiente el tipo de módulo a utilizar (fija)
- Longitud total eslabón =  $d_{21} + d_{22} + d_{23}$

■ Tercer eslabón

- $d_{31}$  = Longitud elemento pasivo radio (fija)
  - $d_{32}$  = Parametrizable
  - $d_{33}$  = Longitud del efecto final (fija)
- Longitud total eslabón =  $d_{31} + d_{32} + d_{33}$

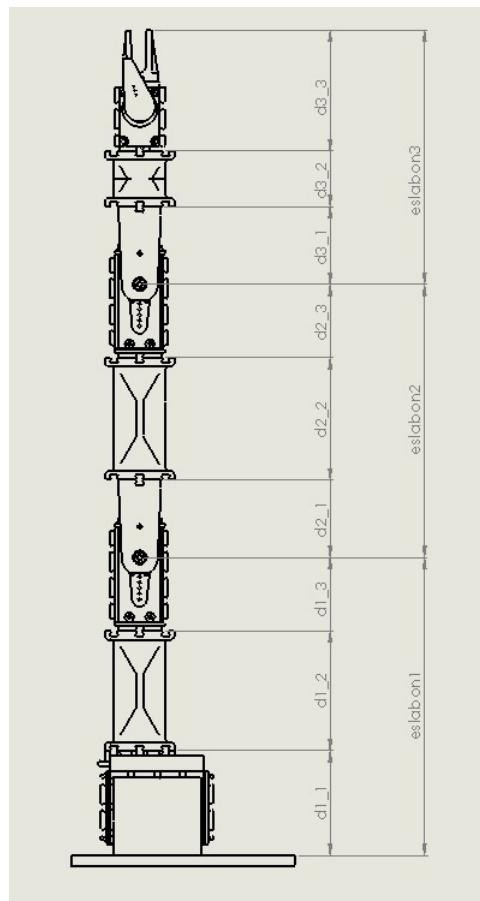


Figura 24: Longitudes de los elementos que forman el manipulador de 3DOF

Las longitudes de los elementos fijos están incluídas en el programa encargado de calcular las partes parametrizables.

Una vez realizados estos cálculos, hay que tener también en cuenta las siguientes consideraciones:

1. Por cuestiones de diseño, la longitud de el elemento parametrizable no puede ser inferior a 2 cm.
2. Si la longitud del elemento parametrizable debe ser menor que la longitud mínima, se prescindirá del elemento parametrizable conectando los elementos fijos directamente entre sí. (es decir, longitud del elemento parametrizable = 0)

3. Si se ha tenido que prescindir de un elemento parametrizable, el eslabón será más pequeño de lo que debería ser. Para compensar esto, sumamos la longitud que hemos perdido a la del elemento parametrizable de otro eslabón, asegurando así que la longitud total del robot es la que inicialmente deseábamos
4. En el caso de que la longitud del elemento parametrizable sea negativa (por ejemplo, un eslabón que debe medir 10 cm y solamente sus partes fijas miden 12 cm, el elemento parametrizable debería ser de  $-2$  cm) también prescindimos del elemento parametrizable, siendo el eslabón más grande de lo que debería ser. Para compensar esto, restamos la longitud que hemos aumentado a la del elemento parametrizable de otro eslabón, asegurando así que la longitud total del robot es la que inicialmente deseábamos

Esta serie de restricciones da lugar a una longitud mínima para el robot, que es de unos 30 cm. Esta configuración correspondería a un montaje donde se ha prescindido de todos los elementos parametrizables.

La longitud máxima del robot no está limitada por las especificaciones del cálculo la longitud de elementos parametrizables (suponiendo que pudiésemos imprimir elementos parametrizables de cualquier tamaño, en caso contrario estaríamos también limitados por el área de trabajo de nuestra impresora), sino que está limitado por el esfuerzo de los servomotores que se coloquen en cada articulación. Obviamente, a mayor tamaño, se necesita entregar un mayor par en cada articulación. En nuestro caso, los servos de los que disponemos nos limitan a un tamaño de robot aproximadamente de 55 cm.

### 3.5. Optimización de los eslabones

Esta sección está desarrollada en el entregable E4.3, Capítulos 5 y 8.

### 3.6. Caracterización automática de las propiedades físicas de los eslabones

Como ya se ha comentado, la correcta definición de las propiedades físicas de los eslabones (masa, centro de gravedad e inercias) es fundamental para hacer análisis dinámico en el robot.

La generación automática de los eslabones conlleva un inconveniente, y es que obviamente, sus características físicas van a cambiar según su longitud. Recordemos que para la obtención de las características de un eslabón concreto se ha utilizado la herramienta correspondiente en SolidWorks. Lógicamente, esto no se puede utilizar cada vez que generemos un robot, por lo que hay que buscar un método que automatice el cálculo de las propiedades físicas.

En un primer momento se planteó analizar unos eslabones de una longitud concreta, obtener sus dimensiones, sobredimensionarlas y establecerlas como unas características fijas para todos los eslabones generados, de modo que garanticemos la rigidez/resistencia de las estructuras robóticas. Esto puede ser una estrategia viable, ya que se puede observar que los resultados son bastante acertados. Sin embargo, al alejarnos de las dimensiones de los eslabones que hemos tomado para generar los valores de referencia, el cálculo de pares puede desvirtuarse, por no hablar del incremento, en la mayor parte de casos innecesario, en la cantidad de material utilizado en la impresión. Por ello se ha buscado otro método que permita calcular los parámetros geométricos con exactitud.

Recordemos que cada uno de los eslabones del robot siempre va a constar de tres elementos, dos fijos (módulo y radio o base) y uno parametrizable. Se pueden obtener las propiedades físicas de cada uno de estos elementos por separado, y tras esto calcular las del conjunto.

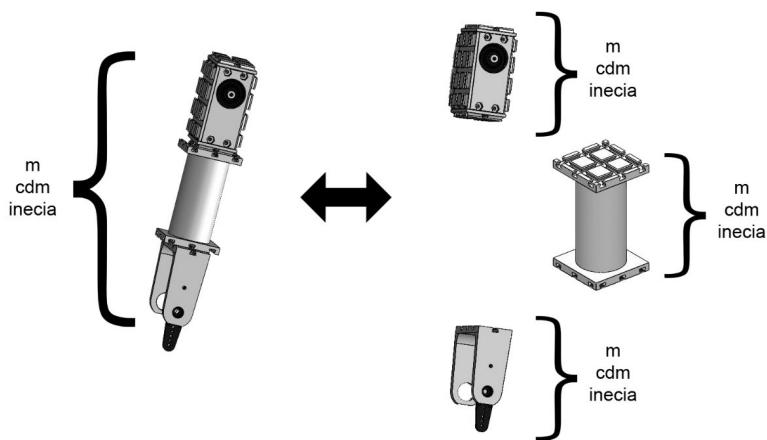


Figura 25: Desglose de las propiedades físicas de un eslabón

Las características de los elementos fijos de pueden obtener en SolidWorks . Se han extraído las propiedades de todos los elementos fijos disponibles (estos datos se encuentran en el fichero 4 del Anexo B) tales como los de la Figura 25). Recordemos que en este paso hay que prestar atención tanto al material como al sistema de coordenadas.

Los elementos parametrizables tienen una geometría sencilla (cilindro hueco), lo cual permite calcular las propiedades sin necesidad de recurrir a SolidWorks. Es necesario obtener previamente la longitud total del elemento parametrizable ( $l$ ), su radio externo ( $r_{externo}$ ) y su radio interno ( $r_{interno}$ ). Para simplificar, los redondeos internos y externo del elemento parametrizable no se tendrán en cuenta en estos cálculos.

- Posición del centro de masas ( $\chi_{cdm}$ ): estará siempre en el centro de la pieza.

$$\chi_{cdm} = \frac{l}{2} \quad (2)$$

- Masa( $m$ ): Se calculará primero el volumen de la pieza ( $v$ ), para luego sacar la masa en función de la densidad( $\rho$ )

$$v = l\pi(r_{externo}^2 - r_{interno}^2) \quad (3)$$

$$m = \rho v \quad (4)$$

- Matriz de inercias( $I$ ). Se calcula tanto la componente de inercia en el eje que atraviesa la pieza longitudinalmente ( $I_z$ ) como la componente en los otros dos ejes ( $I_{x,y}$ )

$$I_z = \frac{1}{2}m(r_{externo}^2 + r_{interno}^2) \quad (5)$$

$$I_{x,y} = \frac{1}{4}m(r_{externo}^2 + r_{interno}^2) + \frac{1}{12}ml^3 \quad (6)$$

Una vez que tenemos las propiedades de todos los elementos, se pueden obtener las del eslabón completo:

- Masa( $m_T$ ): La masa del conjunto es la suma de las masas de los elementos por separado

$$m_T = \sum m_i \quad (7)$$

- Posición del centro de masas ( $\chi_{cdmT}$ ): Para un sistema formado por un conjunto de masas, el centro de masas se puede calcular como:

$$\chi_{cdmT} = \frac{\sum \chi_{cdmi} m_i}{\sum m_i} \quad (8)$$

- Matriz de inercias. ( $I_T$ ). La componente de inercia del conjunto total en el eje longitudinal será la suma de las inercias de cada elemento en el mismo eje

$$I_{zT} = \sum I_{zi} \quad (9)$$

La componente de inercia en los otros dos ejes se calcularán mediante el teorema de Steiner.

$$I_T = I_i + \sum m_i r_i^2 \quad (10)$$

Donde  $r$  es la distancia entre el centro de masas del elemento aislado y el centro de masas del elemento en el conjunto

$$r = \chi_{cdm \text{ elemento aislado}} - \chi_{cdm \text{ elemento en el conjunto}} \quad (11)$$

### 3.7. Cálculo del par en las articulaciones

Encontrar el tipo de servo que se debe colocar en cada articulación es un paso fundamental para el correcto funcionamiento del robot. Para esto, hay que calcular el par o torque máximo que se puede ejercer en cada articulación, y en base a los resultados, elegir un servo que sea capaz de proporcionar el par requerido.

Para este tipo de cálculos dinámicos, se puede utilizar los jacobianos. La matriz jacobiana relaciona las velocidades lineales y angulares del efecto final del robot con las velocidades angulares de cada articulación. A partir de esto, y mediante el principio de los trabajos virtuales, se puede conocer la relación entre los pares ejercidos en cada articulación y las fuerzas efectuadas en el efecto final.

Sin embargo, mediante el cálculo con jacobiano, no se tienen en cuenta las propiedades físicas del propio robot. Es decir, no se tiene en cuenta los efectos que producen la masa o las inercias de los eslabones en la dinámica del sistema.

Esto hace que este tipo de cálculos no resulten los más apropiados cuando se está trabajando con modelos reales.

Para solucionar esto, se va a emplear la formulación de Newton-Euler. Mediante el método de Newton-Euler se pueden relacionar el movimiento del robot con todas las fuerzas implicadas en el mismo, o lo que es equivalente, permite encontrar relaciones matemáticas entre la posición, velocidad y aceleración en cada articulación y las fuerzas y pares aplicadas en las mismas, teniendo en cuenta los parámetros físicos del robot.

Dejando de lado la metodología para el cálculo de la dinámica del sistema, tenemos que podemos encontrar en dos tipos de situaciones a la hora de buscar el torque en las articulaciones:

- Análisis estático. El robot se encuentra en una posición fija. Hay que calcular el par necesario para mantener al robot en esa posición sin que se mueva.
- Análisis dinámico. El robot está en movimiento. Hay que calcular el par necesario para asegurar que se pueda producir movimiento.

Resulta interesante analizar ambos tipos de situaciones e incluirlas en nuestros cálculos.

En función de los resultados que se obtengan en estos análisis, se debe elegir el tipo de módulo que se colocará en la articulación. Recordemos que se dispone de tres tipos de módulo, cada uno con un servo distinto. Las características de los mismos son:

Tipo de servo	Peak Torque (kg-cm)	Velocidad máxima (rpm)
Pequeño (ES08A)	1.5	83.3
Medio (HK)	4	66.6
Dynamixel (AX-12a)	12.29	59

Tabla 1: Datos de los servomotores disponibles

### 3.7.1. Cálculo del par estático

El par estático es el par necesario para mantener al robot en una posición fija. Esto se traduce en que al aplicar el método de Newton-Euler, tanto la velocidad ( $Qd$ ) como la aceleración ( $Qdd$ ) será cero.

El par ejercido en cada articulación dependerá de la posición articular del robot. Por esto, hay que evaluar al robot en la posición en la que el par sea máximo. Para ello, la opción más sencilla y deseable es encontrar la posición más desfavorable para el robot.

En el caso del robot manipulador, el máximo esfuerzo siempre se dará en la posición en la cual el robot se encuentra completamente extendido de manera horizontal. Esto es lógico, pues en esta posición se generan los momentos más grandes, momentos que se deben contrarrestar con el par ejercido en las articulaciones.

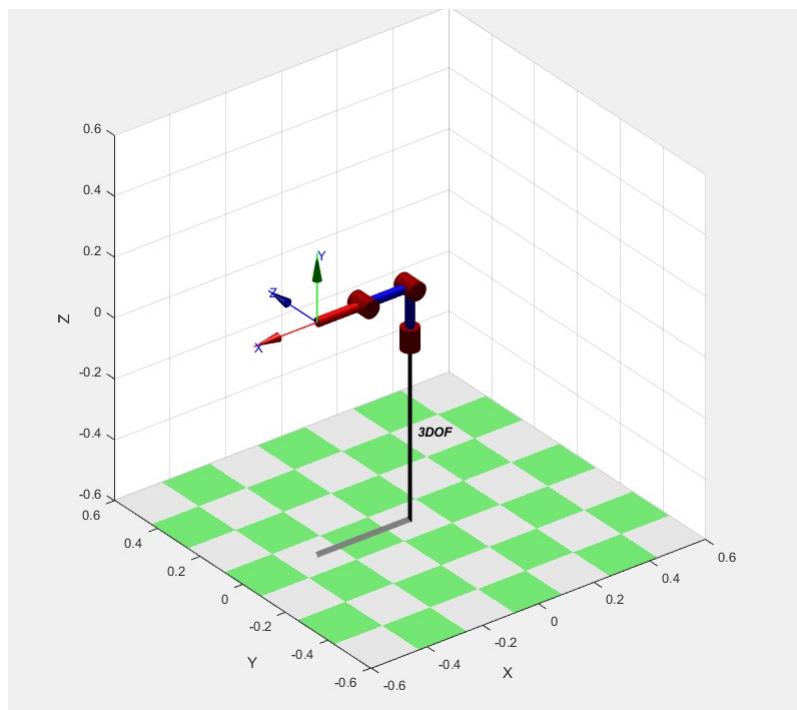


Figura 26: Posición más desfavorable para un manipulador de 3 grados de libertad

Según la matriz de Denavit-Hartenberg que se ha utilizado para definir el manipulador, esta posición corresponde al valor articular  $Q = [0, -\pi/2, 0]$

Nótese que en esta posición, el mayor requerimiento de par siempre se dará en la segunda articulación del robot. Por tanto, es esta articulación siempre se necesitará el servo de mayor par y será la que limite nuestro diseño.

Si bien evaluar en la posición más desfavorable es la opción más sencilla para encontrar el par máximo, se pueden emplear otros métodos.

- Analizando en posiciones articulares aleatorias. Si se analiza el par en un número lo suficientemente grande de posiciones aleatorias, nos aseguraremos de encontrar siempre el par máximo, o al menos un valor muy próximo a este.
- Analizando en todas las posiciones articulares. Consiste en barrer todas las posiciones posibles en busca del par máximo.

El problema de estos métodos es que requiere un número muy elevado de iteraciones para la obtención de resultados correctos. Este hecho, unido a que el método de Newton-Euler es un proceso bastante costoso computacionalmente hablando, hacen que estos métodos no sean la mejor de las opciones. Más aún si tenemos en cuenta que estos análisis se han de efectuar varias veces en la generación automática de los robots, ya que se trata de un proceso recursivo e iterativo.

Sin embargo, es posible que se presenten casos en el que no se conozca cual es la posición más desfavorable del robot. En esos casos, sí que es recomendable recurrir a estos métodos. De esta manera, esta metodología puede ser extrapolable a otros casos aparte del manipulador de 3 grados de libertad.

También hay que tener en cuenta que iterar hasta encontrar la posición donde se da el par máximo puede ser viable en robots con un número reducido de grados de libertad. En un robot de 3 grados de libertad se ha comprobado que se pueden obtener buenos resultados con un número de iteraciones del orden de  $10^4$ , lo cual puede ser aceptable. Para un robot de 6 grados de libertad, el número de iteraciones se dispara hasta cifras de aproximadamente  $10^9$ , lo cual ya resulta demasiado problemático.

Con todo esto, se llega a la conclusión de la que la mejor opción siempre es evaluar en la posición más desfavorable.

### 3.7.2. Cálculo del par dinámico

El par dinámico es el par necesario para mantener un movimiento en una posición articular  $Q$ , a una velocidad angular  $Qd$  y una aceleración angular  $Qdd$ . Para una posición determinada, el par dinámico siempre será mayor que el par estático, por lo que el análisis dinámico siempre será el determinante para la elección de servos.

Para poder realizar un análisis dinámico, es necesario comprender primero el funcionamiento de un servomotor. El par que puede ofrecer un servo viene determinado por los siguientes valores:

- ‘*Peak torque*’ o par de pico. Par que el motor es capaz de alcanzar durante un breve periodo de tiempo, normalmente en los periodos de aceleración y deceleración.
- ‘*Rated torque*’ o par nominal. Par que el motor puede entregar de manera continua a una velocidad constante.
- ‘*Stall torque*’ o par de parada. Par que el motor debe proporcionar para generar una velocidad nula, es decir, mantener la articulación bloqueada en una posición fija.

La relación entre los distintos tipos de par varía según el tipo de servo y el fabricante, pero por normal general, el valor de par de parada coincide con el par de pico, y el torque nominal es unas 3 o 4 veces más pequeño que el par de pico.

Con esto queda claro que el par de parada es el valor que debemos emplear para los análisis estáticos. Para el análisis dinámico hay que tomar como referencia el par de pico y no el nominal, pues al ser el más grande es el que nos limita.

Para poder evaluar el comportamiento de un servo se puede generar un perfil de velocidades. Esto nos permite observar como varía la velocidad, aceleración y par entregado por un servo durante el seguimiento de una trayectoria. El perfil de velocidad es una gráfica que relaciona la velocidad con el tiempo. Se pueden modelar varios tipos de perfiles de velocidad, siendo el más adecuado para estos casos el perfil trapezoidal, ya que los movimientos que se generan en las trayectorias de los robots suelen ser lo suficientemente duraderos como para alcanzar la velocidad máxima o de saturación. El perfil de velocidades trapezoidal (figura 27) consta de las siguientes etapas:

- Intervalo de aceleración ( $t_{acel}$ ): Durante este tiempo se introduce a la articulación la aceleración máxima a la que va a estar sometida hasta alcanzar la velocidad máxima a la que puede operar. En este periodo por lo tanto se trabaja con el par de pico

- Intervalo constante ( $t_{const}$ ): Durante este tiempo la articulación alcanza la velocidad máxima (o de salto) a la que va a estar sometida, considerando aceleración nula. En este periodo se da el par nominal
- Intervalo de deceleración ( $t_{decel}$ ): Durante este intervalo se introduce a la articulación una deceleración hasta alcanzar una velocidad nula.

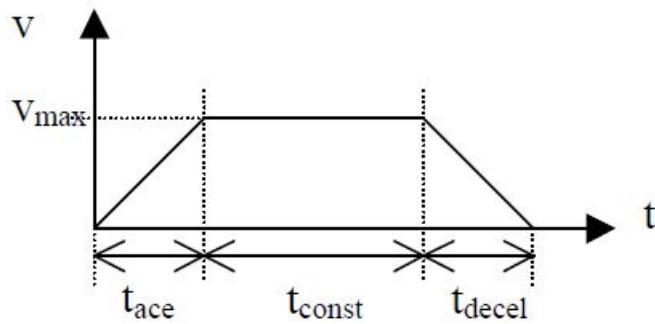


Figura 27: Perfil trapezoidal de velocidades

Por lo tanto, a la hora de encontrar el par máximo para nuestros análisis, tenemos que examinar el primer intervalo, donde la aceleración es máxima. El problema reside en que si bien la velocidad máxima es un parámetro definido por las especificaciones del servo, la aceleración no lo es, ya que depende del perfil de velocidades.

Ya que nuestros robots no están diseñados para una aplicación concreta (como lo puede estar un robot industrial, que siempre debe realizar la misma trayectoria) no se puede generar un perfil de velocidades de donde extraer el valor de aceleración máxima. Para solucionar esto, se ha decidido definir un valor fijo de aceleración máxima para todos los análisis dinámicos ( $\alpha_{max}$ ), lo suficientemente grande como para que el par real ejercido nunca sea mayor que el par calculado. Sobreestimar el valor de aceleración máxima también es positivo, y es que esto actúa como coeficiente de seguridad en los análisis. En las pruebas realizadas con el modelo real del robot se ha observado que los resultados obtenidos con este procedimiento son correctos.

$$\alpha_{max} = 5(\text{rad}/\text{s}^2) \quad (12)$$

En resumen, teniendo en cuenta lo analizado en esta sección, el cálculo del par dinámico se realizará mediante el método de Newton-Euler con las siguientes entradas:

- Posición articular  $Q$ . Posición más desfavorable del robot (como ya se ha comentado en la sección 3.7.1)
- Velocidad angular  $Qd$ .  $\frac{\omega_{max}}{2}$ .
- Aceleración angular  $Qdd$ . Valor de aceleración máxima fijado ( $\alpha_{max}$ ).

Para la velocidad se elige la velocidad máxima del servo ( $\omega_{max}$ ) partida de dos ya que es la velocidad promedio en la zona de aceleración en el perfil de velocidades. La velocidad máxima de cada servo está indicada en la tabla 1.

Puede encontrarse más información sobre estos procedimientos en las referencias [7, 6, 5].

### 3.8. Programa final para el diseño automático de manipuladores de 3 grados de libertad

Con todo lo analizado anteriormente se ha creado un programa en MATLAB capaz de generar automáticamente todos los parámetros necesarios para la construcción del manipulador de 3 grados de libertad. El diseño se efectuará en función de los datos de entrada:

- Longitud total del manipulador (en cm)
- Masa a levantar en el extremo (en kg)

Como salida, se entregarán los siguientes parámetros.

- Tipo de servos a utilizar en cada articulación
- Información de los elementos parametrizables, incluyendo:
  - Longitud del eslabón
  - Radio interno y externo del eslabón
  - Radio del redondeo interno y externo del eslabón

- Información sobre los análisis de par en las articulaciones

La idea es desarrollar un programa iterativo, en el que se parte de una configuración determinada del robot y realizar los análisis de par. Cuando la configuración actual del manipulador no satisface los requerimientos necesarios obtenidos en los análisis, la configuración de los servos cambia, volviendo al inicio del proceso. Esto se repetirá hasta que cumpla todas las necesidades. En cada iteración, se utiliza la información obtenida en los análisis para optimizar los eslabones, asegurándonos así de que el resultado final sea el óptimo. El funcionamiento del proceso está definido por el diagrama de flujo (figura 28), que consta de los siguientes bloques:

- Definición del tipo de servo: Se especifica el tipo de servo a utilizar en cada articulación a la hora de generar el robot. Por defecto, se considera que todos los servos son medios.
- Definición del robot: Se crea el objeto robot de MATLAB sobre el que se va a trabajar durante todo el proceso. Aquí se incluye:
  - Cálculo de la longitud de los elementos parametrizables
  - Cálculo de las propiedades físicas de los eslabones
  - Optimización de los eslabones. Este proceso necesita la información de los pares y las fuerzas ejercidas en cada eslabón, que proviene de los análisis de torques. En la primera iteración, todavía no se han efectuado dichos análisis, por lo que no se dispone de estos datos. Por ello, para la primera iteración se han definido unos eslabones genéricos. En el resto de iteraciones ya sí que se realiza la optimización.
- Análisis de torque estático y dinámico. Devuelve el par máximo que se necesita en cada eslabón con la configuración actual del robot
- Selección de servos. Se compara el par necesario en cada articulación con el par máximo que puede ofrecer cada tipo de servo. Se asigna el tipo de servo necesario en función de esta comparación. Tras esto, se comprueba si la configuración del tipo de servos ha cambiado con respecto a la configuración que había antes de realizar el análisis de par. Si no ha cambiado, significa que la configuración actual del manipulador es la correcta y se puede seguir

adelante en el proceso. Si ha cambiado, significa que la configuración actual no es correcta, el robot no puede generar los pares requeridos, y se vuelve al principio del proceso. En esta nueva iteración, el robot se definirá con la nueva configuración de servos obtenida.

- Comprobación de la optimización de los eslabones. Tras los análisis de par, se verifica el estado de la optimización. Con esto conseguimos que los resultados de fuerza y par obtenidos en el último análisis de par realizado se tengan en cuenta en el proceso de optimización. De este modo, siempre se va a realizar una iteración extra que asegura que los eslabones sean óptimos

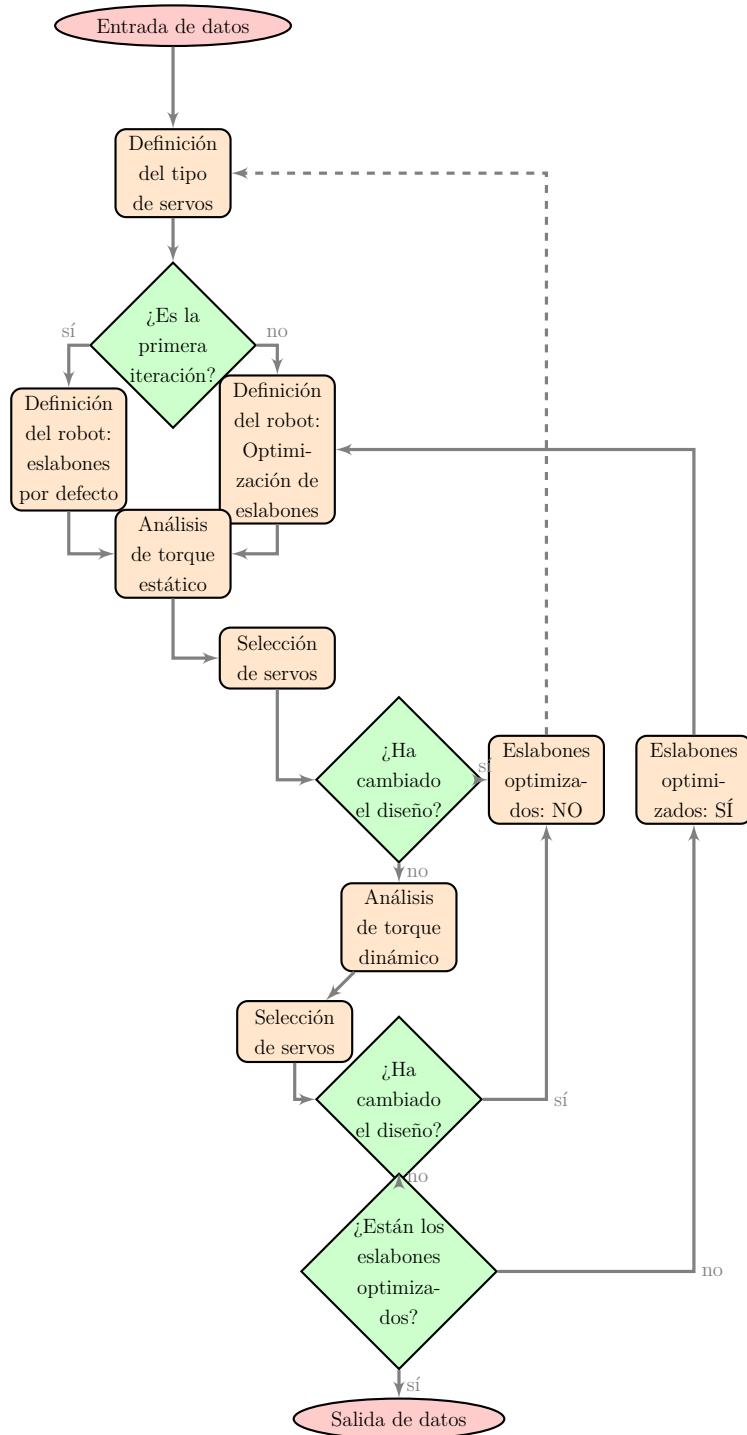


Figura 28: Diagrama de flujo

## 4. Anexo A. Código de la aplicación en Python

Este anexo muestra los códigos escritos en Python para la aplicación de diseño de la estructura abstracta del robot en base a los comportamientos solicitados por el usuario.

### 4.1. App principal

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """ Qt requirements """
5
6  import os
7
8  from PySide.QtGui import QMainWindow, QApplication, QLabel, QPixmap
9  from PySide.QtGui import QAction, QMenu, QMessageBox, QComboBox
10 from PySide import QtCore, QtGui # NEW WINDOW
11
12 """ Importing a Window form (.ui file) """
13 from output import Ui_MainWindow
14 from output_man import Ui_ManipulatorWindow
15 from v2 import Ui_Form2 # NEW WINDOW
16 from robot_movil import Ui_param_robot_movil
17
18 from robot_movil import Ui_param_robot_movil
19
20 import sys
21 sys.path.insert(0, './param')
22
23 from hex import Hexapodo
24 from serp import Serpiente
25 from human import Humanoide
26
27 """ Matlab engine for Python """
28 import matlab.engine
29
30 """ pySUMO """
31 import sys
32
33 sys.path.append('src')
34 from tempfile import mkdtemp
35 import math
36
37 import json
38 import subprocess
```

```
39
40 from pysumo import parser
41 from pysumo.indexabstractor import *
42 from pysumo.syntaxcontroller import Ontology
43
44
45 from final_link_fillet_final import *
46 from generar_rueda import *
47
48
49 # sumo = Ontology('/home/kaiser/.pysumo/ontologias/fullOntology_proofs.kif', ...
50 #                   name='SUMO')
50 sumo = Ontology('./fullOntology_proofs.kif', name='SUMO')
51
52 with open(sumo.path) as f:
53     mergeKif = parser.kifparse(f, sumo)
54 index = IndexAbstractor()
55 index.update_index(mergeKif)
56
57
58 def query(variant, root):
59     # variant MUST NOT CONTAIN ANY CAPITAL LETTER.
60     q = index.get_graph([(0, variant)], root=root)
61     qResult = q.nodes[0:(len(q.nodes) - 1)] # Deleting all relations and the ...
62     #           last node (root)
62     #     qResult = q.nodes[0:(len(q.nodes)-1)] #We could do the same with Relations
63     return qResult
64
65
66 def reversedQuery(variant, root):
67     # variant MUST NOT CONTAIN ANY CAPITAL LETTER.
68     q = index.get_graph([(0, variant)], root=root, major=1, minor=2)
69     # print("Los nodos son: {}".format(q.nodes))
70     # print("El resultado de la búsqueda es: ", q.nodes[0:(len(q.nodes) - 1)])
71     qResult = q.nodes[0:(len(q.nodes) - 1)] # Deleting all relations and the ...
72     #           last node (root)
72     #     qResult = q.nodes[0:(len(q.nodes)-1)] #We could do the same with Relations
73     return qResult
74
75
76 actions = {"<<Elige tu opción>>": "Sin elegir", "Andar": "RobotWalking", ...
77             "Correr": "RobotRunning",
77             "Circular": "RobotRolling", "Subir escaleras": "RobotGoingUpSteps", ...
78             "Bajar escaleras": "RobotGoingDownSteps",
78             "Esquivar obstáculos": "RobotObstacleAvoiding", "Mantener el ...",
79             "equilibrio": "RobotSelfBalancing",
79             "Seguir líneas": "RobotLineTracking", "Agarrar": "RobotGrasping", ...
80             "Transportar": "RobotCarrying"}  

81
82 # Functions
```

```
83 def manipulatorFN(self):
84     if self.ManipulatorWindow is None:
85         self.ManipulatorWindow = Manipulator()
86         self.ManipulatorWindow.show()
87
88
89 def humanoidFN(self):
90     print("Humanoide")
91     if self.HumanWindow is None:
92         self.HumanWindow = Robot_humanoide()
93         self.HumanWindow.show()
94
95
96 def hexapodFN(self):
97
98     if self.HexWindow is None:
99         self.HexWindow = Robot_hexapodo()
100        self.HexWindow.show()
101    print("Hexapodo")
102
103
104 def twoWheelCarFN(self):
105     print("Coche de dos ruedas")
106     if self.RobotMovilWindow is None:
107         self.RobotMovilWindow = Robot_movil(2)
108         self.RobotMovilWindow.show()
109
110 def fourWheelCarFN(self):
111     print("Coche de cuatro ruedas")
112     if self.RobotMovilWindow is None:
113         self.RobotMovilWindow = Robot_movil(4)
114         self.RobotMovilWindow.show()
115
116
117 def quadrupedFN(self):
118     print("Cuadrupedo")
119
120
121 # Function Map
122 definedRobots = {"robindArm": manipulatorFN, "RobInd": humanoidFN, ...
123                 "robindSpider": hexapodFN,
124                 "robind2Wheeled": twoWheelCarFN, "robind4Wheeled": ...
125                               fourWheelCarFN, "robindDog": quadrupedFN}
126
127 # defined_robots = ...
128     ['RobInd', 'robindSpider', 'robindArm', 'robind2Wheeled', 'robind4Wheeled', 'robindDog', 'robindSnake', ...
129
130 class MainWindow(QMainWindow, Ui_MainWindow):
131     def __init__(self, parent=None):
132         super(MainWindow, self).__init__(parent)
133         self.setupUi(self)
```

```

131         self.searchingRobotButton.clicked.connect(self.searching_structural_requirement)
132         self.customizeRobotButton.clicked.connect(self.customizingRobot)
133         self.window2 = None # NEW WINDOW
134         self.ManipulatorWindow = None
135         self.RobotMovilWindow = None
136         self.HexWindow = None
137         self.SerpWindow = None
138         self.HumanWindow = None
139
140         # Some necessary variables
141         self.thereAreNOSomeFoundedRobots = True
142         self.requiredRobots = []
143
144         self.cb_andar.clicked.connect(lambda: self.pulsado("RobotWalking", ...
145                                         self.cb_andar))
145         self.cb_subir_es.clicked.connect(lambda: ...
146                                         self.pulsado("RobotGoingUpSteps", self.cb_subir_es))
146         self.cb_circular.clicked.connect(lambda: self.pulsado("RobotRolling", ...
147                                         self.cb_circular))
147         self.cb_seguir_linea.clicked.connect(lambda: ...
148                                         self.pulsado("RobotLineTracking", self.cb_seguir_linea))
148         self.cb_agarrar.clicked.connect(lambda: self.pulsado("RobotGrasping", ...
149                                         self.cb_agarrar))
149         self.cb_correr.clicked.connect(lambda: self.pulsado("RobotRunning", ...
150                                         self.cb_correr))
150         self.cb_bajar_escaleras.clicked.connect(lambda: ...
151                                         self.pulsado("RobotGoingDownSteps", self.cb_bajar_escaleras))
151         self.cb_esquivar.clicked.connect(lambda: ...
152                                         self.pulsado("RobotObstacleAvoiding", self.cb_esquivar))
152         self.cb_equilibrio.clicked.connect(lambda: ...
153                                         self.pulsado("RobotSelfBalancing", self.cb_equilibrio))
153         self.cb_transportar.clicked.connect(lambda: ...
154                                         self.pulsado("RobotCarrying", self.cb_transportar))
154
155         self.acciones_a_realizar = []
156         self.partes_requeridas = []
157
158     def pulsado(self, accion, boton):
159         if boton.isChecked():
160             self.acciones_a_realizar.append(accion)
161         else:
162             self.acciones_a_realizar.remove(accion)
163
164     def searching_structural_requirement(self):
165         self.partes_requeridas.clear()
166         self.suggestedRobots.clear()
167         self.ChosenRobot.clear()
168         self.thereAreNOSomeFoundedRobots = True
169
170         if len(self.acciones_a_realizar):
171             for action in self.acciones_a_realizar:

```

```

172         self.partes_requeridas.append(self.realizar_busqueda_partes_requeridas(action))
173
174     # Aplanar lista de listas (flat list)
175     self.partes_requeridas = [val for sublist in self.partes_requeridas ...
176         for val in sublist]
177     # Eliminar duplicados
178     self.partes_requeridas = list(set(self.partes_requeridas))
179     self.realizar_busqueda_robots_segun_partes_requeridas(self.partes_requeridas)
180
181     def realizar_busqueda_partes_requeridas(self, action_ontology):
182         queriedGraph = reversedQuery('structuralrequirement', action_ontology)
183         return queriedGraph
184
185     def realizar_busqueda_robots_segun_partes_requeridas(self, partes_requeridas):
186         robots_validos = []
187         for parte_requerida in partes_requeridas:
188             robot_que_usa_parte = reversedQuery('uses', parte_requerida)
189             robot_que_usa_parte2 = robot_que_usa_parte[:]
190             for robots in robot_que_usa_parte:
191                 # print(robots)
192                 if not robots in definedRobots.keys():
193                     robot_que_usa_parte2.remove(robots)
194             robots_validos.append(robot_que_usa_parte2[:])
195
196             if len(robots_validos) ≥ 2:
197                 robots_validos2 = robots_validos[0]
198                 for i in range(1, len(robots_validos)):
199                     print(robots_validos2)
200                     robots_validos2 = ...
201                     list(set(robots_validos2).intersection(robots_validos[i]))
202                     print("Los robots validos son: ", robots_validos2)
203             else:
204                 robots_validos2 = [val for sublist in robots_validos for val in ...
205                     sublist]
206
207             print("Robot(s) que cumple {} son: {}".format(parte_requerida, ...
208                 robots_validos2))
209             for idx, val in enumerate(robots_validos2):
210                 self.suggestedRobots.append('%d %s' % (idx, val))
211                 self.ChosenRobot.addItem('%d %s' % (idx, val))
212                 # print(robot_que_usa_parte[0])
213                 if len(robots_validos2)>0:
214                     self.thereAreNOSomeFoundedRobots = False
215
216             def customizingRobot(self):
217
218                 textChosenRobot = self.ChosenRobot.currentText()[1:].strip()
219                 print("Robot marcado para parametrizar {}".format(textChosenRobot))
220
221                 if self.thereAreNOSomeFoundedRobots:
222                     msg = QMessageBox()

```

```
219         msg.setIcon(QMessageBox.Warning)
220         msg.setText(
221             "No hay ninguna busqueda de robot realizada. Por favor, ...
222             seleccione una accion antes de continuar.")
223         msg.exec_()
224
225     elif textChosenRobot == '<<Elige tu robot>>':
226         msg = QMessageBox()
227         msg.setIcon(QMessageBox.Warning)
228         msg.setText("Por favor, seleccione un robot de entre los sugeridos ...
229             antes de continuar.")
230         msg.exec_()
231
232     else:
233         msg = QMessageBox()
234         msg.setIcon(QMessageBox.Warning)
235         msg.setText(
236             "El robot elegido es: %s. Quieres comenzar el proceso de ...
237             parametrizacion de este robot? Si prefieres elegir otro ...
238             robot, cancela el proceso." %
239             textChosenRobot)
240         msg.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
241         retVal = msg.exec_()
242
243     if retVal == QMessageBox.Ok:
244         print("Comenzando proceso de parametrizacion")
245         functionToCall = definedRobots[textChosenRobot]
246         functionToCall(self)
247     else:
248         print("Elige un nuevo robot")
249
250
251
252 # NEW WINDOW
253 class Manipulator(QtGui.QWidget, Ui_ManipulatorWindow):
254     def __init__(self, parent=None):
255         QtGui.QWidget.__init__(self, parent)
256         self.setupUi(self)
257         self.customizingRobotButton.clicked.connect(self.customizingRobotProcess)
258
259     def customizingRobotProcess(self):
260         object_shape = self.objectShape.currentText()
261         degree_of_freedom = self.DOFs.currentText()
262         weight = self.objectWeight.text()
263         length = self.workSpace.text()
264
265         if (object_shape == "<<Elige tu opcion>>") or (degree_of_freedom == ...
266             "<<Elige tu opcion>>") or (
267                 weight == '') or (length == ''):
268             msg = QMessageBox()
269             msg.setIcon(QMessageBox.Warning)
```

```

264         msg.setText("Por favor, asegurate de haber rellenado todos los ...  
265             campos que se piden.")  
266         msg.exec_()  
267     else:  
268         eng = matlab.engine.start_matlab()  
269         matlab_files_dir = os.path.dirname(os.path.realpath(__file__)) + ...  
270             "/parte_matlab/rvctools"  
271         eng.addpath(matlab_files_dir, nargout=0)  
272         matlab_files_dir = os.path.dirname(os.path.realpath(__file__)) + ...  
273             "/parte_matlab"  
274         print(matlab_files_dir)  
275         eng.addpath(matlab_files_dir, nargout=0)  
276         ret = eng.calcular_robot_3DOF(float(self.workSpace.text())/100, ...  
277             float(self.objectWeight.text()), nargout=4)  
278  
279         # print(ret[1][0])  
280         print(ret[1][0][0]*1000, ret[1][0][1]*1000, ret[1][0][2]*1000)  
281         parametrizar_viga(ret[1][0][0]*1000, ret[1][0][1]*1000, ...  
282             ret[1][0][2]*1000,  
283                 ret[1][0][3]*1000,ret[1][0][4]*1000,"1_L_{ }_M_{ }".format(self.workSpace.  
284  
285         parametrizar_viga(ret[2][0][0] * 1000, ret[2][0][1] * 1000, ...  
286             ret[2][0][2] * 1000,  
287                 ret[2][0][3] * 1000, ret[2][0][4] * 1000, ...  
288                     "2_L_{ }_M_{ }".format(self.workSpace.text(),self.objectWeight.text()))  
289  
290         parametrizar_viga(ret[3][0][0] * 1000, ret[3][0][1] * 1000, ...  
291             ret[3][0][2] * 1000,  
292                 ret[3][0][3] * 1000, ret[3][0][4] * 1000, ...  
293                     "3_L_{ }_M_{ }".format(self.workSpace.text(),self.objectWeight.text()))  
294  
295         data = {  
296             'modulo motor 1': "tipo {}".format(ret[0][0][0]),  
297             'modulo motor 2': "tipo {}".format(ret[0][0][1]),  
298             'modulo motor 3': "tipo {}".format(ret[0][0][2]),  
299             'link 1': {'longitud (mm)': ret[1][0][0]*1000,  
300                 'radio externo (mm)': ret[1][0][1]*1000,  
301                 'radio interno (mm)': ret[1][0][2] * 1000,  
302                 'radio redondeo externo (mm)': ret[1][0][3] * 1000,  
303                 'radio redondeo interno (mm)': ret[1][0][4] * 1000},  
304             'link 2': {'longitud (mm)': ret[2][0][0] * 1000,  
305                 'radio externo (mm)': ret[2][0][1] * 1000,  
306                 'radio interno (mm)': ret[2][0][2] * 1000,  
307                 'radio redondeo externo (mm)': ret[2][0][3] * 1000,  
308                 'radio redondeo interno (mm)': ret[2][0][4] * 1000},  
309             'link 3': {'longitud (mm)': ret[3][0][0] * 1000,  
310                 'radio externo (mm)': ret[3][0][1] * 1000,  
311                 'radio interno (mm)': ret[3][0][2] * 1000,  
312                 'radio redondeo externo (mm)': ret[3][0][3] * 1000,  
313                 'radio redondeo interno (mm)': ret[3][0][4] * 1000}

```

```
306         }
307
308         with open('archivos_generados/data_eslabones.json', 'w') as f:
309             json.dump(data, f)
310
311         print("ACABADO")
312
313         # for i in ret:
314         #     print(i)
315         eng.quit()
316
317
318 # NEW WINDOW
319 class Robot_movil(QtGui.QWidget, Ui_param_robot_movil):
320     def __init__(self, tipo, parent=None):
321         QtGui.QWidget.__init__(self, parent)
322         self.ln_movil_batalla = 700
323         self.setupUi(self)
324
325         self.btn_dimensionar.clicked.connect(self.dimensionar)
326         self.btn_generar.clicked.connect(self.generar)
327
328
329     def dimensionar(self):
330
331         vel_ang = float(self.vel_ang.text())
332         vel_lin = float(self.vel_lin.text())
333
334         self.r_max = 45
335
336         w_motor =[ 4.1887902, 6.178465545]
337         self.r = [vel_lin/wr for wr in w_motor]
338         self.l = [wr*r/vel_ang for (wr, r) in zip(w_motor, self.r)]
339         print(self.r)
340         print(self.l)
341         self.resultado.setText("radio = {} cm, ancho vias = {} ...
342             cm".format(round(self.r[0]*100,1),round(self.l[0]*100,1)))
343         #generar_rueda(r[1]*1000)
344
345         if self.r[0] * 1000 > self.r_max:
346             print("radio impresion dyn = {}".format(self.r[1] * 1000))
347             self.resultado.setText(
348                 "radio = {} cm, ancho vias = {} cm".format(round(self.r[1] * ...
349                     100, 1), round(self.l[1] * 100, 1)))
350             self.seleccionar_modulo(1)
351             self.tipo_servo = 1
352             #
353         else:
354             print("radio impresion = {}".format(self.r[0] * 1000))
355             self.resultado.setText(
```

```

354         "radio = {} cm, ancho_vias = {} cm".format(round(self.r[0] * ...
355             100, 1), round(self.l[0] * 100, 1)))
356     self.seleccionar_modulo(0)
357     self.tipo_servo = 0
358
359     def generar(self):
360
361         generar_rueda(self.r[self.tipo_servo] * 1000)
362
363         data = {
364             'radio': "{} mm".format(round(self.r[self.tipo_servo] * 1000, 1)),
365             'tipo servo ': "{}".format(self.tipo_servo),
366             'ancho vias': "{} mm".format(round(self.l[1] * 1000, 1))
367         }
368
369         with open('archivos_generados/data_rueda.json', 'w') as f:
370             json.dump(data, f)
371
372
373         if self.num_motores== 2:
374             cmd = 'rosrun xacro xacro -o ./param/model_movil_diferencial.urdf ...
375                 ./param/robot_movil_diferencial.xacro'
376         else:
377             cmd = 'rosrun xacro xacro -o ./param/model_movil_skid_steer.urdf ...
378                 ./param/robot_movil_skid_steer.xacro'
379
380         radio_rueda = 'radio_rueda:=' + str(self.r[self.tipo_servo] * 1000)
381         ancho_vias = 'ancho_vias:=' + str(round(self.l[1] * 1000))
382         batalla = 'batalla:=' + str(self.ln_movil_batalla)
383
384         cmd = str(
385             cmd + ' ' + radio_rueda + ' ' + ancho_vias + ' ' + batalla + ' ')
386
387         # cmd = 'rosrun xacro xacro -o model_biped.urdf bipedo.xacro ...
388         # anchura_cadera:=1 long_x_pie:=32 long_y_pie:=321 ...
389         # long_pieza_pasiva:=2 radio_servo:=42 long_servo:=1'
390
391         p = subprocess.Popen(cmd, stdout=subprocess.PIPE, shell=True)
392         out, err = p.communicate()
393
394
395         print("ACABADO")
396
397     def seleccionar_modulo(self,tipo_servo):
398
399         VD = float(self.vel_lin.text()) # vel lineal
400         VC = self.r[tipo_servo] # radio
401         VI = 80 # efficiencia
402         VG = VD #aceleracion deseada
403         VF = 12 # voltaje alimentacion

```

```

400     VE = float(self.inclinacion.text()) # inclinacion maxima (en grados)
401     VA = float(self.masa.text()) # Masa total
402     VB = 4 # numero de motores
403     VH = 1 # punto de funcionamiento (minutos)
404
405     self.torques = [0.5,1.5]
406     torque_servo = self.torques[tiposervo]
407
408     VJ = (VD / VC) # Angular Velocity
409     self.num_motores = (100 / VI) * (VG + 9.81 * (math.sin(3.1415926 * VE / ...
410         180))) * VA * VC / torque_servo # torque
411
412     print("numero motores: {}".format(self.num_motores))
413     if self.num_motores < 2: print("Necesarios 2 motores")
414     else: print("Necesarios 4 motores")
415
416 class Robot_humanoide(QtGui.QWidget, Humanoide):
417     def __init__(self, parent=None):
418         QtGui.QWidget.__init__(self, parent)
419         self.setupUi(self)
420         self.btn_salir.clicked.connect(exit)
421         self.btn_generar.clicked.connect(self.generar_urdf)
422
423     def generar_urdf(self):
424
425
426         cmd = 'rosrun xacro xacro -o ./param/model_biped.urdf ./param/bipedo.xacro'
427         anchura_cadera = 'anchura_cadera:=' + self.ln_bip_anchura_cadera.text()
428         long_x_pie = 'long_x_pie:=' + self.ln_bip_long_x_pie.text()
429         long_y_pie = 'long_y_pie:=' + self.ln_bip_long_y_pie.text()
430         long_pieza_pasiva = 'long_pieza_pasiva:=' + ...
431             self.ln_bip_long_pieza_pasiva.text()
432         radio_servo = 'radio_servo:=' + self.ln_bip_radio_servo.text()
433         long_servo = 'long_servo:=' + self.ln_bip_long_servo.text()
434
435         cmd = str(cmd + ' ' + anchura_cadera + ' ' + long_x_pie + ' ' + ...
436             long_y_pie + ' ' + long_pieza_pasiva + ' ' + radio_servo + ' ' + ...
437             long_servo + ' ')
438
439
440
441         p = subprocess.Popen(cmd, stdout=subprocess.PIPE, shell=True)
442         out, err = p.communicate()
443
444
445         class Robot_serp(QtWidgets.QWidget, Serpiente):
446             def __init__(self, parent=None):
447                 QtWidgets.QWidget.__init__(self, parent)
448                 self.setupUi(self)
449                 self.btn_salir.clicked.connect(exit)

```

```

447         self.btn_generar.clicked.connect(self.generar_urdf)
448
449     def generar_urdf(self):
450         cmd = 'rosrun xacro xacro -o ./param/model_serpiente.urdf ...'
451         . ./param/serpiente.xacro'
452         numero_servo = 'numero_servo:= ' + self.ln_ser_n_servos.text()
453         cmd = str(
454             cmd + ' ' + numero_servo + ' ')
455
456         # cmd = 'rosrun xacro xacro -o model_biped.urdf bipedo.xacro ...'
457         anchura_cadera:=1 long_x_pie:=32 long_y_pie:=321 ...
458         long_pieza_pasiva:=2 radio_servo:=42 long_servo:=1'
459
460
461     class Robot_hexapodo(QtGui.QWidget, Hexapodo):
462         def __init__(self, parent=None):
463             QtGui.QWidget.__init__(self, parent)
464             self.setupUi(self)
465             self.btn_salir.clicked.connect(exit)
466             self.btn_generar.clicked.connect(self.generar_urdf)
467
468         def generar_urdf(self):
469
470             if int(str(self.ln_hex_n_servos.text())) == 3:
471                 cmd = 'rosrun xacro xacro -o ./param/model_hex_3.urdf ...'
472                 . ./param/hex_3_dof.xacro'
473             else:
474                 cmd = 'rosrun xacro xacro -o ./param/model_hex_4.urdf ...'
475                 . ./param/hex_4_dof.xacro'
476
477             long_x_cuerpo = 'long_x_cuerpo:= ' + self.ln_hex_long_x_cuerpo.text()
478             long_y_cuerpo = 'long_y_cuerpo:= ' + self.ln_hex_long_y_cuerpo.text()
479             long_coxa = 'long_coxa:= ' + self.ln_hex_long_coxa.text()
480             long_femur = 'long_femur:= ' + self.ln_hex_long_femur.text()
481             long_tibia = 'long_tibia:= ' + self.ln_hex_long_tibia.text()
482
483             cmd = str(
484                 cmd + ' ' + long_x_cuerpo + ' ' + long_y_cuerpo + ' ' + long_coxa + ...
485                 ' ' + long_femur + ' ' + long_tibia
486                 + ' ')
487
488             p = subprocess.Popen(cmd, stdout=subprocess.PIPE, shell=True)
489             out, err = p.communicate()
490
491     if __name__ == '__main__':
492         app = QApplication(sys.argv)
493         frame = MainWindow()
494         frame.show()

```

492 app.exec\_()

## 4.2. Código ventana principal

```
1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file 'mainwindow.ui'
4 #
5 # Created: Tue Jun 14 01:37:57 2016
6 #      by: pyside-uic 0.2.15 running on PySide 1.2.4
7 #
8 # WARNING! All changes made in this file will be lost!
9
10 from PySide import QtCore, QtGui
11
12 class Ui_MainWindow(object):
13     def setupUi(self, MainWindow):
14         MainWindow.setObjectName("MainWindow")
15         MainWindow.resize(559, 438)
16         self.centralWidget = QtGui.QWidget(MainWindow)
17         self.centralWidget.setObjectName("centralWidget")
18         self.RobotActions = QtGui.QComboBox(self.centralWidget)
19         self.RobotActions.setGeometry(QtCore.QRect(10, 40, 201, 27))
20         self.RobotActions.setObjectName("RobotActions")
21         self.RobotActions.addItem("")
22         self.RobotActions.addItem("")
23         self.RobotActions.addItem("")
24         self.RobotActions.addItem("")
25         self.RobotActions.addItem("")
26         self.RobotActions.addItem("")
27         self.RobotActions.addItem("")
28         self.RobotActions.addItem("")
29         self.RobotActions.addItem("")
30         self.RobotActions.addItem("")
31         self.RobotActions.addItem("")
32         self.searchResult = QtGui.QTextEdit(self.centralWidget)
33         self.searchResult.setGeometry(QtCore.QRect(10, 290, 541, 91))
34         self.searchResult.setObjectName("searchResult")
35         self.label = QtGui.QLabel(self.centralWidget)
36         self.label.setGeometry(QtCore.QRect(10, 20, 231, 17))
37         self.label.setObjectName("label")
38         self.searchingRobotButton = QtGui.QPushButton(self.centralWidget)
39         self.searchingRobotButton.setGeometry(QtCore.QRect(260, 40, 161, 27))
40         self.searchingRobotButton.setObjectName("searchingRobotButton")
41         self.label_2 = QtGui.QLabel(self.centralWidget)
42         self.label_2.setGeometry(QtCore.QRect(10, 270, 141, 17))
43         self.label_2.setObjectName("label_2")
```

```
44         self.suggestedRobots = QtGui.QTextEdit(self.centralWidget)
45         self.suggestedRobots.setGeometry(QtCore.QRect(10, 120, 161, 121))
46         self.suggestedRobots.setObjectName("suggestedRobots")
47         self.label_3 = QtGui.QLabel(self.centralWidget)
48         self.label_3.setGeometry(QtCore.QRect(10, 100, 131, 17))
49         self.label_3.setObjectName("label_3")
50         self.ChosenRobot = QtGui.QComboBox(self.centralWidget)
51         self.ChosenRobot.setGeometry(QtCore.QRect(180, 120, 201, 27))
52         self.ChosenRobot.setObjectName("ChosenRobot")
53
54         self.customizeRobotButton = QtGui.QPushButton(self.centralWidget)
55         self.customizeRobotButton.setGeometry(QtCore.QRect(390, 120, 161, 27))
56         self.customizeRobotButton.setObjectName("customizeRobotButton")
57         MainWindow.setCentralWidget(self.centralWidget)
58         self.menuBar = QtGui.QMenuBar(MainWindow)
59         self.menuBar.setGeometry(QtCore.QRect(0, 0, 559, 25))
60         self.menuBar.setObjectName("menuBar")
61         MainWindow.setMenuBar(self.menuBar)
62         self.mainToolBar = QtGui.QToolBar(MainWindow)
63         self.mainToolBar.setObjectName("mainToolBar")
64         MainWindow.addToolBar(QtCore.Qt.TopToolBarArea, self.mainToolBar)
65         self.statusBar = QtGui.QStatusBar(MainWindow)
66         self.statusBar.setObjectName("statusBar")
67         MainWindow.setStatusBar(self.statusBar)
68
69         self.retranslateUi(MainWindow)
70         QtCore.QMetaObject.connectSlotsByName(MainWindow)
71
72     def retranslateUi(self, MainWindow):
73         MainWindow.setWindowTitle(QtGui.QApplication.translate("MainWindow", ...
74             "MainWindow", None, QtGui.QApplication.UnicodeUTF8))
75         self.RobotActions.setItemText(0, ...
76             QtGui.QApplication.translate("MainWindow", "<>Elije tu opcion>", ...
77                 None, QtGui.QApplication.UnicodeUTF8))
78         self.RobotActions.setItemText(1, ...
79             QtGui.QApplication.translate("MainWindow", "Andar", None, ...
80                 QtGui.QApplication.UnicodeUTF8))
81         self.RobotActions.setItemText(2, ...
82             QtGui.QApplication.translate("MainWindow", "Correr", None, ...
83                 QtGui.QApplication.UnicodeUTF8))
84         self.RobotActions.setItemText(3, ...
85             QtGui.QApplication.translate("MainWindow", "Subir escaleras", None, ...
86                 QtGui.QApplication.UnicodeUTF8))
87         self.RobotActions.setItemText(4, ...
88             QtGui.QApplication.translate("MainWindow", "Bajar escaleras", None, ...
89                 QtGui.QApplication.UnicodeUTF8))
90         self.RobotActions.setItemText(5, ...
91             QtGui.QApplication.translate("MainWindow", "Circular", None, ...
92                 QtGui.QApplication.UnicodeUTF8))
93         self.RobotActions.setItemText(6, ...
94             QtGui.QApplication.translate("MainWindow", "Esquivar obstaculos", ...
```

```

        None, QtGui.QApplication.UnicodeUTF8))
81     self.RobotActions.setItemText(7, ...
        QtGui.QApplication.translate("MainWindow", "Seguir lineas", None, ...
        QtGui.QApplication.UnicodeUTF8))
82     self.RobotActions.setItemText(8, ...
        QtGui.QApplication.translate("MainWindow", "Mantener el ...
        equilibrio", None, QtGui.QApplication.UnicodeUTF8))
83     self.RobotActions.setItemText(9, ...
        QtGui.QApplication.translate("MainWindow", "Agarrar", None, ...
        QtGui.QApplication.UnicodeUTF8))
84     self.RobotActions.setItemText(10, ...
        QtGui.QApplication.translate("MainWindow", "Transportar", None, ...
        QtGui.QApplication.UnicodeUTF8))
85     self.label.setText(QtGui.QApplication.translate("MainWindow", "Que ...
        quieres que haga tu robot?", None, QtGui.QApplication.UnicodeUTF8))
86     self.searchingRobotButton.setText(QtGui.QApplication.translate("MainWindow", ...
        "Realizar busqueda", None, QtGui.QApplication.UnicodeUTF8))
87     self.label_2.setText(QtGui.QApplication.translate("MainWindow", ...
        "Ventana informativa", None, QtGui.QApplication.UnicodeUTF8))
88     self.label_3.setText(QtGui.QApplication.translate("MainWindow", "Robots ...
        sugeridos", None, QtGui.QApplication.UnicodeUTF8))
89     self.ChosenRobot.setItemText(0, ...
        QtGui.QApplication.translate("MainWindow", "<<Elige tu robot>>", ...
        None, QtGui.QApplication.UnicodeUTF8))
90     self.ChosenRobot.setItemText(1, ...
        QtGui.QApplication.translate("MainWindow", "0", None, ...
        QtGui.QApplication.UnicodeUTF8))
91
92
93     self.customizeRobotButton.setText(QtGui.QApplication.translate("MainWindow", ...
        "Parametrizar robot", None, QtGui.QApplication.UnicodeUTF8))

```

## 4.3. Ventanas auxiliares

### 4.3.1. Manipulador

```

1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file 'Manipulator.ui'
4  #
5  # Created: Tue Oct 25 12:04:23 2016
6  #       by: pyside-uic 0.2.15 running on PySide 1.2.4
7  #
8  # WARNING! All changes made in this file will be lost!
9
10 from PySide import QtCore, QtGui

```

```
11
12 class Ui_ManipulatorWindow(object):
13     def setupUi(self, ManipulatorWindow):
14         ManipulatorWindow.setObjectName("ManipulatorWindow")
15         ManipulatorWindow.resize(562, 475)
16         self.centralWidget = QtGui.QWidget(ManipulatorWindow)
17         self.centralWidget.setObjectName("centralWidget")
18         self.customizingRobotButton = QtGui.QPushButton(self.centralWidget)
19         self.customizingRobotButton.setGeometry(QtCore.QRect(180, 300, 161, 27))
20         self.customizingRobotButton.setObjectName("customizingRobotButton")
21         self.textBrowser = QtGui.QTextBrowser(self.centralWidget)
22         self.textBrowser.setGeometry(QtCore.QRect(20, 340, 521, 81))
23         self.textBrowser.setObjectName("textBrowser")
24         self.textBrowser_2 = QtGui.QTextBrowser(self.centralWidget)
25         self.textBrowser_2.setGeometry(QtCore.QRect(20, 20, 521, 81))
26         self.textBrowser_2.setObjectName("textBrowser_2")
27         self.widget = QtGui.QWidget(self.centralWidget)
28         self.widget.setGeometry(QtCore.QRect(20, 116, 521, 29))
29         self.widget.setObjectName("widget")
30         self.horizontalLayout = QtGui.QHBoxLayout(self.widget)
31         self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
32         self.horizontalLayout.setObjectName("horizontalLayout")
33         self.label_2 = QtGui.QLabel(self.widget)
34         self.label_2.setObjectName("label_2")
35         self.horizontalLayout.addWidget(self.label_2)
36         self.objectShape = QtGui.QComboBox(self.widget)
37         self.objectShape.setObjectName("objectShape")
38         self.objectShape.addItem("")
39         self.objectShape.addItem("")
40         self.objectShape.addItem("")
41         self.objectShape.addItem("")
42         self.objectShape.addItem("")
43         self.objectShape.addItem("")
44         self.objectShape.addItem("")
45         self.objectShape.addItem("")
46         self.objectShape.setItemText(7, "")
47         self.horizontalLayout.addWidget(self.objectShape)
48         self.widget1 = QtGui.QWidget(self.centralWidget)
49         self.widget1.setGeometry(QtCore.QRect(20, 160, 521, 131))
50         self.widget1.setObjectName("widget1")
51         self.verticalLayout = QtGui.QVBoxLayout(self.widget1)
52         self.verticalLayout.setContentsMargins(0, 0, 0, 0)
53         self.verticalLayout.setObjectName("verticalLayout")
54         self.gridLayout = QtGui.QGridLayout()
55         self.gridLayout.setObjectName("gridLayout")
56         self.label_3 = QtGui.QLabel(self.widget1)
57         self.label_3.setObjectName("label_3")
58         self.gridLayout.addWidget(self.label_3, 0, 0, 1, 2)
59         self.objectWeight = QtGui.QLineEdit(self.widget1)
60         self.objectWeight.setObjectName("objectWeight")
61         self.gridLayout.addWidget(self.objectWeight, 0, 2, 1, 1)
```

```
62         self.label_4 = QtGui.QLabel(self.widget1)
63         self.label_4.setObjectName("label_4")
64         self.gridLayout.addWidget(self.label_4, 1, 0, 1, 1)
65         self.workSpace = QtGui.QLineEdit(self.widget1)
66         self.workSpace.setObjectName("workSpace")
67         self.gridLayout.addWidget(self.workSpace, 1, 2, 1, 1)
68         self.verticalLayout.addLayout(self.gridLayout)
69         self.label_5 = QtGui.QLabel(self.widget1)
70         self.label_5.setObjectName("label_5")
71         self.verticalLayout.addWidget(self.label_5)
72         self.DOFs = QtGui.QComboBox(self.widget1)
73         self.DOFs.setObjectName("DOFs")
74         self.DOFs.addItem("")
75         self.DOFs.addItem("")
76         self.DOFs.addItem("")
77         self.verticalLayout.addWidget(self.DOFs)
78         '''ManipulatorWindow.setCentralWidget(self.centralWidget)
79         self.menuBar = QtGui.QMenuBar(ManipulatorWindow)
80         self.menuBar.setGeometry(QtCore.QRect(0, 0, 562, 25))
81         self.menuBar.setObjectName("menuBar")
82         ManipulatorWindow.setMenuBar(self.menuBar)
83         self.mainToolBar = QtGui.QToolBar(ManipulatorWindow)
84         self.mainToolBar.setObjectName("mainToolBar")
85         ManipulatorWindow.addToolBar(QtCore.Qt.TopToolBarArea, self.mainToolBar)
86         self.statusBar = QtGui.QStatusBar(ManipulatorWindow)
87         self.statusBar.setObjectName("statusBar")
88         ManipulatorWindow.setStatusBar(self.statusBar)'''

89
90         self.retranslateUi(ManipulatorWindow)
91         QtCore.QMetaObject.connectSlotsByName(ManipulatorWindow)
92
93     def retranslateUi(self, ManipulatorWindow):
94         ManipulatorWindow.setWindowTitle(QtGui.QApplication.translate("ManipulatorWindow", ...
95             "MainWindow", None, QtGui.QApplication.UnicodeUTF8))
96         self.customizingRobotButton.setText(QtGui.QApplication.translate("ManipulatorWindow", ...
97             "Parametrizar Robot", None, QtGui.QApplication.UnicodeUTF8))
98         self.textBrowser.setHtml(QtGui.QApplication.translate("ManipulatorWindow", ...
99             "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" ...
100            \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
101            "<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
102            "p, li { white-space: pre-wrap; }\n"
103            "</style></head><body style=\" font-family:'Ubuntu'; font-size:11pt; ...
104            font-weight:400; font-style:normal;\">\n"
105            "<p align=\"justify\" style=\" margin-top:0px; margin-bottom:0px; ...
106            margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px; \">* ...
107            El espacio de trabajo de un robot seria todo los puntos a los que el ...
108            extremo (efector final) del robot puede llegar. Imagina una esfera en cuyo ...
109            centro esta el robot, si esa esfera contiene los puntos a los que el robot ...
110            debe llegar, dinos el radio de esa esfera en metros.</p></body></html>", ...
111            None, QtGui.QApplication.UnicodeUTF8))
```

```

101         self.textBrowser_2.setHtml(QtGui.QApplication.translate("ManipulatorWindow",
102             "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" ...
103             \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
104             "<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
105             "p, li { white-space: pre-wrap; }\n"
106             "</style></head><body style=\" font-family:'Ubuntu'; font-size:11pt; ...
107             font-weight:400; font-style:normal;\">\n"
108             "<p align=\"justify\" style=\" margin-top:0px; margin-bottom:0px; ...
109             margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">A ...
110             continuacion, definiremos la forma de tu robot en base a tus preferencias. ...
111             Una vez hayas elegido las especificaciones de tu robot, deberas pulsar el ...
112             boton "Parametrizar Robot" y el proceso de generacion comenzara. ...
113             </p></body></html>", None, QtGui.QApplication.UnicodeUTF8))
114         self.label_2.setText(QtGui.QApplication.translate("ManipulatorWindow", ...
115             "Forma del objeto a mover: ", None, QtGui.QApplication.UnicodeUTF8))
116         self.objectShape.setItemText(0, ...
117             QtGui.QApplication.translate("ManipulatorWindow", "<<Elige tu ...
118             opcion>>", None, QtGui.QApplication.UnicodeUTF8))
119         self.objectShape.setItemText(1, ...
120             QtGui.QApplication.translate("ManipulatorWindow", "Cilindro", None, ...
121             QtGui.QApplication.UnicodeUTF8))
122         self.objectShape.setItemText(2, ...
123             QtGui.QApplication.translate("ManipulatorWindow", "Cono", None, ...
124             QtGui.QApplication.UnicodeUTF8))
125         self.objectShape.setItemText(3, ...
126             QtGui.QApplication.translate("ManipulatorWindow", "Cubo", None, ...
127             QtGui.QApplication.UnicodeUTF8))
128         self.objectShape.setItemText(4, ...
129             QtGui.QApplication.translate("ManipulatorWindow", "Esfera", None, ...
130             QtGui.QApplication.UnicodeUTF8))
131         self.objectShape.setItemText(5, ...
132             QtGui.QApplication.translate("ManipulatorWindow", "Piramide", None, ...
133             QtGui.QApplication.UnicodeUTF8))
134         self.objectShape.setItemText(6, ...
135             QtGui.QApplication.translate("ManipulatorWindow", "Prisma", None, ...
136             QtGui.QApplication.UnicodeUTF8))
137         self.label_3.setText(QtGui.QApplication.translate("ManipulatorWindow", ...
138             "Masa del objeto a levantar (kg): ", None, ...
139             QtGui.QApplication.UnicodeUTF8))
140         self.label_4.setText(QtGui.QApplication.translate("ManipulatorWindow", ...
141             "Espacio de trabajo (cm): ", None, QtGui.QApplication.UnicodeUTF8))
142         self.label_5.setText(QtGui.QApplication.translate("ManipulatorWindow", ...
143             "Quieres que tu robot posicione o posicione y oriente?", None, ...
144             QtGui.QApplication.UnicodeUTF8))
145         self.DOFs.setItemText(0, ...
146             QtGui.QApplication.translate("ManipulatorWindow", "<<Elige tu ...
147             opcion>>", None, QtGui.QApplication.UnicodeUTF8))
148         self.DOFs.setItemText(1, ...
149             QtGui.QApplication.translate("ManipulatorWindow", "Posicionar", ...
150             None, QtGui.QApplication.UnicodeUTF8))

```

```
119         self.DOFs.setItemText(2, ...
120             QtGui.QApplication.translate("ManipulatorWindow", "Posicionar y ...
121             orientar", None, QtGui.QApplication.UnicodeUTF8))
```

#### 4.3.2. Robot móvil

```
1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file 'robot_movil.ui'
4 #
5  # Created: Wed Dec  7 23:49:43 2016
6  #       by: pyside-uic 0.2.15 running on PySide 1.2.4
7 #
8  # WARNING! All changes made in this file will be lost!
9
10 from PySide import QtCore, QtGui
11
12 class Ui_param_robot_movil(object):
13     def setupUi(self, param_robot_movil):
14         param_robot_movil.setObjectName("param_robot_movil")
15         param_robot_movil.resize(370, 308)
16         self.verticalLayout = QtGui.QVBoxLayout(param_robot_movil)
17         self.verticalLayout.setObjectName("verticalLayout")
18         self.horizontalLayout = QtGui.QHBoxLayout()
19         self.horizontalLayout.setObjectName("horizontalLayout")
20         self.label = QtGui.QLabel(param_robot_movil)
21         self.label.setObjectName("label")
22         self.horizontalLayout.addWidget(self.label)
23         self.vel_lin = QtGui.QLineEdit(param_robot_movil)
24         self.vel_lin.setObjectName("vel_lin")
25         self.horizontalLayout.addWidget(self.vel_lin)
26         self.verticalLayout.addLayout(self.horizontalLayout)
27         self.horizontalLayout_2 = QtGui.QHBoxLayout()
28         self.horizontalLayout_2.setObjectName("horizontalLayout_2")
29         self.label_2 = QtGui.QLabel(param_robot_movil)
30         self.label_2.setObjectName("label_2")
31         self.horizontalLayout_2.addWidget(self.label_2)
32         self.vel_ang = QtGui.QLineEdit(param_robot_movil)
33         self.vel_ang.setObjectName("vel_ang")
34         self.horizontalLayout_2.addWidget(self.vel_ang)
35         self.verticalLayout.addLayout(self.horizontalLayout_2)
36         self.horizontalLayout_5 = QtGui.QHBoxLayout()
37         self.horizontalLayout_5.setObjectName("horizontalLayout_5")
38         self.label_5 = QtGui.QLabel(param_robot_movil)
39         self.label_5.setObjectName("label_5")
40         self.horizontalLayout_5.addWidget(self.label_5)
41         self.masa = QtGui.QLineEdit(param_robot_movil)
```

```
42         self.masa.setObjectName("masa")
43         self.horizontalLayout_5.addWidget(self.masa)
44         self.verticalLayout.addLayout(self.horizontalLayout_5)
45         self.horizontalLayout_4 = QtGui.QHBoxLayout()
46         self.horizontalLayout_4.setObjectName("horizontalLayout_4")
47         self.label_4 = QtGui.QLabel(param_robot_movil)
48         self.label_4.setObjectName("label_4")
49         self.horizontalLayout_4.addWidget(self.label_4)
50         self.inclinacion = QtGui.QLineEdit(param_robot_movil)
51         self.inclinacion.setObjectName("inclinacion")
52         self.horizontalLayout_4.addWidget(self.inclinacion)
53         self.verticalLayout.addLayout(self.horizontalLayout_4)
54         self.btn_dimensionar = QtGui.QPushButton(param_robot_movil)
55         self.btn_dimensionar.setObjectName("btn_dimensionar")
56         self.verticalLayout.addWidget(self.btn_dimensionar)
57         self.horizontalLayout_3 = QtGui.QHBoxLayout()
58         self.horizontalLayout_3.setObjectName("horizontalLayout_3")
59         self.label_3 = QtGui.QLabel(param_robot_movil)
60         self.label_3.setObjectName("label_3")
61         self.horizontalLayout_3.addWidget(self.label_3)
62         self.resultado = QtGui.QLineEdit(param_robot_movil)
63         self.resultado.setObjectName("resultado")
64         self.horizontalLayout_3.addWidget(self.resultado)
65         self.verticalLayout.addLayout(self.horizontalLayout_3)
66         self.btn_generar = QtGui.QPushButton(param_robot_movil)
67         self.btn_generar.setObjectName("btn_generar")
68         self.verticalLayout.addWidget(self.btn_generar)
69
70         self.retranslateUi(param_robot_movil)
71         QtCore.QMetaObject.connectSlotsByName(param_robot_movil)
72
73     def retranslateUi(self, param_robot_movil):
74         param_robot_movil.setWindowTitle(QtGui.QApplication.translate("param_robot_movil", ...
75             "Form", None, QtGui.QApplication.UnicodeUTF8))
76         self.label.setText(QtGui.QApplication.translate("param_robot_movil", ...
77             "Velocidad lineal max (m/s):", None, QtGui.QApplication.UnicodeUTF8))
78         self.label_2.setText(QtGui.QApplication.translate("param_robot_movil", ...
79             "Velocidad angular max (rad/s):", None, ...
80                 QtGui.QApplication.UnicodeUTF8))
81         self.label_5.setText(QtGui.QApplication.translate("param_robot_movil", ...
82             "Masa a transportar (kg): ", None, QtGui.QApplication.UnicodeUTF8))
83         self.label_4.setText(QtGui.QApplication.translate("param_robot_movil", ...
84             "Maxima inclinacion (): ", None, QtGui.QApplication.UnicodeUTF8))
85         self.btn_dimensionar.setText(QtGui.QApplication.translate("param_robot_movil", ...
86             "Dimensionar", None, QtGui.QApplication.UnicodeUTF8))
87         self.label_3.setText(QtGui.QApplication.translate("param_robot_movil", ...
88             "Resultado:", None, QtGui.QApplication.UnicodeUTF8))
89         self.btn_generar.setText(QtGui.QApplication.translate("param_robot_movil", ...
90             "Generar ruedas", None, QtGui.QApplication.UnicodeUTF8))
```

## 4.4. Generación STL

### 4.4.1. Eslabones Manipulador

```
1 # -*- coding: UTF-8 -*-
2 from __future__ import division
3 import os
4 import sys
5 import re
6
7 from solid import *
8 from solid.utils import *
9
10 import subprocess
11
12 import os
13
14
15
16 def include_part(l_cono):
17     # scad_path = os.path.join(os.path.dirname(__file__), "parte_der.scad")
18     scad_path = os.path.join(os.path.dirname(__file__), "otra_parte.scad")
19     use(scad_path)
20     escala = scale([1,1,1])(stl2scad())
21     # stl2scad es un modulo dentro de la pieza "parte_der.scad"
22     # return translate(v=[0, l_cono, 0])(rotate(a=180, v=[1, 0, 0])(stl2scad()))
23     return translate(v=[0, l_cono, 0])(rotate(a=180, v=[1, 0, 0])(escala()))
24
25 def cilindro_hueco(r1,r2,l):
26     c = difference()
27         cylinder(r=r2, h=l),
28         cylinder(r=r1, h=l*1)
29
30
31     # c = right(-l/2)(rotate(a=90, v=FORWARD_VEC))(c)
32     c = rotate(a=90, v=[1, 0, 0])(c)
33
34     return c
35
36 def crear_cono(r1,r2,l):
37     c =cylinder(r1=r1,r2=r2,h=l)
38     c = rotate(a=90, v=[-1, 0, 0])(c)
39     return c
40
41
42 def fillet(r_menor, r_mayor,r_fillet_ext, r_fillet_int):
43     lado_largo_cuadrado_principal = r_mayor+r_fillet_ext
44     radio_circulo_principal = r_fillet_ext
45     expesor_exterior = r_mayor-r_menor
```

```

46     radio_redondeo_peq = r_fillet_int
47     lado_corto_cuadrado_peq1 = r_fillet_ext-r_fillet_int
48
49
50     cuad1 = translate([radio_circulo_principal+expesor_exterior, 0, 0])\
51         (square([lado_largo_cuadrado_principal-radio_circulo_principal+expesor_exterior, ...
52             lado_corto_cuadrado_peq1]))
53     cir1= ...
54         translate([radio_circulo_principal+expesor_exterior+radio_redondeo_peq, ...
55             lado_corto_cuadrado_peq1, 0])\
56         (circle(radio_redondeo_peq))
57     cuad2 = ...
58         translate([radio_circulo_principal+expesor_exterior+radio_redondeo_peq, ...
59             lado_corto_cuadrado_peq1, 0])\
60         (square([lado_largo_cuadrado_principal-radio_circulo_principal+expesor_exterior, ...
61             radio_redondeo_peq]))
62
63     parte_neg = cuad1+cir1+cuad2
64
65     parte_neg2 = square([lado_largo_cuadrado_principal, ...
66         radio_circulo_principal])-circle(radio_circulo_principal)
67
68     parte_fin = parte_neg2-parte_neg
69
70
71 def ...
72     parametrizar_viga(longitud_total,radio_externo,radio_interno,r_fillet_ext,r_fillet_int,nombre):
73     SEGMENTS = 48
74
75     # longitud_total = 60
76     # radio_interno = 8
77     # radio_externo = 10.5
78
79     out_dir = sys.argv[1] if len(sys.argv) > 1 else os.curdir
80     file_out = os.path.join(out_dir, 'final_link_fillet_.scad')
81
82     r2=radio_externo
83     r1=radio_interno
84     # r_fillet_ext = 2
85     l=longitud_total-10-2*r_fillet_ext
86     # r_fillet_int = 1
87     print("Radio redondeo interior: {}".format(r_fillet_int))
88
89     a = cilindro_hueco(r1,r2,l)

```

```

89     b = include_part(r_fillet_ext)
90     c= translate(v=[0,-1,0])(mirror(v=[0, 1, 0])(b))
91     a= a + b + c
92     #
93
94     cono1 = fillet(r1, r2,r_fillet_ext, r_fillet_int)
95     cono2 = translate(v=[0, -1, 0])(mirror(v=[0, 1, ...
96         0])(fillet(r1,r2,r_fillet_ext,r_fillet_int)))
97
98     a = a+ cono1 + cono2
99     # a = a + cono1 + cono2
100
101
102     # print("%(__file__)s: SCAD file written to: \n%(file_out)s" % vars())
103
104
105
106     # bashCommand = '''openscad -o {a}/hoals.stl -D 'quality="production"' ...
107         # {a}/final_link_fillet_.scad'''.format(a=os.path.dirname(__file__))
108     bashCommand = '''openscad -o {a}/archivos_generados/eslabon_{nombre}.stl ...
109         # {a}/final_link_fillet_.scad'''.format(a=os.path.dirname(__file__), nombre=nombre)
110
111     os.system(bashCommand)

```

#### 4.4.2. Ruedas robot móvil

```

1  # -*- coding: UTF-8 -*-
2  from __future__ import division
3
4  from PyKDL import Segment
5
6  from solid import *
7  from solid.utils import *
8
9  import os
10
11 SEGMENTS = 48
12
13 longitud_total = 60
14 radio_interno = 8
15 radio_externo = 10.5
16
17 def rueda(radio_rueda):
18     xaxis = [1, 0, 0]
19     yaxis = [0, 1, 0]

```



```

20 zaxis = [0, 0, 1]
21
22 radio_rueda = radio_rueda
23 radio_in = radio_rueda*0.8
24 radio_enganche = 15#radio_rueda*0.4
25 radio_servo = 3
26
27 # ancho del diente
28 lt = 4
29 # altura del diente
30 h = 7
31 # radio interno
32 r = radio_rueda
33 # numero de dientes
34 n = 3
35
36 resta = translate([0, 0, h/2])(cylinder(h=h,r = radio_enganche,center = ...  
True,segments=50))\  
+cylinder(h=h,r = radio_servo,center = True,segments=50)
37 resta2 = cylinder(h = 3,r = 1.1*radio_rueda,center = True,segments=50)-\  
cylinder(h = 3,r = 0.95*radio_rueda,center = True,segments=50)
38
39 return linear_extrude(height=h,center=True) ((circle(r = ...  
radio_rueda)-circle(r = radio_in))  
+circle(r = 1.18*radio_enganche)+wheel(n,radio_in,lt))-resta-resta2
40
41
42
43
44 def wheel(n,radio_in,lt):
45     sal = circle(r=0)
46     for i in range(0,n ):
47         sal +=rotate(i / n * ...  
360)(polygon([[radio_in,lt],[radio_in,-lt],[-lt,-lt],[-lt,lt]]))
48     return sal
49
50
51 def generar_rueda(radio_rueda):
52     out_dir = sys.argv[1] if len(sys.argv) > 1 else os.curdir
53     file_out = os.path.join(out_dir, 'rueda.scad')
54
55     a = rueda(radio_rueda)
56
57
58     scad_render_to_file(a, file_out, file_header='$fn = %s;' % SEGMENTS)
59
60     bashCommand = '''openscad -o {a}/archivos_generados/rueda.stl ...  
{a}/rueda.scad''' .format(a=os.path.dirname(__file__))
61
62     os.system(bashCommand)

```

#### 4.4.3. Xacro Robot móvil

```
1  <?xml version="1.0"?>
2  <robot name="labrob" xmlns:xacro="http://www.ros.org/wiki/xacro">
3
4  <!-- Wheels -->
5  <xacro:property name="wheel_radius" value="$(arg radio_rueda)" />
6  <xacro:property name="wheel_height" value="0.1" />
7  <xacro:property name="wheel_mass" value="2.5" /> <!-- in kg-->
8
9  <xacro:property name="base_x_origin_to_wheel_origin" value="$(arg batalla)" />
10 <xacro:property name="base_y_origin_to_wheel_origin" value="$(arg ...
     ancho_vias)" />
11 <xacro:property name="base_z_origin_to_wheel_origin" value="0.0" />
12
13 <!-- Included URDF/XACRO Files -->
14
15 <xacro:macro name="cylinder_inertia" params="m r h">
16   <inertia ixx="${m*(3*r*r+h*h)/12}" ixy = "0" ixz = "0"
17     iyy="${m*(3*r*r+h*h)/12}" iyz = "0"
18     izz="${m*r*r/2}" />
19 </xacro:macro>
20
21 <xacro:macro name="wheel" params="fb lr parent translateX translateY flipY"> ...
22   <!--fb : front, back ; lr: left, right -->
23   <link name="${fb}_${lr}_wheel">
24     <visual>
25       <origin xyz="0 0 0" rpy="${flipY*M_PI/2} 0 0" />
26       <geometry>
27         <cylinder length="${wheel_height}" radius="${wheel_radius}" />
28       </geometry>
29       <material name="DarkGray" />
30     </visual>
31     <collision>
32       <origin xyz="0 0 0" rpy="${flipY*M_PI/2} 0 0" />
33       <geometry>
34         <cylinder length="${wheel_height}" radius="${wheel_radius}" />
35       </geometry>
36     </collision>
37     <inertial>
38       <mass value="${wheel_mass}" />
39       <origin xyz="0 0 0" />
40       <cylinder_inertia m="${wheel_mass}" r="${wheel_radius}" ...
           h="${wheel_height}" />
41     </inertial>
42   </link>
43
44   <gazebo reference="${fb}_${lr}_wheel">
45     <mul value="1.0"/>
```

```

45      <mu2 value="1.0"/>
46      <kp value="10000000.0" />
47      <kd value="1.0" />
48      <fdirl value="1 0 0"/>
49      <material>Gazebo/Grey</material>
50      <turnGravityOff>false</turnGravityOff>
51  </gazebo>
52
53  <joint name="${fb}_${lr}_wheel_joint" type="continuous">
54      <parent link="${parent}"/>
55      <child link="${fb}_${lr}_wheel"/>
56      <origin xyz="${translateX * base_x_origin_to_wheel_origin} ${translateY * ...
           base_y_origin_to_wheel_origin} ${base_z_origin_to_wheel_origin}" ...
           rpy="0 0 0" />
57      <axis xyz="0 1 0" rpy="0 0 0" />
58      <limit effort="100" velocity="100"/>
59      <joint_properties damping="0.0" friction="0.0"/>
60  </joint>
61
62  <!-- Transmission is important to link the joints and the controller -->
63  <transmission name="${fb}_${lr}_wheel_joint_trans">
64      <type>transmission_interface/SimpleTransmission</type>
65      <joint name="${fb}_${lr}_wheel_joint">
66          <hardwareInterface>EffortJointInterface</hardwareInterface>
67      </joint >
68      <actuator name="${fb}_${lr}_wheel_joint_motor">
69          <hardwareInterface>EffortJointInterface</hardwareInterface>
70          <mechanicalReduction>1</mechanicalReduction>
71      </actuator>
72  </transmission>
73
74  </xacro:macro>
75
76  <!-- PROPERTY LIST -->
77  <!--All units in m-kg-s-radians unit system -->
78  <xacro:property name="M_PI" value="3.1415926535897931" />
79
80  <!-- Main Body-base -->
81  <xacro:property name="base_x_size" ...
           value="${base_x_origin_to_wheel_origin*1.1}" />
82  <xacro:property name="base_y_size" ...
           value="${base_y_origin_to_wheel_origin*1.1}" />
83  <xacro:property name="base_z_size" value="0.25" />
84  <xacro:property name="base_mass" value="35" /> <!-- in kg-->
85
86  <!--Inertial macros for the box and cylinder. Units are kg*m^2-->
87  <xacro:macro name="box_inertia" params="m x y z">
88      <inertia ixx="${m*(y*y+z*z)/12}" ixy = "0" ixz = "0"
89          iyy="${m*(x*x+z*z)/12}" iyz = "0"
90          izz="${m*(x*x+z*z)/12}" />
91  </xacro:macro>

```

```
92      <!-- BASE-FOOTPRINT -->
93      <!-- base_footprint is a fictitious link(frame) that is on the ground right ...
94          below base_link origin -->
95      <link name="base_footprint">
96          <inertial>
97              <mass value="0.0001" />
98              <origin xyz="0 0 0" />
99              <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
100                 iyy="0.0001" iyz="0.0"
101                 izz="0.0001" />
102          </inertial>
103          <visual>
104              <origin xyz="0 0 0" rpy="0 0 0" />
105              <geometry>
106                  <box size="0.001 0.001 0.001" />
107              </geometry>
108          </visual>
109      </link>
110
111      <gazebo reference="base_footprint">
112          <turnGravityOff>false</turnGravityOff>
113      </gazebo>
114
115      <joint name="base_footprint_joint" type="fixed">
116          <origin xyz="0 0 ${wheel_radius - base_z_origin_to_wheel_origin}" rpy="0 0 0" />
117          <parent link="base_footprint"/>
118          <child link="base_link" />
119      </joint>
120
121      <!-- BASE-LINK -->
122      <!--Actual body/chassis of the robot-->
123      <link name="base_link">
124          <inertial>
125              <mass value="${base_mass}" />
126              <origin xyz="0 0 0" />
127              <!--The 3x3 rotational inertia matrix. -->
128              <box_inertia m="${base_mass}" x="${base_x_size}" y="${base_y_size}" ...
129                 z="${base_z_size}"/>
130          </inertial>
131          <visual>
132              <origin xyz="0 0 0" rpy="0 0 0" />
133              <geometry>
134                  <box size="1 0.5 0.25"/>
135              </geometry>
136              <material name="Yellow" />
137          </visual>
138          <collision>
139              <origin xyz="0 0 0" rpy="0 0 0" />
140              <geometry>
141                  <box size="1 0.5 0.25"/>
```

```

141      </geometry>
142      </collision>
143  </link>
144  <gazebo reference="base_link">
145    <material>Gazebo/Green</material>
146    <turnGravityOff>false</turnGravityOff>
147  </gazebo>
148
149  <!-- WHEELS -->
150  <wheel fb="front" lr="right" parent="base_link" translateX="1" ...
151    translateY="-1" flipY="-1"/>
152  <wheel fb="front" lr="left" parent="base_link" translateX="1" translateY="1" ...
153    flipY="-1"/>
154  <wheel fb="back" lr="right" parent="base_link" translateX="-1" ...
155    translateY="-1" flipY="-1"/>
156  <wheel fb="back" lr="left" parent="base_link" translateX="-1" translateY="1" ...
157    flipY="-1"/>
158
159  <gazebo>
160    <plugin name="skid_steer_drive_controller" ...
161      filename="libgazebo_ros_skid_steer_drive.so">
162      <alwaysOn>true</alwaysOn>
163      <updateRate>100.0</updateRate>
164      <leftFrontJoint>front_left_wheel_joint</leftFrontJoint>
165      <rightFrontJoint>front_right_wheel_joint</rightFrontJoint>
166      <leftRearJoint>back_left_wheel_joint</leftRearJoint>
167      <rightRearJoint>back_right_wheel_joint</rightRearJoint>
168      <wheelSeparation>${base_y_size}</wheelSeparation>
169      <wheelDiameter>${2*wheel_radius}</wheelDiameter>
170      <torque>35</torque>
171      <broadcastTF>1</broadcastTF>
172      <odometryFrame>map</odometryFrame>
173      <commandTopic>cmd_vel</commandTopic>
174      <odometryTopic>odom</odometryTopic>
175      <robotBaseFrame>base_footprint</robotBaseFrame>
176    </plugin>
177    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so" />
178  </gazebo>
179
180  <!-- NUEVO -->
181
182  <xacro:property name="camera_link" value="0.05" /> <!-- Size of square ...
183  'camera' box -->
184  ...
185  <!-- Camera -->
186  <joint name="camera_joint" type="fixed">
187    <axis xyz="0 1 0" />
188    <origin xyz="${base_x_size/2-camera_link/2} 0 ${base_z_size/2 + ...
189      camera_link/2}" rpy="0 0 0"/>
190    <parent link="base_link"/>
191    <child link="camera_link"/>

```

```

185      </joint>
186
187      <link name="camera_link">
188          <collision>
189              <origin xyz="0 0 0" rpy="0 0 0"/>
190              <geometry>
191                  <box size="${camera_link} ${camera_link} ${camera_link}" />
192              </geometry>
193          </collision>
194
195          <visual>
196              <origin xyz="0 0 0" rpy="0 0 0"/>
197              <geometry>
198                  <box size="${camera_link} ${camera_link} ${camera_link}" />
199              </geometry>
200              <material name="black"/>
201          </visual>
202
203          <inertial>
204              <mass value="1e-5" />
205              <origin xyz="0 0 0" rpy="0 0 0"/>
206              <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
207          </inertial>
208      </link>
209
210  <!--gazebo reference="camera_link"-->
211      <sensor type="camera" name="camer1">
212          <update_rate>30.0</update_rate>
213          <camera name="head">
214              <horizontal_fov>1.3962634</horizontal_fov>
215              <image>
216                  <width>800</width>
217                  <height>800</height>
218                  <format>R8G8B8</format>
219              </image>
220              <clip>
221                  <near>0.02</near>
222                  <far>300</far>
223              </clip>
224              <noise>
225                  <type>gaussian</type>
226                  Noise is sampled independently per pixel on each frame.
227                  That pixel's noise value is added to each of its color
228                  channels, which at that point lie in the range [0,1].
229                  <mean>0.0</mean>
230                  <stddev>0.007</stddev>
231              </noise>
232          </camera>
233          <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
234              <alwaysOn>true</alwaysOn>
235              <updateRate>0.0</updateRate>

```

```
236      <cameraName>labrob/camera</cameraName>
237      <imageTopicName>image_raw</imageTopicName>
238      <cameraInfoTopicName>camera_info</cameraInfoTopicName>
239      <frameName>camera_link</frameName>
240      <hackBaseline>0.07</hackBaseline>
241      <distortionK1>0.0</distortionK1>
242      <distortionK2>0.0</distortionK2>
243      <distortionK3>0.0</distortionK3>
244      <distortionT1>0.0</distortionT1>
245      <distortionT2>0.0</distortionT2>
246    </plugin>
247  </sensor>
248 </gazebo-->
249
250 </robot>
```

## 4.5. Parametrización URDF

### 4.5.1. Parametrizar Hexápodo

```
1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file 'hex.ui'
4 #
5 # Created: Wed Dec 14 21:22:39 2016
6 #           by: pyside-uic 0.2.15 running on PySide 1.2.4
7 #
8 # WARNING! All changes made in this file will be lost!
9
10 from PySide import QtCore, QtGui
11
12 class Hexapodo(object):
13     def setupUi(self, Form):
14         Form.setObjectName("Form")
15         Form.resize(400, 301)
16         self.gridLayout_11 = QtGui.QGridLayout(Form)
17         self.gridLayout_11.setObjectName("gridLayout_11")
18         self.btn_salir = QtGui.QCommandLinkButton(Form)
19         self.btn_salir.setObjectName("btn_salir")
20         self.gridLayout_11.addWidget(self.btn_salir, 1, 0, 1, 1)
21         self.btn_generar = QtGui.QPushButton(Form)
22         self.btn_generar.setObjectName("btn_generar")
23         self.gridLayout_11.addWidget(self.btn_generar, 1, 1, 1, 1)
24         self.tab = QtGui.QStackedWidget(Form)
25         self.tab.setObjectName("tab")
26         self.tabPage1 = QtGui.QWidget()
```

```
27         self.tabPage1.setObjectName("tabPage1")
28         self.gridLayout_4 = QtGui.QGridLayout(self.tabPage1)
29         self.gridLayout_4.setObjectName("gridLayout_4")
30         self.label_24 = QtGui.QLabel(self.tabPage1)
31         self.label_24.setObjectName("label_24")
32         self.gridLayout_4.addWidget(self.label_24, 2, 0, 1, 1)
33         self.ln_hex_long_x_cuerpo = QtGui.QLineEdit(self.tabPage1)
34         self.ln_hex_long_x_cuerpo.setObjectName("ln_hex_long_x_cuerpo")
35         self.gridLayout_4.addWidget(self.ln_hex_long_x_cuerpo, 2, 2, 1, 1)
36         self.label_12 = QtGui.QLabel(self.tabPage1)
37         self.label_12.setObjectName("label_12")
38         self.gridLayout_4.addWidget(self.label_12, 6, 0, 1, 1)
39         self.ln_hex_long_tibia = QtGui.QLineEdit(self.tabPage1)
40         self.ln_hex_long_tibia.setObjectName("ln_hex_long_tibia")
41         self.gridLayout_4.addWidget(self.ln_hex_long_tibia, 6, 2, 1, 1)
42         self.ln_hex_n_servos = QtGui.QLineEdit(self.tabPage1)
43         self.ln_hex_n_servos.setObjectName("ln_hex_n_servos")
44         self.gridLayout_4.addWidget(self.ln_hex_n_servos, 1, 2, 1, 1)
45         self.label_13 = QtGui.QLabel(self.tabPage1)
46         self.label_13.setObjectName("label_13")
47         self.gridLayout_4.addWidget(self.label_13, 5, 0, 1, 1)
48         self.label_11 = QtGui.QLabel(self.tabPage1)
49         self.label_11.setObjectName("label_11")
50         self.gridLayout_4.addWidget(self.label_11, 4, 0, 1, 1)
51         self.ln_hex_long_femur = QtGui.QLineEdit(self.tabPage1)
52         self.ln_hex_long_femur.setObjectName("ln_hex_long_femur")
53         self.gridLayout_4.addWidget(self.ln_hex_long_femur, 5, 2, 1, 1)
54         self.label_9 = QtGui.QLabel(self.tabPage1)
55         self.label_9.setObjectName("label_9")
56         self.gridLayout_4.addWidget(self.label_9, 1, 0, 1, 1)
57         self.ln_hex_long_coxa = QtGui.QLineEdit(self.tabPage1)
58         self.ln_hex_long_coxa.setObjectName("ln_hex_long_coxa")
59         self.gridLayout_4.addWidget(self.ln_hex_long_coxa, 4, 2, 1, 1)
60         self.label_25 = QtGui.QLabel(self.tabPage1)
61         self.label_25.setObjectName("label_25")
62         self.gridLayout_4.addWidget(self.label_25, 3, 0, 1, 1)
63         self.ln_hex_lon_y_cuerpo = QtGui.QLineEdit(self.tabPage1)
64         self.ln_hex_lon_y_cuerpo.setObjectName("ln_hex_lon_y_cuerpo")
65         self.gridLayout_4.addWidget(self.ln_hex_lon_y_cuerpo, 3, 2, 1, 1)
66         self.tab.addWidget(self.tabPage1)
67         self.tabPage2 = QtGui.QWidget()
68         self.tabPage2.setObjectName("tabPage2")
69         self.gridLayout_12 = QtGui.QGridLayout(self.tabPage2)
70         self.gridLayout_12.setObjectName("gridLayout_12")
71         self.label_78 = QtGui.QLabel(self.tabPage2)
72         self.label_78.setMaximumSize(QtCore.QSize(16777215, 20))
73         self.label_78.setObjectName("label_78")
74         self.gridLayout_12.addWidget(self.label_78, 0, 2, 1, 1)
75         self.label_43 = QtGui.QLabel(self.tabPage2)
76         self.label_43.setObjectName("label_43")
77         self.gridLayout_12.addWidget(self.label_43, 3, 0, 1, 1)
```

```

78         self.label_44 = QtGui.QLabel(self.tabPage2)
79         self.label_44.setObjectName("label_44")
80         self.gridLayout_12.addWidget(self.label_44, 4, 0, 1, 1)
81         self.ln_movil_batalla = QtGui.QLineEdit(self.tabPage2)
82         self.ln_movil_batalla.setObjectName("ln_movil_batalla")
83         self.gridLayout_12.addWidget(self.ln_movil_batalla, 3, 2, 1, 1)
84         self.ln_movil_radio_rue = QtGui.QLineEdit(self.tabPage2)
85         self.ln_movil_radio_rue.setObjectName("ln_movil_radio_rue")
86         self.gridLayout_12.addWidget(self.ln_movil_radio_rue, 2, 2, 1, 1)
87         self.label_60 = QtGui.QLabel(self.tabPage2)
88         self.label_60.setObjectName("label_60")
89         self.gridLayout_12.addWidget(self.label_60, 1, 0, 1, 1)
90         self.ln_movil_ancho_vias = QtGui.QLineEdit(self.tabPage2)
91         self.ln_movil_ancho_vias.setObjectName("ln_movil_ancho_vias")
92         self.gridLayout_12.addWidget(self.ln_movil_ancho_vias, 4, 2, 1, 1)
93         self.label_10 = QtGui.QLabel(self.tabPage2)
94         self.label_10.setObjectName("label_10")
95         self.gridLayout_12.addWidget(self.label_10, 2, 0, 1, 1)
96         self.ln_movil_tipo = QtGui.QLineEdit(self.tabPage2)
97         self.ln_movil_tipo.setObjectName("ln_movil_tipo")
98         self.gridLayout_12.addWidget(self.ln_movil_tipo, 1, 2, 1, 1)
99         self.tab.addWidget(self.tabPage2)
100        self.gridLayout_11.addWidget(self.tab, 0, 0, 1, 2)
101
102        self.retranslateUi(Form)
103        QtCore.QMetaObject.connectSlotsByName(Form)
104
105    def retranslateUi(self, Form):
106        Form.setWindowTitle(QtGui.QApplication.translate("Form", "Form", None, ...
107            QtGui.QApplication.UnicodeUTF8))
108        self.btn_salir.setText(QtGui.QApplication.translate("Form", "Salir", ...
109            None, QtGui.QApplication.UnicodeUTF8))
110        self.btn_generar.setText(QtGui.QApplication.translate("Form", ...
111            "Generar", None, QtGui.QApplication.UnicodeUTF8))
112        self.label_24.setText(QtGui.QApplication.translate("Form", "long x ...
113            cuerpo", None, QtGui.QApplication.UnicodeUTF8))
114        self.label_12.setText(QtGui.QApplication.translate("Form", "long ...
115            tibia", None, QtGui.QApplication.UnicodeUTF8))
116        self.label_13.setText(QtGui.QApplication.translate("Form", "long ...
117            femur", None, QtGui.QApplication.UnicodeUTF8))
118        self.label_11.setText(QtGui.QApplication.translate("Form", "long coxa", ...
119            None, QtGui.QApplication.UnicodeUTF8))
120        self.label_25.setText(QtGui.QApplication.translate("Form", "long y ...
121            cierpo", None, QtGui.QApplication.UnicodeUTF8))

```

## 4.6. Xacro Hexápodo

```

1  <?xml version="1.0"?>
2  <robot xmlns:xacro="http://ros.org/wiki/xacro" name="hexapodo">
3
4
5      <xacro:property name="long_cuerpo_x" value="$(arg long_x_cuerpo)" />
6      <xacro:property name="long_cuerpo_y" value="$(arg long_y_cuerpo)" />
7      <xacro:property name="long_cuerpo_z" value="0.01" />
8      <xacro:property name="long_coxa" value="$(arg long_coxa)" />
9      <xacro:property name="long_femur" value="$(arg long_femur)" />
10     <xacro:property name="long_tibia" value="$(arg long_tibia)" />
11     <xacro:property name="long_tarsus" value="0.12065" />
12
13     <xacro:property name="masa_base" value="0.8" />
14     <xacro:property name="masa_tarsus" value="0.15" />
15     <xacro:property name="masa_tibia" value="0.3" />
16     <xacro:property name="masa_femur" value="0.3" />
17     <xacro:property name="masa_coxa" value="0.15" />
18
19     <xacro:macro name="box_inertia" params="m x y z">
20         <inertia ixx="${500*m*(y*y+z*z)/12}" ixy = "0" ixz = "0"
21             iyy="${500*m*(x*x+z*z)/12}" iyz = "0"
22             izz="${500*m*(x*x+z*z)/12}" />
23     </xacro:macro>
24
25     <!-- Propiedades articulaciones -->
26     <xacro:property name="lim_joint_inferior" value="-${5.14}" />
27     <xacro:property name="lim_joint_superior" value="${5.14}" />
28     <xacro:property name="max_esfu" value="100000" />
29     <xacro:property name="max_vel" value="1000" />
30
31
32
33     <!-- Chasis -->
34     <link name="map" />
35
36     <joint name="base_joint" type="fixed">
37         <parent link="map" />
38         <child link="base_link" />
39         <origin xyz="0 0 0" rpy="0 0 0" />
40     </joint>
41     <link name="base_link">
42         <inertial>
43             <origin xyz="0 0 0" rpy="0 0 0" />
44             <mass value="${masa_base}" />
45             <box_inertia m="${masa_base}" x="${long_cuerpo_x}" ...
46                 y="${long_cuerpo_y}" z="${long_cuerpo_z}"/>
47         </inertial>
48         <visual>
49             <origin xyz="0 0 0" rpy="0 0 0" />

```

```

50             <box size="${1.2*long_cuerpo_x} ${long_cuerpo_y} ...
51                         ${long_cuerpo_z}" />
52         </geometry>
53         <material name="">
54             <color rgba="0.7 0.7 0.7 1" />
55         </material>
56     </visual>
57     <collision>
58         <origin     xyz="0 0 0" rpy="0 0 0" />
59         <geometry>
60             <box size="${long_cuerpo_x} ${long_cuerpo_y} 0.004"/>
61         </geometry>
62     </collision>
63 </link>
64
65     <!-- Leg macro -->
66     <xacro:macro name="leg" params="side position x y angle axis">
67
68         <!-- leg position -->
69         <joint name="leg_center_joint_${side}${position}" type="fixed">
70             <origin xyz="${x} ${y} 0" rpy="0 0 0" />
71             <parent link="base_link" />
72             <child link="leg_center_${side}${position}" />
73         </joint>
74
75         <link name="leg_center_${side}${position}">
76             <inertial>
77                 <origin     xyz="0 0 0" rpy="0 0 0" />
78                 <mass value="${masa_base}" />
79                 <box_inertia m="${masa_base}" x="${long_cuerpo_x}" ...
80                               y="${long_cuerpo_y}" z="${long_cuerpo_z}" />
81             </inertial>
82         </link>
83
84         <!-- coxa -->
85         <joint name="coxa_joint_${side}${position}" type="revolute">
86             <origin xyz="0 0 0" rpy="0 0 ${angle}" />
87             <parent link="leg_center_${side}${position}" />
88             <child link="coxa_${side}${position}" />
89             <axis xyz="0 0 ${axis}" />
90             <limit lower="${lim_joint_inferior}" upper="${lim_joint_superior}" ...
91                           effort="${max_esfu}" velocity="${max_vel}" />
92         </joint>
93
94         <link name="coxa_${side}${position}">
95             <inertial>
96                 <origin xyz="0 0 0" rpy="0 0 0" />

```

```

97      <visual>
98          <origin xyz="${long_coxa/2} 0 0" rpy="0 0 0" />
99          <geometry>
100             <box size="${long_coxa} 0.022 0.0325" />
101         </geometry>
102         <material name="">
103             <color rgba="0.7 0.7 0.7 1" />
104         </material>
105     </visual>
106     <collision>
107         <origin xyz="${long_coxa/2} 0 0" rpy="0 0 0" />
108         <geometry>
109             <box size="${long_coxa} 0.022 0.0325" />
110         </geometry>
111     </collision>
112 </link>
113
114     <!-- femur -->
115     <joint name="femur_joint_${side}${position}" type="revolute">
116         <origin xyz="${long_coxa} 0 0" rpy="-${pi/2} +0.63 0" />
117         <parent link="coxa_${side}${position}" />
118         <child link="femur_${side}${position}" />
119         <axis xyz="0 0 ${axis}" />
120         <limit lower="${lim_joint_inferior}" upper="${lim_joint_superior}" ...
121             effort="${max_esfu}" velocity="${max_vel}" />
122     </joint>
123
124     <link name="femur_${side}${position}">
125         <inertial>
126             <origin xyz="0 -${long_femur/2} 0" rpy="0 0 0" />
127             <mass value="${masa_femur}" />
128             <box_inertia m="${masa_femur}" x="0.022" y="${long_femur}" ...
129                 z="0.0325"/>
130         </inertial>
131         <visual>
132             <origin xyz="0 -${long_femur/2} 0" rpy="0 0 0" />
133             <geometry>
134                 <box size="0.022 ${long_femur} 0.0325" />
135             </geometry>
136             <material name="">
137                 <color rgba="0.7 0.7 0.7 1" />
138             </material>
139         </visual>
140         <collision>
141             <origin xyz="0 -${long_femur/2} 0" rpy="0 0 0" />
142             <geometry>
143                 <box size="0.022 ${long_femur} 0.0325" />
144             </geometry>
145     </link>

```

```

146      <!-- tibia -->
147      <joint name="tibia_joint_${side}${position}" type="revolute">
148          <origin xyz="0 -${long_femur} 0" rpy="${pi} 0 0.63" />
149          <parent link="femur_${side}${position}" />
150          <child link="tibia_${side}${position}" />
151          <axis xyz="0 0 ${axis}" />
152          <limit lower="${lim_joint_inferior}" upper="${lim_joint_superior}" ...
153              effort="${max_esfu}" velocity="${max_vel}" />
154      </joint>
155
156      <link name="tibia_${side}${position}">
157          <inertial>
158              <origin xyz="${long_tibia}/2 0 0" rpy="0 0 0" />
159              <mass value="${masa_tibia}" />
160              <box_inertia m="${masa_tibia}" x="${long_tibia}" y="0.022" ...
161                  z="0.0325"/>
162          </inertial>
163          <visual>
164              <origin xyz="${long_tibia}/2 0 0" rpy="0 0 0" />
165              <geometry>
166                  <box size="${long_tibia} 0.022 0.0325" />
167              </geometry>
168              <material name="">
169                  <color rgba="0.7 0.7 0.7 1" />
170              </material>
171          </visual>
172          <visual>
173              <origin xyz="${long_tibia} 0 0" rpy="0 0 0" />
174              <geometry>
175                  <sphere radius="0.015"/>
176              </geometry>
177              <material name="">
178                  <color rgba="0.7 0.7 0.7 1" />
179              </material>
180          </visual>
181          <collision>
182              <origin xyz="${long_tibia}/2 0 0" rpy="0 0 0" />
183              <geometry>
184                  <box size="${long_tibia} 0.022 0.0325" />
185              </geometry>
186              <origin xyz="${long_tibia} 0 0" rpy="0 0 0" />
187              <geometry>
188                  <sphere radius="0.015"/>
189              </geometry>
190          </collision>
191      </link>
192
193  </xacro:macro>
194

```

```

195
196
197 <!-- Modelo del robot -->
198   <xacro:leg side="R" position="R" x="0.1778" y="-0.08335" angle="-${pi/2}" ...
199     axis="1" />
200   <xacro:leg side="R" position="M" x="0.0" y="-0.08335" angle="-${pi/2}" ...
201     axis="1" />
202   <xacro:leg side="R" position="F" x="-0.1778" y="-0.08335" angle="-${pi/2}" ...
203     axis="1" />
204   <xacro:leg side="L" position="R" x="0.1778" y="0.08335" angle="${pi/2}" ...
205     axis="-1" />
206   <xacro:leg side="L" position="M" x="0.0" y="0.08335" angle="${pi/2}" ...
207     axis="-1" />
208   <xacro:leg side="L" position="F" x="-0.1778" y="0.08335" angle="${pi/2}" ...
209     axis="-1" />
210
211 </robot>
```

#### 4.6.1. Parametrizar Humanoide

```

1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file 'humanoide.ui'
4  #
5  # Created: Wed Dec 14 21:22:32 2016
6  #           by: pyside-uic 0.2.15 running on PySide 1.2.4
7  #
8  # WARNING! All changes made in this file will be lost!
9
10 from PySide import QtCore, QtGui
11
12 class Humanoide(object):
13     def setupUi(self, Form):
14         Form.setObjectName("Form")
15         Form.resize(400, 301)
16         self.gridLayout_11 = QtGui.QGridLayout(Form)
17         self.gridLayout_11.setObjectName("gridLayout_11")
18         self.btn_salir = QtGui.QCommandLinkButton(Form)
19         self.btn_salir.setObjectName("btn_salir")
20         self.gridLayout_11.addWidget(self.btn_salir, 1, 0, 1, 1)
21         self.btn_generar = QtGui.QPushButton(Form)
22         self.btn_generar.setObjectName("btn_generar")
23         self.gridLayout_11.addWidget(self.btn_generar, 1, 1, 1, 1)
24         self.tab = QtGui.QStackedWidget(Form)
25         self.tab.setObjectName("tab")
26         self.tabPage1 = QtGui.QWidget()
27         self.tabPage1.setObjectName("tabPage1")
```

```
28         self.gridLayout = QtGui.QGridLayout(self.tabPage1)
29         self.gridLayout.setObjectName("gridLayout")
30         self.label = QtGui.QLabel(self.tabPage1)
31         self.label.setObjectName("label")
32         self.gridLayout.addWidget(self.label, 3, 0, 1, 1)
33         self.label_26 = QtGui.QLabel(self.tabPage1)
34         self.label_26.setObjectName("label_26")
35         self.gridLayout.addWidget(self.label_26, 0, 0, 1, 1)
36         self.label_27 = QtGui.QLabel(self.tabPage1)
37         self.label_27.setObjectName("label_27")
38         self.gridLayout.addWidget(self.label_27, 1, 0, 1, 1)
39         self.label_3 = QtGui.QLabel(self.tabPage1)
40         self.label_3.setObjectName("label_3")
41         self.gridLayout.addWidget(self.label_3, 6, 0, 1, 1)
42         self.ln_bip_anchura_cadera = QtGui.QLineEdit(self.tabPage1)
43         self.ln_bip_anchura_cadera.setObjectName("ln_bip_anchura_cadera")
44         self.gridLayout.addWidget(self.ln_bip_anchura_cadera, 0, 2, 1, 1)
45         self.ln_bip_long_pieza_pasiva = QtGui.QLineEdit(self.tabPage1)
46         self.ln_bip_long_pieza_pasiva.setObjectName("ln_bip_long_pieza_pasiva")
47         self.gridLayout.addWidget(self.ln_bip_long_pieza_pasiva, 3, 2, 1, 1)
48         self.ln_bip_long_servo = QtGui.QLineEdit(self.tabPage1)
49         self.ln_bip_long_servo.setObjectName("ln_bip_long_servo")
50         self.gridLayout.addWidget(self.ln_bip_long_servo, 6, 2, 1, 1)
51         self.ln_bip_radio_servo = QtGui.QLineEdit(self.tabPage1)
52         self.ln_bip_radio_servo.setObjectName("ln_bip_radio_servo")
53         self.gridLayout.addWidget(self.ln_bip_radio_servo, 4, 2, 1, 1)
54         self.label_2 = QtGui.QLabel(self.tabPage1)
55         self.label_2.setObjectName("label_2")
56         self.gridLayout.addWidget(self.label_2, 4, 0, 1, 1)
57         self.ln_bip_long_x_pie = QtGui.QLineEdit(self.tabPage1)
58         self.ln_bip_long_x_pie.setObjectName("ln_bip_long_x_pie")
59         self.gridLayout.addWidget(self.ln_bip_long_x_pie, 1, 2, 1, 1)
60         self.label_42 = QtGui.QLabel(self.tabPage1)
61         self.label_42.setObjectName("label_42")
62         self.gridLayout.addWidget(self.label_42, 2, 0, 1, 1)
63         self.ln_bip_long_y_pie = QtGui.QLineEdit(self.tabPage1)
64         self.ln_bip_long_y_pie.setObjectName("ln_bip_long_y_pie")
65         self.gridLayout.addWidget(self.ln_bip_long_y_pie, 2, 2, 1, 1)
66         self.tab.addWidget(self.tabPage1)
67         self.tabPage2 = QtGui.QWidget()
68         self.tabPage2.setObjectName("tabPage2")
69         self.gridLayout_3 = QtGui.QGridLayout(self.tabPage2)
70         self.gridLayout_3.setObjectName("gridLayout_3")
71         self.ln_ser_n_servos = QtGui.QLineEdit(self.tabPage2)
72         self.ln_ser_n_servos.setObjectName("ln_ser_n_servos")
73         self.gridLayout_3.addWidget(self.ln_ser_n_servos, 0, 1, 1, 1)
74         self.label_7 = QtGui.QLabel(self.tabPage2)
75         self.label_7.setObjectName("label_7")
76         self.gridLayout_3.addWidget(self.label_7, 0, 0, 1, 1)
77         self.tab.addWidget(self.tabPage2)
78         self.tabPage3 = QtGui.QWidget()
```

```
79         self.tabPage3.setObjectName("tabPage3")
80         self.gridLayout_4 = QtGui.QGridLayout(self.tabPage3)
81         self.gridLayout_4.setObjectName("gridLayout_4")
82         self.label_24 = QtGui.QLabel(self.tabPage3)
83         self.label_24.setObjectName("label_24")
84         self.gridLayout_4.addWidget(self.label_24, 2, 0, 1, 1)
85         self.ln_hex_long_x_cuerpo = QtGui.QLineEdit(self.tabPage3)
86         self.ln_hex_long_x_cuerpo.setObjectName("ln_hex_long_x_cuerpo")
87         self.gridLayout_4.addWidget(self.ln_hex_long_x_cuerpo, 2, 2, 1, 1)
88         self.label_12 = QtGui.QLabel(self.tabPage3)
89         self.label_12.setObjectName("label_12")
90         self.gridLayout_4.addWidget(self.label_12, 6, 0, 1, 1)
91         self.ln_hex_long_tibia = QtGui.QLineEdit(self.tabPage3)
92         self.ln_hex_long_tibia.setObjectName("ln_hex_long_tibia")
93         self.gridLayout_4.addWidget(self.ln_hex_long_tibia, 6, 2, 1, 1)
94         self.ln_hex_n_servos = QtGui.QLineEdit(self.tabPage3)
95         self.ln_hex_n_servos.setObjectName("ln_hex_n_servos")
96         self.gridLayout_4.addWidget(self.ln_hex_n_servos, 1, 2, 1, 1)
97         self.label_13 = QtGui.QLabel(self.tabPage3)
98         self.label_13.setObjectName("label_13")
99         self.gridLayout_4.addWidget(self.label_13, 5, 0, 1, 1)
100        self.label_11 = QtGui.QLabel(self.tabPage3)
101        self.label_11.setObjectName("label_11")
102        self.gridLayout_4.addWidget(self.label_11, 4, 0, 1, 1)
103        self.ln_hex_long_femur = QtGui.QLineEdit(self.tabPage3)
104        self.ln_hex_long_femur.setObjectName("ln_hex_long_femur")
105        self.gridLayout_4.addWidget(self.ln_hex_long_femur, 5, 2, 1, 1)
106        self.label_9 = QtGui.QLabel(self.tabPage3)
107        self.label_9.setObjectName("label_9")
108        self.gridLayout_4.addWidget(self.label_9, 1, 0, 1, 1)
109        self.ln_hex_long_coxa = QtGui.QLineEdit(self.tabPage3)
110        self.ln_hex_long_coxa.setObjectName("ln_hex_long_coxa")
111        self.gridLayout_4.addWidget(self.ln_hex_long_coxa, 4, 2, 1, 1)
112        self.label_25 = QtGui.QLabel(self.tabPage3)
113        self.label_25.setObjectName("label_25")
114        self.gridLayout_4.addWidget(self.label_25, 3, 0, 1, 1)
115        self.ln_hex_lon_y_cuerpo = QtGui.QLineEdit(self.tabPage3)
116        self.ln_hex_lon_y_cuerpo.setObjectName("ln_hex_lon_y_cuerpo")
117        self.gridLayout_4.addWidget(self.ln_hex_lon_y_cuerpo, 3, 2, 1, 1)
118        self.tab.addWidget(self.tabPage3)
119        self.tabPage4 = QtGui.QWidget()
120        self.tabPage4.setObjectName("tabPage4")
121        self.gridLayout_12 = QtGui.QGridLayout(self.tabPage4)
122        self.gridLayout_12.setObjectName("gridLayout_12")
123        self.label_78 = QtGui.QLabel(self.tabPage4)
124        self.label_78.setMaximumSize(QtCore.QSize(16777215, 20))
125        self.label_78.setObjectName("label_78")
126        self.gridLayout_12.addWidget(self.label_78, 0, 2, 1, 1)
127        self.label_43 = QtGui.QLabel(self.tabPage4)
128        self.label_43.setObjectName("label_43")
129        self.gridLayout_12.addWidget(self.label_43, 3, 0, 1, 1)
```

```

130         self.label_44 = QtGui.QLabel(self.tabPage4)
131         self.label_44.setObjectName("label_44")
132         self.gridLayout_12.addWidget(self.label_44, 4, 0, 1, 1)
133         self.ln_movil_batalla = QtGui.QLineEdit(self.tabPage4)
134         self.ln_movil_batalla.setObjectName("ln_movil_batalla")
135         self.gridLayout_12.addWidget(self.ln_movil_batalla, 3, 2, 1, 1)
136         self.ln_movil_radio_rue = QtGui.QLineEdit(self.tabPage4)
137         self.ln_movil_radio_rue.setObjectName("ln_movil_radio_rue")
138         self.gridLayout_12.addWidget(self.ln_movil_radio_rue, 2, 2, 1, 1)
139         self.label_60 = QtGui.QLabel(self.tabPage4)
140         self.label_60.setObjectName("label_60")
141         self.gridLayout_12.addWidget(self.label_60, 1, 0, 1, 1)
142         self.ln_movil_ancho_vias = QtGui.QLineEdit(self.tabPage4)
143         self.ln_movil_ancho_vias.setObjectName("ln_movil_ancho_vias")
144         self.gridLayout_12.addWidget(self.ln_movil_ancho_vias, 4, 2, 1, 1)
145         self.label_10 = QtGui.QLabel(self.tabPage4)
146         self.label_10.setObjectName("label_10")
147         self.gridLayout_12.addWidget(self.label_10, 2, 0, 1, 1)
148         self.ln_movil_tipo = QtGui.QLineEdit(self.tabPage4)
149         self.ln_movil_tipo.setObjectName("ln_movil_tipo")
150         self.gridLayout_12.addWidget(self.ln_movil_tipo, 1, 2, 1, 1)
151         self.tab.addWidget(self.tabPage4)
152         self.gridLayout_11.addWidget(self.tab, 0, 0, 1, 2)
153
154     self.retranslateUi(Form)
155     QtCore.QMetaObject.connectSlotsByName(Form)
156
157     def retranslateUi(self, Form):
158         Form.setWindowTitle(QtGui.QApplication.translate("Form", "Form", None, ...
159             QtGui.QApplication.UnicodeUTF8))
160         self.btn_salir.setText(QtGui.QApplication.translate("Form", "Salir", ...
161             None, QtGui.QApplication.UnicodeUTF8))
162         self.btn_generar.setText(QtGui.QApplication.translate("Form", ...
163             "Generar", None, QtGui.QApplication.UnicodeUTF8))
164         self.label.setText(QtGui.QApplication.translate("Form", "long pieza ...
165             pasiva", None, QtGui.QApplication.UnicodeUTF8))
166         self.label_26.setText(QtGui.QApplication.translate("Form", "ancho ...
167             cadera", None, QtGui.QApplication.UnicodeUTF8))
168         self.label_27.setText(QtGui.QApplication.translate("Form", "long x ...
169             pie", None, QtGui.QApplication.UnicodeUTF8))
170         self.label_3.setText(QtGui.QApplication.translate("Form", "longitud ...
171             servo", None, QtGui.QApplication.UnicodeUTF8))
172         self.label_2.setText(QtGui.QApplication.translate("Form", "radio ...
173             servo", None, QtGui.QApplication.UnicodeUTF8))
174         self.label_42.setText(QtGui.QApplication.translate("Form", "long y ...
175             pie", None, QtGui.QApplication.UnicodeUTF8))

```

## 4.7. Xacro Humanoide

```

1 <robot xmlns:xacro="http://ros.org/wiki/xacro" name="piernas">
2
3
4 <xacro:property name="porcentaje" value="1" />
5 <xacro:property name="porcentajelx" value="1" />
6 <xacro:property name="porcentajely" value="1" />
7
8 <xacro:property name="servo_caja_x" value="0.04" />
9 <xacro:property name="servo_caja_y" value="0.04" />
10 <xacro:property name="servo_caja_z" value="0.004" />
11
12 <xacro:property name="alargador_caja_x" value="0.04" />
13 <xacro:property name="alargador_caja_y" value="0.04" />
14
15
16 <xacro:property name="base_y" value="0.04" />
17 <xacro:property name="base_z" value="0.004" />
18 <xacro:property name="base_mass" value="0.35" /> <!-- en kg-->
19
20 <xacro:property name="radio" value="$(arg radio_servo)"/>
21 <xacro:property name="longitud" value="$(arg long_servo)"/>
22 <xacro:property name="alargador_caja_z" value="$(arg long_pieza_pasiva)"/>
23 <xacro:property name="long_x_pie" value="$(arg long_x_pie)"/>
24 <xacro:property name="long_y_pie" value="$(arg long_y_pie)"/>
25 <xacro:property name="base_x" value="$(arg anchura_cadera)"/>
26
27
28 <xacro:property name="masa_cilindro" value="0.5" />
29 <xacro:property name="esfuerzoMax" value="10000000000000000000" />
30 <xacro:property name="velMax" value="1000000000000000000" />
31
32 <xacro:macro name="box_inertia" params="m x y z">
33   <inertia ixx="${50*m*(y*y+z*z)/12}" ixy = "0" ixz = "0"
34     iyy="${50*m*(x*x+z*z)/12}" iyz = "0"
35     izz="${50*m*(x*x+z*z)/12}" />
36 </xacro:macro>
37 <xacro:macro name="cylinder_inertia" params="m r h">
38   <inertia ixx="${50*m*(3*r*r+h*h)/12}" ixy = "0" ixz = "0"
39     iyy="${50*(m*(3*r*r+h*h)/12)" iyz = "0"
40     izz="${50*m*r*r/2}" />
41 </xacro:macro>
42
43 <link name="map"></link>
44
45 <joint name="map" type="fixed">
46   <origin xyz="0 0 0" rpy="0 0 0" />
47   <parent link="map"/>
48   <child link="base_link"/>
49 </joint>
50
51

```

```

52 <link name="base_link">
53   <inertial>
54     <origin xyz="0 0 0" rpy="0 0 0" />
55     <mass value="8" />
56     <box_inertia m="${2*base_mass}" x="${2*porcentaje*base_x}" ...
57       y="${2*base_y}" z="${2*base_z}"/>
58   </inertial>
59   <visual>
60     <origin xyz="0 0 0" rpy="0 0 0" />
61     <geometry>
62       <box size="${porcentaje*base_x} ${base_y} ${base_z}" />
63     </geometry>
64     <material name="">
65       <color rgba="0.5 0.5 0.5 1" />
66     </material>
67   </visual>
68   <collision>
69     <origin xyz="0 0 0" rpy="0 0 0" />
70     <geometry>
71       <box size="${porcentaje*base_x} ${base_y} ${base_z}" />
72     </geometry>
73   </collision>
74 </link>
75
76 <!--<xacro:macro name="pierna" params="side side2">
77   <xacro:servo side="base" side2="box" angulo="-${pi/2}" />
78 </xacro:macro>-->
79
80 <xacro:macro name="servo" params="side side2 x y z orientacion ejex ejey ejez">
81
82 <joint name="${side}_${side2}_joint" type="fixed">
83   <origin xyz="${x} ${y} ${z}" rpy="0 0 0" />
84   <parent link="${side}_link" />
85   <child link="${side2}_link" />
86   <axis xyz="0 0 0" />
87   <limit lower = "-2.5" upper = "2.5" effort="${esfuerzoMax}" ...
88     velocity="${velMax}" />
89 </joint>
90
91 <link name="${side2}_link">
92   <inertial>
93     <origin xyz="0 0 0" rpy="0 0 0" />
94     <mass value="${base_mass}" />
95     <box_inertia m="${base_mass}" x="${servo_caja_x}" y="${servo_caja_y}" ...
96       z="${servo_caja_z}"/>
97   </inertial>
98   <inertial>
99     <origin xyz="0.11 2.7756E-17 1.3293E-50" rpy="0 0 0" />
      <mass value="${masa_cilindro}" />
      <cylinder_inertia m="${masa_cilindro}" r="${radio}" h="${longitud}" />

```

```

100    </inertial>
101    <visual>
102        <origin xyz="0 0 0" rpy="0 0 0" />
103        <geometry>
104            <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" />
105        </geometry>
106        <material name=""> <color rgba="0.79216 0 0.93333 1" /> </material>
107    </visual>
108    <visual>
109        <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="${pi/2} 0 ${orientacion}" />
110        <geometry> <cylinder radius="${radio}" length="${longitud}" /> </geometry>
111        <material name=""> <color rgba="1 0 0 1" /> </material>
112    </visual>
113    <collision> <origin xyz="0 0 0" rpy="0 0 0" />
114        <geometry>
115            <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" />
116        </geometry>
117    </collision>
118    <collision>
119        <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="${pi/2} 0 ${orientacion}" />
120        <geometry> <cylinder radius="${radio}" length="${longitud}" /> </geometry>
121    </collision>
122 </link>
123
124
125 <joint name="cilinder_blue_${side2}_joint" type="revolute">
126     <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="0 0 0" />
127     <parent link="${side2}_link" />
128     <child link="blue_${side2}_link" />
129     <axis xyz="${ejex} ${ejey} ${ejez}" />
130     <limit lower = "-2.5" upper = "2.5" effort="${esfuerzoMax}" ...
131         velocity="${velMax}" />
132 </joint>
133 <link name="blue_${side2}_link">
134     <inertial>
135         <origin xyz="0 0 0" rpy="0 0 0" />
136         <mass value="${base_mass}" />
137         <box_inertia m="${base_mass}" x="${servo_caja_x}" y="${servo_caja_y}" ...
138             z="${servo_caja_z}" />
139     </inertial>
140     <visual>
141         <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="0 0 0" />
142         <geometry> <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" /> ...
143             </geometry>
144         <material
145             name="">
146             <color rgba="0 0 1 1" />
147         </material>
148     </visual>
149     <collision>
```

```

148      <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="0 0 0" />
149      <geometry> <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" /> ...
150      </geometry>
151    </collision>
152  </link>
153 </xacro:macro>
154
155
156 <xacro:macro name="servoz" params="side side2 x y z orientacion ejex ejey ejez">
157
158   <joint name="${side}_${side2}_joint" type="fixed">
159     <origin xyz="${x} ${y} ${z}" rpy="0 0 0" />
160     <parent link="${side}_link" />
161     <child link="${side2}_link" />
162     <axis xyz="0 0 0" />
163     <limit lower = "-2.5" upper = "2.5" effort="${esfuerzoMax}" ...
164       velocity="${velMax}" />
165   </joint>
166
167   <link name="${side2}_link">
168     <inertial>
169       <origin xyz="0 0 0" rpy="0 0 0" />
170       <mass value="${5*base_mass}" />
171       <box_inertia m="${base_mass}" x="${servo_caja_x}" y="${servo_caja_y}" ...
172         z="${servo_caja_z}" />
173     </inertial>
174     <inertial>
175       <origin xyz="0.11 2.7756E-17 1.3293E-50" rpy="0 0 0" />
176       <mass value="${masa_cilindro}" />
177       <cylinder_inertia m="${masa_cilindro}" r="${radio}" h="${longitud}" />
178     </inertial>
179     <visual>
180       <origin xyz="0 0 0" rpy="0 0 0" />
181       <geometry>
182         <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" />
183       </geometry>
184       <material name=""> <color rgba="0.79216 0 0.93333 1" /> </material>
185     </visual>
186     <visual>
187       <origin xyz="0 0 ${-longitud/2+servo_caja_z/2}" rpy="0 0 ${orientacion}" />
188       <geometry> <cylinder radius="${radio}" length="${longitud}" /> </geometry>
189       <material name=""> <color rgba="1 0 0 1" /> </material>
190     </visual>
191     <collision> <origin xyz="0 0 0" rpy="0 0 0" />
192       <geometry>
193         <box size="${0.2*servo_caja_x} ${0.2*servo_caja_y} ${0.2*servo_caja_z}" />
194       </geometry>
195     </collision>
196     <collision>
197       <origin xyz="0 0 ${-longitud/2+servo_caja_z/2}" rpy="0 0 ${orientacion}" />

```

```

196      <geometry> <cylinder radius="${0.2*radio}" length="${0.2*longitud}" /> ...
197          </geometry>
198      </link>
199
200  <joint name="cilinder_blue_${side2}_joint" type="revolute">
201      <origin xyz="0 0 ${-longitud/2-servo_caja_z/2}" rpy="0 0 0" />
202      <parent link="${side2}_link" />
203      <child link="blue_${side2}_link" />
204      <axis xyz="${ejex} ${ejey} ${ejez}" />
205      <limit lower = "-2.5" upper = "2.5" effort="${esfuerzoMax}" ...
206          velocity="${velMax}" />
207  </joint>
208
209  <link name="blue_${side2}_link">
210      <inertial>
211          <origin xyz="0 0 0" rpy="0 0 0" />
212          <mass value="${5*base_mass}" />
213          <box_inertia m="${base_mass}" x="${servo_caja_x}" y="${servo_caja_y}" ...
214              z="${servo_caja_z}" />
215      </inertial>
216      <visual>
217          <origin xyz="0 0 ${-longitud/2+servo_caja_z/2}" rpy="0 0 0" />
218          <geometry> <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" /> ...
219              </geometry>
220          <material name="" > <color rgba="0 0 1 1" /> </material>
221      </visual>
222      <collision>
223          <origin xyz="0 0 ${-longitud/2+servo_caja_z/2}" rpy="0 0 0" />
224          <geometry> <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" /> ...
225              </geometry>
226      </collision>
227  </link>
228
229  </xacro:macro>
230
231  <xacro:macro name="servo2ejes" params="side side2 x y z orientacion ejex ejey ...
232      ejez">
233
234  <joint name="${side}_${side2}_joint" type="fixed">
235      <origin xyz="${x} ${y} ${z}" rpy="0 0 0" />
236      <parent link="${side}_link" />
237      <child link="${side2}_link" />
238      <axis xyz="0 0 0" />
239      <limit lower = "-2.5" upper = "2.5" effort="${esfuerzoMax}" ...
          velocity="${velMax}" />
240  </joint>
241
242  <link name="${side2}_link">
243      <inertial>
244          <origin xyz="0 0 0" rpy="0 0 0" />

```

```

240      <mass value="${base_mass}" />
241      <box_inertia m="${base_mass}" x="${servo_caja_x}" y="${servo_caja_y}" ...
242          z="${servo_caja_z}" />
243    </inertial>
244
245    <inertial>
246      <origin xyz="0.11 2.7756E-17 1.3293E-50" rpy="0 0 0" />
247      <mass value="${masa_cilindro}" />
248      <cylinder_inertia m="${masa_cilindro}" r="${radio}" h="${longitud}" />
249    </inertial>
250
251    <visual>
252      <origin xyz="0 0 0" rpy="0 0 0" />
253      <geometry>
254        <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" />
255      </geometry>
256      <material name=""> <color rgba="0.79216 0 0.93333 1" /> </material>
257    </visual>
258
259
260    <visual>
261      <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="${pi/2} 0 ...
262          ${orientacion+pi/2}" />
263      <geometry> <cylinder radius="${radio}" length="${longitud}" /> </geometry>
264      <material name=""> <color rgba="1 0 0 1" /> </material>
265    </visual>
266
267    <collision> <origin xyz="0 0 0" rpy="0 0 0" />
268      <geometry>
269        <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" />
270      </geometry>
271    </collision>
272
273    <collision>
274      <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="${pi/2} 0 ${orientacion}" />
275      <geometry> <cylinder radius="${0.3*radio}" length="${0.3*longitud}" /> ...
276      </geometry>
277    </collision>
278
279
280  <joint name="cilinder_blue1_${side2}_joint" type="revolute">
281    <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="0 0 0" />
282    <parent link="${side2}_link" />
283    <child link="auxiliar${side2}" />
284    <axis xyz="0 1 ${ejez}" />
285    <limit lower = "-2.5" upper = "2.5" effort="${esfuerzoMax}" ...
286        velocity="${velMax}" />
287  </joint>

```

```

287
288 <link name="auxiliar${side2}">
289   <inertial>
290     <origin xyz="0 0 0" rpy="0 0 0" />
291     <mass value="${base_mass}" />
292     <box_inertia m="0" x="0" y="0" z="0"/>
293   </inertial>
294   <visual>
295     <origin xyz="0 0 0" rpy="${pi/2} 0 ${orientacion}" />
296     <geometry> <cylinder radius="${radio}" length="${longitud}" /> </geometry>
297     <material name=""> <color rgba="1 0 0 1" /> </material>
298   </visual>
299   <collision>
300     <origin xyz="0 0 0" rpy="${pi/2} 0 ${orientacion+pi/2}" />
301     <geometry> <cylinder radius="${0.3* radio}" length="${0.3*longitud}" /> ...
302   </collision>
303 </link>
304
305 <joint name="cilinder_blue_${side2}_joint" type="revolute">
306   <origin xyz="0 0 0" rpy="0 0 0" />
307   <parent link="auxiliar${side2}" />
308   <child link="blue_${side2}_link" />
309   <axis xyz="1 0 ${ejez}" />
310   <limit lower = "-2.5" upper = "2.5" effort="${esfuerzoMax}" ...
311     velocity="${velMax}" />
312 </joint>
313
314 <link name="blue_${side2}_link">
315   <inertial>
316     <origin xyz="0 0 0" rpy="0 0 0" />
317     <mass value="${base_mass}" />
318     <box_inertia m="${base_mass}" x="${servo_caja_x}" y="${servo_caja_y}" ...
319       z="${servo_caja_z}" />
320   </inertial>
321   <visual>
322     <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="0 0 0" />
323     <geometry> <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" /> ...
324       </geometry>
325     <material
326       name="">
327       <color rgba="0 0 1 1" />
328     </material>
329   </visual>
330   <collision>
331     <origin xyz="0 0 ${-radio+servo_caja_z/2}" rpy="0 0 0" />
332     <geometry> <box size="${servo_caja_x} ${servo_caja_y} ${servo_caja_z}" /> ...
333       </geometry>
334   </collision>
335 </link>
336

```

```
333  </xacro:macro>
334
335  <xacro:macro name="alargador" params="side side2 x y z orientacion ejex ejey ejez">
336
337  <joint name="${side}_${side2}_joint" type="fixed">
338      <origin xyz="${x} ${y} ${z}" rpy="0 0 0" />
339      <parent link="${side}_link" />
340      <child link="${side2}_link" />
341      <axis xyz="0 0 0" />
342      <limit lower = "-2.5" upper = "2.5" effort="${esfuerzoMax}" ...
343          velocity="${velMax}" />
344  </joint>
345
346  <link name="${side2}_link">
347
348      <inertial>
349          <origin xyz="0 0 0" rpy="0 0 0" />
350          <mass value="${base_mass}" />
351          <box_inertia m="${base_mass}" x="${servo_caja_x}" y="${servo_caja_y}" ...
352              z="${servo_caja_z}" />
353      </inertial>
354
355      <visual>
356          <origin xyz="0 0 0" rpy="0 0 0" />
357          <geometry>
358              <box size="${alargador_caja_x} ${alargador_caja_y} ${alargador_caja_z}" />
359          </geometry>
360          <material name=""> <color rgba="0.79216 0 0.93333 1" /> </material>
361      </visual>
362
363      <collision> <origin xyz="0 0 0" rpy="0 0 0" />
364          <geometry>
365              <box size="${alargador_caja_x} ${alargador_caja_y} ${alargador_caja_z}" />
366          </geometry>
367      </collision>
368  </link>
369
370  <xacro:macro name="pie" params="side side2 x y z orientacion ejex ejey ejez">
371      <joint name="${side}_${side2}_joint" type="fixed">
372          <origin xyz="${x} ${y} ${z}" rpy="0 0 0" />
373          <parent link="${side}_link" />
374          <child link="${side2}_link" />
375          <axis xyz="0 0 0" />
376          <limit lower = "-2.5" upper = "2.5" effort="100" velocity="${velMax}" />
377      </joint>
378      <link name="${side2}_link">
379          <inertial>
380              <origin xyz="0 0 0" rpy="0 0 0" />
381              <mass value="${base_mass}" />
```

```

382     <box_inertia m="${base_mass}" x="${2*servo_caja_x}" ...
383         y="${2*servo_caja_y}" z="${servo_caja_z}"/>
384     </inertial>
385     <visual>
386         <origin xyz="0 0 0" rpy="0 0 0" />
387         <geometry>
388             <box size="${2.5*servo_caja_x*porcentajejelx} ${4*servo_caja_y*porcentajejely} ...
389                 ${servo_caja_z}" />
390         </geometry>
391         <material name=""> <color rgba="0.79216 0 0.93333 1" /> </material>
392     </visual>
393     <collision> <origin xyz="0 0 0" rpy="0 0 0" />
394         <geometry>
395             <box size="${3*servo_caja_x*porcentajejelx} ${4*servo_caja_y*porcentajejely} ...
396                 ${servo_caja_z}" />
397         </geometry>
398     </collision>
399     </link>
400
401 <!-- Build robot model -->
402     <!-- <xacro:pierna side="base_link" side2="2"/>-->
403     <xacro:servoz side="base" side2="box1_izq" ...
404         x="${porcentaje*base_x/2-servo_caja_x/2}" y="0" ...
405         z="${-base_z/2-servo_caja_z/2}" orientacion="${pi/2}" ejex="0" ...
406         ejey="0" ejez="1"/>
407     <xacro:servo2ejes side="blue_box1_izq" side2="box2_izq" x="0" y="0" ...
408         z="${-longitud/2-servo_caja_z/2}" orientacion="0" ejex="1" ejey="0" ...
409         ejez="0"/>
410     <xacro:alargador side="blue_box2_izq" side2="box3_izq" x="0" y="0" ...
411         z="${-radio-alargador_caja_z/2}" orientacion="0" ejex="0" ejey="1" ...
412         ejez="0"/>
413     <xacro:servo side="box3_izq" side2="box4_izq" x="0" y="0" ...
414         z="${-alargador_caja_z/2}" orientacion="${pi/2}" ejex="1" ejey="0" ...
415         ejez="0"/>
416     <xacro:alargador side="blue_box4_izq" side2="box5_izq" x="0" y="0" ...
417         z="${-radio-alargador_caja_z/2}" orientacion="0" ejex="0" ejey="1" ...
418         ejez="0"/>
419     <xacro:servo2ejes side="box5_izq" side2="box6_izq" x="0" y="0" ...
420         z="${-alargador_caja_z/2}" orientacion="0" ejex="1" ejey="0" ejez="0"/>
421     <xacro:pie side="blue_box6_izq" side2="pie_izq" x="0" y="0" ...
422         z="${-radio-servo_caja_z/2}" orientacion="0" ejex="0" ejey="0" ejez="0"/>
423
424     <xacro:servoz side="base" side2="box1_der" ...
425         x="${-porcentaje*base_x/2+servo_caja_x/2}" y="0" ...
426         z="${-base_z/2-servo_caja_z/2}" orientacion="${pi/2}" ejex="0" ...
427         ejey="0" ejez="1"/>

```

```

413     <xacro:servo2ejes side="blue_box1_der" side2="box2_der" x="0" y="0" ...
414         z="${-radio-servo_caja_z/2}" orientacion="0" ejex="1" ejey="0" ejez="0"/>
415     <xacro:alargador side="blue_box2_der" side2="box3_der" x="0" y="0" ...
416         z="${-longitud/2-alargador_caja_z/2}" orientacion="0" ejex="0" ejey="1" ...
417         ejez="0"/>
418     <xacro:servo side="box3_der" side2="box4_der" x="0" y="0" ...
419         z="${-alargador_caja_z/2}" orientacion="${pi/2}" ejex="1" ejey="0" ...
420         ejez="0"/>
421     <xacro:alargador side="blue_box4_der" side2="box5_der" x="0" y="0" ...
422         z="${-radio-alargador_caja_z/2}" orientacion="0" ejex="0" ejey="1" ...
423         ejez="0"/>
424     <xacro:servo2ejes side="box5_der" side2="box6_der" x="0" y="0" ...
425         z="${-alargador_caja_z/2}" orientacion="0" ejex="1" ejey="0" ejez="0"/>
426     <xacro:pie side="blue_box6_der" side2="pie_der" x="0" y="0" ...
427         z="${-radio-servo_caja_z/2}" orientacion="0" ejex="0" ejey="0" ejez="0"/>
428   <!---->
429
430
431
432 </robot>
```

#### 4.7.1. Parametrizar Serpiente

```

1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file 'serp.ui'
4  #
5  # Created: Wed Dec 14 21:22:47 2016
6  #      by: pyside-uic 0.2.15 running on PySide 1.2.4
7  #
8  # WARNING! All changes made in this file will be lost!
9
10 from PySide import QtCore, QtGui
11
12 class Serpiente(object):
13     def setupUi(self, Form):
14         Form.setObjectName("Form")
15         Form.resize(400, 301)
16         self.gridLayout_11 = QtGui.QGridLayout(Form)
17         self.gridLayout_11.setObjectName("gridLayout_11")
18         self.btn_salir = QtGui.QCommandLinkButton(Form)
19         self.btn_salir.setObjectName("btn_salir")
20         self.gridLayout_11.addWidget(self.btn_salir, 1, 0, 1, 1)
21         self.btn_generar = QtGui.QPushButton(Form)
22         self.btn_generar.setObjectName("btn_generar")
23         self.gridLayout_11.addWidget(self.btn_generar, 1, 1, 1, 1)
24         self.tab = QtGui.QStackedWidget(Form)
25         self.tab.setObjectName("tab")
```

```
26         self.tabPage1 = QtGui.QWidget()
27         self.tabPage1.setObjectName("tabPage1")
28         self.gridLayout_3 = QtGui.QGridLayout(self.tabPage1)
29         self.gridLayout_3.setObjectName("gridLayout_3")
30         self.ln_ser_n_servos = QtGui.QLineEdit(self.tabPage1)
31         self.ln_ser_n_servos.setObjectName("ln_ser_n_servos")
32         self.gridLayout_3.addWidget(self.ln_ser_n_servos, 0, 1, 1, 1)
33         self.label_7 = QtGui.QLabel(self.tabPage1)
34         self.label_7.setObjectName("label_7")
35         self.gridLayout_3.addWidget(self.label_7, 0, 0, 1, 1)
36         self.tab.addWidget(self.tabPage1)
37         self.tabPage2 = QtGui.QWidget()
38         self.tabPage2.setObjectName("tabPage2")
39         self.gridLayout_4 = QtGui.QGridLayout(self.tabPage2)
40         self.gridLayout_4.setObjectName("gridLayout_4")
41         self.label_24 = QtGui.QLabel(self.tabPage2)
42         self.label_24.setObjectName("label_24")
43         self.gridLayout_4.addWidget(self.label_24, 2, 0, 1, 1)
44         self.ln_hex_long_x_cuerpo = QtGui.QLineEdit(self.tabPage2)
45         self.ln_hex_long_x_cuerpo.setObjectName("ln_hex_long_x_cuerpo")
46         self.gridLayout_4.addWidget(self.ln_hex_long_x_cuerpo, 2, 2, 1, 1)
47         self.label_12 = QtGui.QLabel(self.tabPage2)
48         self.label_12.setObjectName("label_12")
49         self.gridLayout_4.addWidget(self.label_12, 6, 0, 1, 1)
50         self.ln_hex_long_tibia = QtGui.QLineEdit(self.tabPage2)
51         self.ln_hex_long_tibia.setObjectName("ln_hex_long_tibia")
52         self.gridLayout_4.addWidget(self.ln_hex_long_tibia, 6, 2, 1, 1)
53         self.ln_hex_n_servos = QtGui.QLineEdit(self.tabPage2)
54         self.ln_hex_n_servos.setObjectName("ln_hex_n_servos")
55         self.gridLayout_4.addWidget(self.ln_hex_n_servos, 1, 2, 1, 1)
56         self.label_13 = QtGui.QLabel(self.tabPage2)
57         self.label_13.setObjectName("label_13")
58         self.gridLayout_4.addWidget(self.label_13, 5, 0, 1, 1)
59         self.label_11 = QtGui.QLabel(self.tabPage2)
60         self.label_11.setObjectName("label_11")
61         self.gridLayout_4.addWidget(self.label_11, 4, 0, 1, 1)
62         self.ln_hex_long_femur = QtGui.QLineEdit(self.tabPage2)
63         self.ln_hex_long_femur.setObjectName("ln_hex_long_femur")
64         self.gridLayout_4.addWidget(self.ln_hex_long_femur, 5, 2, 1, 1)
65         self.label_9 = QtGui.QLabel(self.tabPage2)
66         self.label_9.setObjectName("label_9")
67         self.gridLayout_4.addWidget(self.label_9, 1, 0, 1, 1)
68         self.ln_hex_long_coxa = QtGui.QLineEdit(self.tabPage2)
69         self.ln_hex_long_coxa.setObjectName("ln_hex_long_coxa")
70         self.gridLayout_4.addWidget(self.ln_hex_long_coxa, 4, 2, 1, 1)
71         self.label_25 = QtGui.QLabel(self.tabPage2)
72         self.label_25.setObjectName("label_25")
73         self.gridLayout_4.addWidget(self.label_25, 3, 0, 1, 1)
74         self.ln_hex_lon_y_cuerpo = QtGui.QLineEdit(self.tabPage2)
75         self.ln_hex_lon_y_cuerpo.setObjectName("ln_hex_lon_y_cuerpo")
76         self.gridLayout_4.addWidget(self.ln_hex_lon_y_cuerpo, 3, 2, 1, 1)
```

```
77         self.tab.addWidget(self.tabPage2)
78         self.tabPage3 = QtGui.QWidget()
79         self.tabPage3.setObjectName("tabPage3")
80         self.gridLayout_12 = QtGui.QGridLayout(self.tabPage3)
81         self.gridLayout_12.setObjectName("gridLayout_12")
82         self.label_78 = QtGui.QLabel(self.tabPage3)
83         self.label_78.setMaximumSize(QtCore.QSize(16777215, 20))
84         self.label_78.setObjectName("label_78")
85         self.gridLayout_12.addWidget(self.label_78, 0, 2, 1, 1)
86         self.label_43 = QtGui.QLabel(self.tabPage3)
87         self.label_43.setObjectName("label_43")
88         self.gridLayout_12.addWidget(self.label_43, 3, 0, 1, 1)
89         self.label_44 = QtGui.QLabel(self.tabPage3)
90         self.label_44.setObjectName("label_44")
91         self.gridLayout_12.addWidget(self.label_44, 4, 0, 1, 1)
92         self.ln_movil_batalla = QtGui.QLineEdit(self.tabPage3)
93         self.ln_movil_batalla.setObjectName("ln_movil_batalla")
94         self.gridLayout_12.addWidget(self.ln_movil_batalla, 3, 2, 1, 1)
95         self.ln_movil_radio_rue = QtGui.QLineEdit(self.tabPage3)
96         self.ln_movil_radio_rue.setObjectName("ln_movil_radio_rue")
97         self.gridLayout_12.addWidget(self.ln_movil_radio_rue, 2, 2, 1, 1)
98         self.label_60 = QtGui.QLabel(self.tabPage3)
99         self.label_60.setObjectName("label_60")
100        self.gridLayout_12.addWidget(self.label_60, 1, 0, 1, 1)
101        self.ln_movil_ancho_vias = QtGui.QLineEdit(self.tabPage3)
102        self.ln_movil_ancho_vias.setObjectName("ln_movil_ancho_vias")
103        self.gridLayout_12.addWidget(self.ln_movil_ancho_vias, 4, 2, 1, 1)
104        self.label_10 = QtGui.QLabel(self.tabPage3)
105        self.label_10.setObjectName("label_10")
106        self.gridLayout_12.addWidget(self.label_10, 2, 0, 1, 1)
107        self.ln_movil_tipo = QtGui.QLineEdit(self.tabPage3)
108        self.ln_movil_tipo.setObjectName("ln_movil_tipo")
109        self.gridLayout_12.addWidget(self.ln_movil_tipo, 1, 2, 1, 1)
110        self.tab.addWidget(self.tabPage3)
111        self.gridLayout_11.addWidget(self.tab, 0, 0, 1, 2)
112
113        self.retranslateUi(Form)
114        QtCore.QMetaObject.connectSlotsByName(Form)
115
116    def retranslateUi(self, Form):
117        Form.setWindowTitle(QtGui.QApplication.translate("Form", "Form", None, ...
118                                         QtGui.QApplication.UnicodeUTF8))
119        self.label_9.setText(QtGui.QApplication.translate("Form", "n servos", ...
120                                         None, QtGui.QApplication.UnicodeUTF8))
```

#### 4.7.2. Xacro Serpiente

```
1  <?xml version="1.0"?>
2  <robot
3    name="serp"
4    xmlns:xacro="http://www.ros.org/wiki/xacro">
5
6    <link name="map">
7    </link>
8
9      <xacro:property name="pi" value="3.1415926535897931" />
10
11     <xacro:property name="n_servo" value="$(arg numero_servo)" />
12
13   <xacro:macro name="eslabon" params="parent position x y angle axis">
14     <joint name="${parent}" type="fixed">
15       <origin xyz="${x} 0 0" rpy="0 0 0" />
16       <parent link="${parent}" />
17       <child link="base_link${position}" />
18     </joint>
19
20
21   <link
22     name="base_link${position}">
23     <inertial>
24       <origin
25         xyz="${x+0.03} 0 -0.0015809"
26         rpy="0 0 0" />
27       <mass
28         value="0.2" />
29       <inertia
30         ixx="0.033333"
31         ixy="7.6472E-21"
32         ixz="-1.039E-21"
33         iyy="0.017333"
34         iyz="2.322E-21"
35         izz="0.017333" />
36     </inertial>
37     <visual>
38       <origin
39         xyz="0 0 0"
40         rpy="0 0 0" />
41       <geometry>
42         <mesh
43           filename="package://serpiente_description/urdf/meshes_serpiente/base_link.stl" ...
44           />
45         <!!-- <mesh ...
46           filename="file:///home/kaiser/catkin_ws_CromaBueno/src/croma_description/meshes/base_link.stl"
47           /> -->
48       </geometry>
49       <material
50         name="n">
51         <color
```

```
49         rgba="0.79216 0 0.93333 1" />
50     </material>
51   </visual>
52   <collision>
53     <origin
54       xyz="0 0 0"
55       rpy="0 0 0" />
56     <geometry>
57       <mesh
58         filename="package://serpiente_description/urdf/meshes_serpiente/base_link.stl" ...
59           />
60     </geometry>
61   </collision>
62 </link>
63
64 <link
65   name="cylinder${position}">
66   <inertial>
67     <origin
68       xyz="${x+0.07} 2.7756E-17 1.3293E-50"
69       rpy="0 0 0" />
70     <mass
71       value="0.7854" />
72     <inertia
73       ixx="0.011454"
74       ixy="2.2588E-20"
75       ixz="3.7494E-36"
76       iyy="0.0098175"
77       iyz="3.7944E-36"
78       izz="0.011454" />
79   </inertial>
80   <visual>
81     <origin
82       xyz="0 0 0"
83       rpy="0 0 0" />
84     <geometry>
85       <mesh
86         filename="package://serpiente_description/urdf/meshes_serpiente/cylinder.stl" ...
87           />
88     </geometry>
89     <material
90       name="">
91       <color
92         rgba="1 0 0 1" />
93     </material>
94   </visual>
95   <collision>
96     <origin
97       xyz="0 0 0"
98       rpy="0 0 0" />
99     <geometry>
```

```
98      <mesh
99          filename="package://serpiente_description/urdf/meshes_serpiente/cylinder.stl" ...
100         />
101     </geometry>
102   </collision>
103 </link>
104
105 <joint
106   name="base_cylinder_joint${position}"
107   type="fixed">
108   <origin
109     xyz="0 0 0"
110     rpy="${angle} 0 0" />
111   <parent
112     link="base_link${position}" />
113   <child
114     link="cylinder${position}" />
115   <axis
116     xyz="0 0 0" />
117 </joint>
118
119 <link
120   name="blue_box${position}">
121   <inertial>
122     <origin
123       xyz="0.04 0 2.0668E-19"
124       rpy="0 0 0" />
125     <mass
126       value="0.2" />
127     <inertia
128       ixx="0.033333"
129       ixy="3.4121E-21"
130       ixz="-1.3214E-21"
131       iyy="0.017333"
132       iyz="2.322E-21"
133       izz="0.017333" />
134   </inertial>
135   <visual>
136     <origin
137       xyz="0 0 0"
138       rpy="0 0 0" />
139     <geometry>
140       <mesh
141         filename="package://serpiente_description/urdf/meshes_serpiente/blue_box.stl" ...
142         />
143     </geometry>
144     <material
145       name="">
146       <color
147         rgba="0 0 1 1" />
148     </material>
```

```
147     </visual>
148     <collision>
149         <origin
150             xyz="0 0 0"
151             rpy="0 0 0" />
152         <geometry>
153             <mesh
154                 filename="package://serpiente_description/urdf/meshes_serpiente/blue_box.stl" ...
155             />
156         </geometry>
157     </collision>
158 </link>
159
160 <joint
161     name="box_cilinder_joint${position}"
162     type="revolute">
163     <origin
164         xyz="0.05 0 0"
165         rpy="0 0 0" />
166     <parent
167         link="cilinder${position}" />
168     <child
169         link="blue_box${position}" />
170     <axis
171         xyz="0 1 0" />
172     <limit
173         lower = "-2.5"
174         upper = "2.5"
175         effort="100"
176         velocity="1000" />
177     </joint>
178
179 <xacro:if value="${axis}">
180     <gazebo reference="base_link${position}">
181         <material>Gazebo/Yellow</material>
182     </gazebo>
183     <gazebo reference="cilinder${position}">
184         <material>Gazebo/Yellow</material>
185     </gazebo>
186     <gazebo reference="blue_box${position}">
187         <material>Gazebo/Yellow</material>
188     </gazebo>
189 </xacro:if>
190
191 <xacro:unless value="${axis}">
192     <gazebo reference="base_link${position}">
193         <material>Gazebo/Green</material>
194     </gazebo>
195     <gazebo reference="cilinder${position}">
196         <material>Gazebo/Green</material>
```

```
197  </gazebo>
198  <gazebo reference="blue_box${position}">
199    <material>Gazebo/Green</material>
200  </gazebo>
201  </xacro:unless>
202
203
204
205
206  <transmission name="t_box_cylinder_joint${position}">
207    <type>transmission_interface/SimpleTransmission</type>
208    <joint name="box_cylinder_joint${position}">
209      <hardwareInterface>EffortJointInterface</hardwareInterface>
210    </joint>
211    <actuator name="m_box_cylinder_joint${position}">
212      <hardwareInterface>EffortJointInterface</hardwareInterface>
213      <mechanicalReduction>1</mechanicalReduction>
214    </actuator>
215  </transmission>
216
217 </xacro:macro>
218
219
220
221  <gazebo>
222    <plugin filename="libgazebo_ros_control.so" name="gazebo_ros_control">
223      <robotNamespace>/serp</robotNamespace>
224    </plugin>
225  </gazebo>
226
227
228
229  <!--scale="1 0.95 1"-->
230  <xacro:unless value="${n_servo-2}">
231    <xacro:eslabon parent="map" position="1" x="0.01" y="-0.08335" angle="0" ...
232      axis="0" />
233    <xacro:eslabon parent="blue_box1" position="2" x="0.05" y="-0.08335" ...
234      angle="${pi/2}" axis="1" />
235  </xacro:unless>
236
237  <xacro:unless value="${n_servo-3}">
238    <xacro:eslabon parent="map" position="1" x="0.01" y="-0.08335" angle="0" ...
239      axis="0" />
240    <xacro:eslabon parent="blue_box1" position="2" x="0.05" y="-0.08335" ...
241      angle="${pi/2}" axis="1" />
242  </xacro:unless>
243
244  <xacro:unless value="${n_servo-4}">
```

```

242 <xacro:eslabon parent="map" position="1" x="0.01" y="-0.08335" angle="0" ...
243   axis="0" />
244 <xacro:eslabon parent="blue_box1" position="2" x="0.05" y="-0.08335" ...
245   angle="${pi/2}" axis="1" />
246 <xacro:eslabon parent="blue_box2" position="3" x="0.05" y="-0.08335" ...
247   angle="-${pi/2}" axis="0" />
248 <xacro:eslabon parent="blue_box3" position="4" x="0.05" y="-0.08335" ...
249   angle="${pi/2}" axis="1" />
250 </xacro:unless>
251
252 <xacro:unless value="${n_servo-5}">
253 <xacro:eslabon parent="map" position="1" x="0.01" y="-0.08335" angle="0" ...
254   axis="0" />
255 <xacro:eslabon parent="blue_box1" position="2" x="0.05" y="-0.08335" ...
256   angle="${pi/2}" axis="1" />
257 <xacro:eslabon parent="blue_box2" position="3" x="0.05" y="-0.08335" ...
258   angle="-${pi/2}" axis="0" />
259 <xacro:eslabon parent="blue_box3" position="4" x="0.05" y="-0.08335" ...
260   angle="${pi/2}" axis="1" />
261 <xacro:eslabon parent="blue_box4" position="5" x="0.05" y="-0.08335" ...
262   angle="-${pi/2}" axis="0" />
263 <xacro:eslabon parent="blue_box5" position="6" x="0.05" y="-0.08335" ...
264   angle="${pi/2}" axis="1" />
265 </xacro:unless>
266 <xacro:unless value="${n_servo-6}">
267 <xacro:eslabon parent="map" position="1" x="0.01" y="-0.08335" angle="0" ...
268   axis="0" />
269 <xacro:eslabon parent="blue_box1" position="2" x="0.05" y="-0.08335" ...
270   angle="${pi/2}" axis="1" />
271 <xacro:eslabon parent="blue_box2" position="3" x="0.05" y="-0.08335" ...
272   angle="-${pi/2}" axis="0" />
273 <xacro:eslabon parent="blue_box3" position="4" x="0.05" y="-0.08335" ...
274   angle="${pi/2}" axis="1" />
275 <xacro:eslabon parent="blue_box4" position="5" x="0.05" y="-0.08335" ...
276   angle="-${pi/2}" axis="0" />
277 <xacro:eslabon parent="blue_box5" position="6" x="0.05" y="-0.08335" ...
278   angle="${pi/2}" axis="1" />

```

```

272 <xacro:eslabon parent="blue_box6" position="7" x="0.05" y="-0.08335" ...
273   angle="-${pi/2}" axis="0" />
274 </xacro:unless>
275 <xacro:unless value="${n_servo-8}">
276 <xacro:eslabon parent="map" position="1" x="0.01" y="-0.08335" angle="0" ...
277   axis="0" />
278 <xacro:eslabon parent="blue_box1" position="2" x="0.05" y="-0.08335" ...
279   angle="${pi/2}" axis="1" />
280 <xacro:eslabon parent="blue_box2" position="3" x="0.05" y="-0.08335" ...
281   angle="-${pi/2}" axis="0" />
282 <xacro:eslabon parent="blue_box3" position="4" x="0.05" y="-0.08335" ...
283   angle="${pi/2}" axis="1" />
284 <xacro:eslabon parent="blue_box4" position="5" x="0.05" y="-0.08335" ...
285   angle="-${pi/2}" axis="0" />
286 <xacro:eslabon parent="blue_box5" position="6" x="0.05" y="-0.08335" ...
287   angle="${pi/2}" axis="1" />
288 <xacro:eslabon parent="blue_box6" position="7" x="0.05" y="-0.08335" ...
289   angle="-${pi/2}" axis="0" />
290 <xacro:eslabon parent="blue_box7" position="8" x="0.05" y="-0.08335" ...
291   angle="${pi/2}" axis="1" />
292 <xacro:eslabon parent="blue_box8" position="9" x="0.05" y="-0.08335" ...
293   angle="-${pi/2}" axis="0" />
294 <xacro:eslabon parent="blue_box9" position="10" x="0.05" y="-0.08335" ...
295   angle="${pi/2}" axis="1" />
296 <xacro:eslabon parent="blue_box10" position="11" x="0.05" y="-0.08335" ...
297   angle="-${pi/2}" axis="0" />
298 </xacro:unless>
299 <xacro:unless value="${n_servo-10}">
300 <xacro:eslabon parent="map" position="1" x="0.01" y="-0.08335" angle="0" ...
301   axis="0" />
302 <xacro:eslabon parent="blue_box1" position="2" x="0.05" y="-0.08335" ...
303   angle="${pi/2}" axis="1" />
```

```
302 <xacro:eslabon parent="blue_box2" position="3" x="0.05" y="-0.08335" ...  
     angle="-${pi/2}" axis="0" />  
303 <xacro:eslabon parent="blue_box3" position="4" x="0.05" y="-0.08335" ...  
     angle="${pi/2}" axis="1" />  
304 <xacro:eslabon parent="blue_box4" position="5" x="0.05" y="-0.08335" ...  
     angle="-${pi/2}" axis="0" />  
305 <xacro:eslabon parent="blue_box5" position="6" x="0.05" y="-0.08335" ...  
     angle="${pi/2}" axis="1" />  
306 <xacro:eslabon parent="blue_box6" position="7" x="0.05" y="-0.08335" ...  
     angle="-${pi/2}" axis="0" />  
307 <xacro:eslabon parent="blue_box7" position="8" x="0.05" y="-0.08335" ...  
     angle="${pi/2}" axis="1" />  
308 <xacro:eslabon parent="blue_box8" position="9" x="0.05" y="-0.08335" ...  
     angle="-${pi/2}" axis="0" />  
309 <xacro:eslabon parent="blue_box9" position="10" x="0.05" y="-0.08335" ...  
     angle="${pi/2}" axis="0" />  
310 </xacro:unless>  
311  
312 </robot>
```

## 5. Anexo B. Programa final para el diseño automático de manipuladores de 3 grados de libertad. Código MATLAB

A continuación se muestra el código del programa desarrollado en MATLAB para la generación automática del manipulador.

Para ejecutar el programa hay que usar la primera función (código 2), pues es la encargada de recoger los datos de entrada y llamar al resto de funciones.

Para poder ejecutar correctamente el código es necesario incluir la última función (código 10) dentro del directorio de la Toolbox (en directorio de instalación de MATLAB → toolbox → rvctools → robot → @SerialLink)

Listing 2: V3\_calcular\_robot\_3DOF\_programa

```

1 startup_rvc %EJECUTAR ESTO SIEMPRE QUE SE INICIE MATLAB
2
3 %Programa que permite analizar y calcular automaticamente un robot de 3
4 %grados de libertad con una estructura y distribucion ya predefinida.
5 %El programa entrega por pantalla todos los datos necesarios para el diseño
6 %y construccion del robot.
7 %En la estructura datos_robot se pueden encontrar todos los datos de cada
8 %eslabon del mismo
9
10 %ENTRADA DE DATOS
11 %Entrada por teclado de los datos necesarios, longitud total del robot y
12 %masa a levantar
13 longitud_total_robot= input('Introduce la longitud total del robot (en cm): ')/100;
14 m = input('Introduce la masa que debe levantar el robot (en kg): ');
15
16 %Declaracion de variables necesarias
17 ok=0;
18 ok_dinamico=0;%variable para comprobar si el diseño ha cambiado tras en ...
    %analisis dinamico
19 ok_estatico=0;%variable para comprobar si el diseño ha cambiado tras en ...
    %analisis estatico
20 ok_optimizacion=0;%variable para comprobar si se ha checkeado la optimizacion ...
    %de los links
21 ok_pre_optimizacion=0;
22 i=0; %numero de iteraciones que ha necesitado el programa para calcular el robot
23
24
25 %Estructura en la que se almacenan todos los datos del robot
26 datos_robot = struct ([]);
27

```

```
28 %DEFINICION TIPO SERVO
29 servos_iniciales=[1 1 1]; %Inicialmente suponemos que todos los servos son medios
30 %servo_pequeno=0 servo_medio=1 dinamixel=2
31
32 %Iniciamos cada iteracion con una configuracion de servos. Tras definir y
33 %analizar el robot volvemos a determinar el tipo de servos necesarios. Si
34 %la configuracion de servos inicial y final es la misma, el robot esta
35 %bien calculado y salimos del bucle. Si no, se realiza otra iteracion hasta
36 %encontrar un robot que cumpla las condiciones
37
38 while ok_optimizacion==0
39
40     if ok_pre_optimizacion==1
41         ok_optimizacion=1;
42     end
43
44 while ok==0
45
46 while ok_estatico==0
47 %Se actualiza el numero de iteraciones
48 i=i+1;
49
50 ok_dinamico=0;
51
52 %ROBOT DEFINITION
53 %Se define el robot y se extraen todos los datos necesarios para calcular
54 %los pares
55
56 fprintf('\n DEFINIENDO ROBOT. ITERACION %d \n',i)
57
58 [R,datos_robot] = V3_definir_robot_3DOF(longitud_total_robot,servos_iniciales, ...
    i, datos_robot);
59
60
61 %ANALISIS TORQUE ESTATICO
62 fprintf('\n ANALISIS TORQUE ESTATICO \n')
63 [ torque_estatico, wbase, momentos, fuerzas ] = ...
    V3_calcular_pares_3DOF_estatico( R,m );
64 datos_robot(1).momentos =momentos(:,1);
65 datos_robot(2).momentos =momentos(:,2);
66 datos_robot(3).momentos =momentos(:,3);
67 datos_robot(1).fuerzas =fuerzas(:,1);
68 datos_robot(2).fuerzas =fuerzas(:,2);
69 datos_robot(3).fuerzas =fuerzas(:,3);
70
71 %SELECCION TIPO SERVO (ESTATICO)
72 %En funcion del par se elige el tipo de servo necesario
73 fprintf('\n COMPROBANDO SERVO ESTATICO \n')
74 [ servos_estatico ] = V3_elegir_servos_3DOF( torque_estatico, servos_iniciales );
75
76 %Tras determinar el tipo de servo, se comprueba si es igual a la
```

```
77 %suposicion inicial
78 if isequal(servos_iniciales,servos_estatico) %El robot esta bien calculado
79     fprintf('\n LOS SERVOS NO HAN CAMBIADO \n')
80     ok_estatico=1;
81 else %El robot no esta bien calculado
82     fprintf('\n LOS SERVOS HAN CAMBIADO, VOLVIENDO AL PRINCIPIO \n')
83     servos_iniciales=servos_estatico;
84     ok_optimizacion=0;
85     ok_pre_optimizacion=0;
86 end
87
88 end
89
90 while ok_dinamico==0
91
92 %ANALISIS TORQUE DINAMICO
93 %Calculos rne
94 fprintf('\n ANALISIS TORQUE DINAMICO \n')
95 [ torque_dinamico, wbase, momentos, fuerzas] = V3_calcular_pares_3DOF_rne( R,m, ...
96     datos_robot, servos_estatico);
97 datos_robot(1).momentos =momentos(:,1);
98 datos_robot(2).momentos =momentos(:,2);
99 datos_robot(3).momentos =momentos(:,3);
100 datos_robot(1).fuerzas =fuerzas(:,1);
101 datos_robot(2).fuerzas =fuerzas(:,2);
102 datos_robot(3).fuerzas =fuerzas(:,3);
103 %SELECCION TIPO SERVO (DINAMICO)
104 %Elección de servos tras rne
105 fprintf('\n COMPROBANDO SERVO DINAMICO \n')
106 [ servos_dinamico ] = V3_elegir_servos_3DOF( torque_dinamico, servos_estatico );
107
108 %Comprobacion servos estaticos y finales
109 if isequal(servos_estatico,servos_dinamico) %El robot esta bien calculado
110     fprintf('\n LOS SERVOS NO HAN CAMBIADO \n')
111     ok_dinamico=1;
112 else %El robot no esta bien calculado
113     fprintf('\n LOS SERVOS HAN CAMBIADO, VOLVIENDO AL PRINCIPIO \n')
114     servos_iniciales=servos_dinamico;
115     ok_estatico=0;
116     ok_dinamico=1;
117     ok_optimizacion=0;
118     ok_pre_optimizacion=0;
119 end
120 end
121
122 if ok_dinamico == 1 && ok_estatico == 1
123     ok = 1;
124 end
125
126 end %end ok
```

```
127
128 %COMPROBACIoN DE LA OPTIMIZACION DE LOS LINKS
129 if ok_optimizacion==0;
130     ok_pre_optimizacion=1;
131     ok_estatico=0;
132     ok_dinamico=0;
133     ok=0;
134
135     fprintf('\n ANaLISIS OK. VOLVIENDO AL PRINCIPIO PARA OPTIMIZAR LINKS \n')
136 end
137
138 end
139
140 fprintf('\n ROBOT CALCULADO \n')
141
142 %Se imprimen por pantalla los resultados
143 fprintf('\nDIMENSIONES ELEMENTOS PASIVOS \n')
144 for n=1:3
145     if datos_robot(n).long_elemento_parametrizable == 0
146         fprintf('Elemento parametrizable eslabon %d: Sin elemento ...
147             parametrizable \n',n);
148     else
149         fprintf('Elemento parametrizable eslabon %d: longitud %f cm, radio exterior ...
150             %f mm, radio interior %f mm\n', n, ...
151                 datos_robot(n).long_elemento_parametrizable*100, ...
152                 datos_robot(n).radio(1)*1000, datos_robot(n).radio(2)*1000)
153     end
154 end
155
156
157 fprintf('\n \nTIPO DE MODULOS \n')
158 for n=1:3
159     if servos_dinamico(n)==0;
160         fprintf('Modulo %d: pequeno \n',n);
161     elseif servos_dinamico(n)==1;
162         fprintf('Modulo %d: medio \n',n);
163     elseif servos_dinamico(n)==2;
164         fprintf('Modulo %d: dinamixel \n',n);
165     end
166 end
167
168
169 fprintf('\n \nTORQUE NECESARIO \n')
170 for n=1:3
171     fprintf('El torque maximo requerido en el servo %d es: %f Kg-cm ...
172             \n',n,torque_dinamico(n))
173 end
```

Listing 3: V3\_definir\_robot\_3DOF

```

1  function [R,datos_robot] = V3_definir_robot_3DOF( longitud_total_robot, ...
2      tipo_servo, numero_iteracion, datos_robot)
3  %Funcion para calcular la longitud de los elementos pasivos y definir el
4  %robot, en funcion de la longitud total del robot y del tipo de modulos a
5  %utilizar
6  %Entradas:
7  %-longitud_total_robot
8  %-tipo_servo: vector de 1x3 en el que se especifica el tipo de modulo a
9  %utilizar, donde: servo_pequeno=0 servo_medio=1 dinamixel=2
10 %Salidas:
11 %R: robot definido
12 %datos_robot: estructura que contiene todos los datos del robot. Contiene
13 %tantas filas como eslabones tiene el robot
14
15 %CALCULO DE LONGITUDES DE ESLABONES
16 %Medidas fijas
17 base_SM=0.06065;
18 parte_inferior_SP=0.026;
19 parte_inferior_SM=0.0426;
20 parte_inferior_dinamixel=0.0446;
21 radio=0.045;
22 pinza=0.078;
23
24 %Longitud minima que puede tener el robot.
25 longitud_minima=base_SM+parte_inferior_SM*2+radio*2+pinza;
26 if longitud_total_robot<longitud_minima
27     error('Error. No se puede hacer un robot tan pequeno.')
28 end
29
30 for n=1:3
31 datos_robot(n).long_total_eslabon = longitud_total_robot/3;
32 end
33
34 %Definicion de la longitud de los elementos dependientes del tipo de servo
35 %del eslabon
36 %Eslalon 1
37 if tipo_servo(2)==1
38     datos_robot(1).long_elemento_fijo_2= parte_inferior_SM;
39 elseif tipo_servo(2)==0
40     datos_robot(1).long_elemento_fijo_2= parte_inferior_SP;
41 elseif tipo_servo(2)==2
42     datos_robot(1).long_elemento_fijo_2= parte_inferior_dinamixel;
43 end
44 %Eslalon 2
45 if tipo_servo(3)==1
46     datos_robot(2).long_elemento_fijo_2= parte_inferior_SM;
47 elseif tipo_servo(3)==0
48     datos_robot(2).long_elemento_fijo_2= parte_inferior_SP;
49 elseif tipo_servo(3)==2

```

```

50     datos_robot(2).long_elemento_fijo_2= parte_inferior_dinamixel;
51 end
52
53 %Definicion de la dimension de elementos independientes del tipo de
54 %servo
55 datos_robot(1).long_elemento_fijo_1= base_SM;
56 datos_robot(2).long_elemento_fijo_1= radio;
57 datos_robot(3).long_elemento_fijo_1= radio;
58 datos_robot(3).long_elemento_fijo_2= pinza;
59
60 %Calculo de la longitud del elemento parametrizable
61 for n=1:3
62     datos_robot(n).long_elemento_parametrizable = ...
63         datos_robot(n).long_total_eslabon-datos_robot(n).long_elemento_fijo_1-datos_robot(n).long_elem
64 end
65
66 %Comprobacion de las dimensiones de los elementos parametrizables.
67 %Si la dimension de alguno de los pasivos sale negativa (es decir, la suma
68 %de los elementos fijos ya es mayor que la longitud que debe tener el
69 %eslabon total) se resta esta medida al segundo eslabon (donde nunca vamos a ...
70 %tener este problema)
71 %para compensar lo grande de mas que es el otro eslabon, y se pone ese eslabon ...
72 % a 0
73
74 %Comprobacion eslabon 1
75 if datos_robot(1).long_elemento_parametrizable < 0.02 && ...
76     datos_robot(1).long_elemento_parametrizable ≥ 0
77     datos_robot(2).long_elemento_parametrizable = ...
78         datos_robot(2).long_elemento_parametrizable + ...
79         datos_robot(1).long_elemento_parametrizable;
80     datos_robot(1).long_elemento_parametrizable = 0;
81 elseif datos_robot(1).long_elemento_parametrizable<0
82     datos_robot(2).long_elemento_parametrizable = ...
83         datos_robot(2).long_elemento_parametrizable - ...
84         abs(datos_robot(1).long_elemento_parametrizable);
85     datos_robot(1).long_elemento_parametrizable = 0;
86 end
87
88 % Comprobacion eslabon 3
89 if datos_robot(3).long_elemento_parametrizable < 0.02 && ...
90     datos_robot(3).long_elemento_parametrizable ≥ 0
91     datos_robot(2).long_elemento_parametrizable = ...
92         datos_robot(2).long_elemento_parametrizable + ...
93         datos_robot(3).long_elemento_parametrizable;
94     datos_robot(3).long_elemento_parametrizable = 0;
95 elseif datos_robot(3).long_elemento_parametrizable<0
96     datos_robot(2).long_elemento_parametrizable = ...
97         datos_robot(2).long_elemento_parametrizable - ...
98         abs(datos_robot(3).long_elemento_parametrizable);
99     datos_robot(3).long_elemento_parametrizable = 0;
100 end

```

```

88
89 % Comprobacion eslabon 2
90 if datos_robot(2).long_elemento_parametrizable<0.02
91   datos_robot(2).long_elemento_parametrizable=0;
92 end
93
94 %Se recalcula la longitud total de los eslabones (si la longitud de los ...
95 %elementos parametrizables ha cambiado en el paso anterior)
96 for n=1:3
97   datos_robot(n).long_total_eslabon = datos_robot(n).long_elemento_fijo_1 + ...
98     datos_robot(n).long_elemento_fijo_2 + ...
99     datos_robot(n).long_elemento_parametrizable;
100 end
101
102 %Definicion manual del tipo de articulacion de revolucion (axial o radial)
103 %de cada eslabon
104 %0=axial 1=radial
105 datos_robot(1).tipo_articulacion = 0;
106 datos_robot(2).tipo_articulacion = 1;
107 datos_robot(3).tipo_articulacion = 1;
108
109 %DEFINICION Y OPTIMIZACION DE LINKS Y ESLABONES COMPLETOS
110 %Llamada a la funcion masa_cdm_inercia, donde se van a definir los elementos ...
111 %parametrizables y a extraer la
112 %informacion de los eslabones del robot
113
114 for n=1:3
115   %En el tercer eslabon siempre colocamos un efector final (pinza)
116   % if n == 3
117   %   servo = 3; %Entrada para la funcion masa_cdm_inercia en caso el motor ...
118   %   tenga una pinza
119   % else
120   %   servo = tipo_servo(n);
121   % end
122
123   %Si estamos en la primera iteracion del programa usamos unos links
124   %genericos ya que aun no tenemos informacion de los analisis de par
125   if numero_iteracion==1
126     fprintf('\n OPTIMIZANDO PRIMERA ITERACION ESLABON %d\n', n)
127     [datos_robot(n).masa, datos_robot(n).cdm, datos_robot(n).inercia, ...
128      datos_robot(n).radio] = ...
129      V3_masa_cdm_inercia_primera_iteracion(n,tipo_servo(n), ...
130      datos_robot(n).long_elemento_parametrizable, ...
131      datos_robot(n).tipo_articulacion);
132
133   %En la iteraciones sucesivas se incluye la informacion de los analisis y
134   %se calculan los links optimizados
135   elseif numero_iteracion>1
136     fprintf('\n OPTIMIZANDO ESLABON %d\n', n)
137     [datos_robot(n).masa, datos_robot(n).cdm, datos_robot(n).inercia, ...
138      datos_robot(n).radio] = V3_masa_cdm_inercia(n, tipo_servo(n), ...

```

```

        datos_robot(n).long_elemento_parametrizable, datos_robot(n).momentos, ...
        datos_robot(n).fuerzas,datos_robot(n).tipo_articulacion);
129 end
130
131 end
132
133 %Calculo de redondeos
134 for n=1:3
135
136 %Redondeo externo
137 datos_robot(n).redondeo_externo=datos_robot(n).long_elemento_parametrizable/5;
138
139 if datos_robot(n).redondeo_externo > (0.02-datos_robot(n).radio(1))
140     datos_robot(n).redondeo_externo = 0.02-datos_robot(n).radio(1);
141 end
142
143 %Redondeo interno
144
145 datos_robot(n).redondeo_interno=datos_robot(n).radio(2)*0.90;
146
147
148 end
149
150 %DEFINICION DEL ROBOT
151
152 rad= pi/180;
153
154 %Primer eslabon
155 L(1) = Revolute('d', datos_robot(1).long_total_eslabon, 'a', 0, 'alpha', pi/2, ...
156 'offset', 0, ...
157 'I', datos_robot(1).inercia, ...
158 'r', [0, 0, datos_robot(1).cdm], ...
159 'm', datos_robot(1).masa, ...
160 'qlim', [-90 90]*rad);
161
162
163 %Segundo eslabon
164 L(2) = Revolute('d', 0, 'a', datos_robot(2).long_total_eslabon, 'alpha', 0, ...
165 'offset', pi/2, ...
166 'I', [datos_robot(2).inercia(3,3) 0 0; 0 datos_robot(2).inercia(1,1) 0 ; 0 0 ...
167 datos_robot(2).inercia(2,2)] , ...
168 'r', [datos_robot(2).cdm, 0, 0], ...
169 'm', datos_robot(2).masa , ...
170 'qlim', [-90 90]*rad);
171
172 %Tercer eslabon
173 L(3) = Revolute('d', 0, 'a', datos_robot(3).long_total_eslabon, 'alpha', 0, ...
174 'offset', 0, ...
175 'I', [datos_robot(3).inercia(3,3) 0 0; 0 datos_robot(3).inercia(1,1) 0 ; 0 0 ...
176 datos_robot(3).inercia(2,2)] , ...

```

```
176 'r', [datos_robot(3).cdm, 0, 0], ...
177 'm', datos_robot(3).masa, ...
178 'qlim', [-90 90]*rad);
179
180
181 %Definicion del robot
182 R = SerialLink(L,'name', '3DOF');
183 end
```

Listing 4: V3\_masa\_cdm\_inercia

```

1  function [masa,cdm,inercia,radio] = ...
   V3_masa_cdm_inercia(n_eslabon,mot,long_e_c,momento,fuerza, tipo_articulacion)
2
3  %% Ayuda
4
5  % La siguiente funcion sirve para proporcionar el centro de masas, la
6  % inercia y la masa del conjunto de elementos formados por:
7
8  % *n_eslabon = indica el numero de eslabon en la serie de eslabones.
9  % *mot = habra cuatro tipos de motor: grande, mediano, pequeno y pinza en cada uno
10 % de los 'eslabones' (llamaremos eslabon al conjunto de los elementos
11 % mencionados anteriormente). Se introducira un 0 para el motor
12 % pequeno, un 1 para el motor mediano, un 2 para el motor grande y, en
13 % caso de que haya una pinza en el motor, se introducira un 3.
14 % *long_e_c = sera la longitud del modulo pasivo, lo que es la parte
15 % estructural, sin tener en cuenta el modulo del motor ni el elemento
16 % fijo. (es decir, seccion circular + colas de milano)
17 % *momento= es el vector de momentos (3x1) a los que esta sometido el
18 % eslabon en el sistema de coordenadas donde esta cada uno de los servos,
19 % que es la seccion mas solicitada del eslabon
20 % *fuerza= es el vector de fuerzas (3x1) que esta aplicado en el sistema de
21 % coordenadas
22 % tipo_articulacion= distingue entre articulacion axial==0 y articulacion
23 % radial == 1
24
25 % TODOS LOS DATOS ESTAN EN EL SISTEMA INTERNACIONAL (METROS, KG, SEGUNDOS)
26
27 %% Datos de los diferentes tipos de elementos fijos
28
29 % Base
30
31 h_b = 0.009; % altura de la base
32 m_b = 0.01224; % masa de la base
33 cdm_b = 0.00397; % centro de masas de la base
34 i_b = [1.6730e-06 0 0; 0 1.6888e-06 0; 0 0 3.2211e-06]; % matriz de inercia de ...
   la base
35
36 % Radio
37
38 h_r = 0.054; % altura del radio
39 m_r = 0.02217; % masa del radio
40 cdm_r = 0.02637; % centro de masas del radio
41 i_r = [2.0305e-05 0 0; 0 7.4269e-06 0; 0 0 1.4941e-05]; % matriz de inercia del ...
   radio
42
43 if (n_eslabon == 1) % Si es el primer eslabon, elegimos que tiene una base
44     h_e = h_b;
45     m_e = m_b;
46     cdm_ef = cdm_b;
47     i_e = i_b;

```

```

48 else % El resto de eslabones tienen un radio
49     h_e = h_r;
50     m_e = m_r;
51     cdm_ef = cdm_r;
52     i_e = i_r;
53 end
54
55 %% Datos de los diferentes tipos de motores
56
57 % Motor pequeno
58
59 h_motp = 0.040; % altura del modulo del motor pequeno
60 m_motp = 0.039; % masa del modulo del motor pequeno
61 cdm_motp = 0.02058; % centro de masas del modulo del motor pequeno
62 i_motp = [9.6487e-06 0 0; 0 6.5431e-06 0; 0 0 7.7284e-06]; % matriz de inercia ...
    del modulo del motor pequeno
63
64 % Motor mediano
65
66 h_motm = 0.0652; % altura del modulo del motor mediano
67 m_motm = 0.106; % masa del modulo del motor mediano
68 cdm_motm = 0.03256; % centro de masas del modulo del motor mediano
69 i_motm = [4.75e-05 0 0; 0 4e-05 0; 0 0 2.4e-05]; % matriz de inercia del modulo ...
    del motor mediano
70
71 % Motor grande (DYNAMIXEL)
72
73 h_motg = 0.0653; % altura del modulo del motor grande
74 m_motg = 0.09347; % masa del modulo del motor grande
75 cdm_motg = 0.03328; % centro de masas del modulo del motor grande
76 i_motg = [4.4e-05 0 0; 0 4.2058e-05 0; 0 0 2.5718e-05]; % matriz de inercia del ...
    modulo del motor grande
77
78 % Motor con pinza
79
80 m_motpinza = 0.0588; % masa del modulo del motor con pinza
81 cdm_motpinza = 0.02971; % centro de masas del modulo del motor con pinza
82 i_motpinza = [3.0282e-05 0 0; 0 2.2759e-05 0; 0 0 1.3567e-05]; % matriz de ...
    inercia del modulo del motor con pinza
83
84 if (mot == 0)
85     h_mot = h_motp;
86     m_mot = m_motp;
87     cdm_mot_o = cdm_motp;
88     i_mot = i_motp;
89 elseif (mot == 1)
90     h_mot = h_motm;
91     m_mot = m_motm;
92     cdm_mot_o = cdm_motm;
93     i_mot = i_motm;
94 elseif (mot == 2)

```

```

95      h_mot = h_motg;
96      m_mot = m_motg;
97      cdm_mot_o = cdm_motg;
98      i_mot = i_motg;
99  end
100
101 if (n_eslabon==3)
102     m_mot = m_motpinza;
103     cdm_mot_o = cdm_motpinza;
104     i_mot = i_motpinza;
105 end
106
107 %% Datos de las colas de milano
108
109 h_c = 0.005; % espesor de la cola de milano (espesor del cuadrado de 40x40)
110 m_c = 0.00716; % masa de la cola de milano (solo un cuadrado de 40x40)
111 cdm_c = 0.00292; % centro de masas de la cola de milano aislada
112 i_c = [9.4656e-07 0 0; 0 9.4656e-07 0; 0 0 1.8628e-06]; % inercia de la cola de ...
113             milano
114
115 %% Calculo del cdm, inercia y masa del modulo pasivo (viga)
116
117 % Llamamos a la funcion optimiza para optimizar los parametros de la viga
118
119 if long_e_c == 0
120     v_m=0;
121     m_m=0;
122     i_m_eje=0;
123     i_m_otros_ejes=0;
124     cdm_mp=0;
125     radio=[0 0];
126     long_p=0;
127 else
128     [radio,long_p] = V3_optimiza(long_e_c,mot,momento,fuerza, tipo_articulacion);
129
130 cdm_mp = long_p/2; % Centro de masas del modulo pasivo (viga)
131
132 densidad = 1250;
133
134 v_m = long_p*pi*(radio(1)^2-radio(2)^2); % Volumen del modulo pasivo
135 m_m = densidad*v_m; % masa del modulo pasivo
136
137 i_m_eje = 1/2*m_m*(radio(2)^2+radio(1)^2); % Inercia del modulo pasivo en el ...
138             eje que atraviesa el agujero
139 i_m_otros_ejes = 1/4*m_m*(radio(2)^2+radio(1)^2)+1/12*m_m*long_p^3; % Inercia ...
140             en los otros dos ejes (que no atraviesan el agujero)
141
142 end
143 %% Calculo del cdm del conjunto y masa total
144
145

```

```

143 if (n_eslabon ≠ 1)
144     d = 0.009; % distancia desde el centro de giro del radio hasta la base del ...
                  radio
145 else
146     d = 0;
147 end
148
149 cdm_e = cdm_ef-d; % centro de masas del elemento fijo cuando esta en el conjunto
150 cdm_m = cdm_mp+h_e+h_c-d; % centro de masas del modulo pasivo en el conjunto
151 cdm_c1 = cdm_c+h_e-d; % centro de masas de la cola de milano unida al elemento fijo
152 cdm_c2 = cdm_c+h_e+h_c+long_p-d; % centro de masas de la cola de milano unida ...
                  al motor
153 cdm_mot = cdm_mot_o+h_e+2*h_c+long_p-d; % centro de masas del modulo del motor ...
                  en el conjunto
154
155 v_cdm = [cdm_e cdm_c1 cdm_m cdm_c2 cdm_mot]; % vector de centro de masas (para ...
                  realizar el producto escalar)
156 masas = [m_e m_c m_m m_c m_mot]'; % vector de las diferentes masas (para el ...
                  producto escalar)
157
158 masa = sum(masas); % Masa total del conjunto
159 cdm = (v_cdm*masas)/masa; % Centro del masas del conjunto
160
161 %% Calculo de la matriz de inercia del conjunto
162
163 inerciax = i_e(1,1)+m_e*(cdm-cdm_e)^2 + i_c(1,1)+m_c*(cdm-cdm_c1)^2 + ...
              i_m_otros_ejes+m_m*(cdm-cdm_m)^2 + i_c(1,1)+m_c*(cdm-cdm_c2)^2 + ...
              i_mot(1,1)+m_mot*(cdm-cdm_mot)^2;
164 inerciay = i_e(2,2)+m_e*(cdm-cdm_e)^2 + i_c(2,2)+m_c*(cdm-cdm_c1)^2 + ...
              i_m_otros_ejes+m_m*(cdm-cdm_m)^2 + i_c(2,2)+m_c*(cdm-cdm_c2)^2 + ...
              i_mot(2,2)+m_mot*(cdm-cdm_mot)^2;
165 inerciaz = i_e(3,3) + i_c(3,3) + i_m_eje + i_c(3,3)+ i_mot(3,3);
166
167 inercia = [inerciax 0 0; 0 inerciay 0; 0 0 inerciaz];

```

Listing 5: V3\_masa\_cdm\_inercia\_primera\_iteracion

```

1 function [masa,cdm,inercia,radio]= ...
2 % Ayuda
3
4 % La siguiente funcion sirve para proporcionar una primera vez estimada
5 % el centro de masas, la inercia y la masa del conjunto de elementos formados por:
6
7 % *n_eslabon = indica el numero de eslabon en la serie de eslabones.
8 % *mot = habra cuatro tipos de motor: grande, mediano, pequeno y pinza en cada uno
9 % de los 'eslabones' (llamaremos eslabon al conjunto de los elementos
10 % mencionados anteriormente). Se introducira un 0 para el motor
11 % pequeno, un 1 para el motor mediano, un 2 para el motor grande y, en
12 % caso de que haya una pinza en el motor, se introducira un 3.
13 % *long_e_c = sera la longitud del modulo pasivo, lo que es la parte
14 % estructural, sin tener en cuenta el modulo del motor ni el elemento
15 % fijo. (es decir, seccion circular + colas de milano)
16
17 % TODOS LOS DATOS ESTAN EN EL SISTEMA INTERNACIONAL (METROS, KG, SEGUNDOS)
18
19 %% Datos de los diferentes tipos de elementos fijos
20
21 % Base
22
23 h_b = 0.009; % altura de la base
24 m_b = 0.01224; % masa de la base
25 cdm_b = 0.00397; % centro de masas de la base
26 i_b = [1.6730e-06 0 0; 0 1.6888e-06 0; 0 0 3.2211e-06]; % matriz de inercia de ...
27 % la base
28
29 % Radio
30
31 h_r = 0.054; % altura del radio
32 m_r = 0.02217; % masa del radio
33 cdm_r = 0.02637; % centro de masas del radio
34 i_r = [2.0305e-05 0 0; 0 7.4269e-06 0; 0 0 1.4941e-05]; % matriz de inercia del ...
35 % radio
36
37 if (n_eslabon == 1) % Si es el primer eslabon, elegimos que tiene una base
38     h_e = h_b;
39     m_e = m_b;
40     cdm_ef = cdm_b;
41     i_e = i_b;
42 else % El resto de eslabones tienen un radio
43     h_e = h_r;
44     m_e = m_r;
45     cdm_ef = cdm_r;
46     i_e = i_r;
47 end
48
49 %% Datos de los diferentes tipos de motores

```

```

48
49 % Motor pequeno
50
51 h_motp = 0.040; % altura del modulo del motor pequeno
52 m_motp = 0.039; % masa del modulo del motor pequeno
53 cdm_motp = 0.02058; % centro de masas del modulo del motor pequeno
54 i_motp = [9.6487e-06 0 0; 0 6.5431e-06 0; 0 0 7.7284e-06]; % matriz de inercia ...
      del modulo del motor pequeno
55
56 % Motor mediano
57
58 h_motm = 0.0652; % altura del modulo del motor mediano
59 m_motm = 0.106; % masa del modulo del motor mediano
60 cdm_motm = 0.03256; % centro de masas del modulo del motor mediano
61 i_motm = [4.75e-05 0 0; 0 4e-05 0; 0 0 2.4e-05]; % matriz de inercia del modulo ...
      del motor mediano
62
63 % Motor grande (DYNAMIXEL)
64
65 h_motg = 0.0653; % altura del modulo del motor grande
66 m_motg = 0.09347; % masa del modulo del motor grande
67 cdm_motg = 0.03328; % centro de masas del modulo del motor grande
68 i_motg = [4.4e-05 0 0; 0 4.2058e-05 0; 0 0 2.5718e-05]; % matriz de inercia del ...
      modulo del motor grande
69
70 % Motor con pinza
71
72 m_motpinza = 0.0588; % masa del modulo del motor con pinza
73 cdm_motpinza = 0.02971; % centro de masas del modulo del motor con pinza
74 i_motpinza = [3.0282e-05 0 0; 0 2.2759e-05 0; 0 0 1.3567e-05]; % matriz de ...
      inercia del modulo del motor con pinza
75
76 if (mot == 0)
77     h_mot = h_motp;
78     m_mot = m_motp;
79     cdm_mot_o = cdm_motp;
80     i_mot = i_motp;
81 elseif (mot == 1)
82     h_mot = h_motm;
83     m_mot = m_motm;
84     cdm_mot_o = cdm_motm;
85     i_mot = i_motm;
86 elseif (mot == 2)
87     h_mot = h_motg;
88     m_mot = m_motg;
89     cdm_mot_o = cdm_motg;
90     i_mot = i_motg;
91 end
92
93 if(n_eslabon==3)
94     m_mot = m_motpinza;

```

```

95      cdm_mot_o = cdm_motpinza;
96      i_mot = i_motpinza;
97  end
98
99  %% Datos de las colas de milano
100
101 h_c = 0.005; % espesor de la cola de milano (espesor del cuadrado de 40x40)
102 m_c = 0.00716; % masa de la cola de milano (solo un cuadrado de 40x40)
103 cdm_c = 0.00292; % centro de masas de la cola de milano aislada
104 i_c = [9.4656e-07 0 0; 0 9.4656e-07 0; 0 0 1.8628e-06]; % inercia de la cola de ...
     milano
105
106 %% Calculo del cdm, inercia y masa del modulo pasivo (viga)
107
108 % Llamamos a la funcion optimiza para optimizar los parametros de la viga
109
110 if long_e_c == 0
111     v_m=0;
112     m_m=0;
113     i_m_eje=0;
114     i_m_otros_ejes=0;
115     cdm_mp=0;
116     radio=[0 0];
117     long_p=0;
118
119 else
120     [radio,long_p] = V3_optimiza(long_e_c,mot,[1.21 1.21 1.21]',[0 0 ...
     0]',tipo_articulacion);
121
122 cdm_mp = long_p/2; % Centro de masas del modulo pasivo (viga)
123
124 densidad = 1250; % Kg/m3
125
126 v_m = long_p*pi*(radio(1)^2-radio(2)^2); % Volumen del modulo pasivo
127 m_m = densidad*v_m; % masa del modulo pasivo
128
129 i_m_eje = 1/2*m_m*(radio(2)^2+radio(1)^2); % Inercia del modulo pasivo en el ...
     eje que atraviesa el agujero
130 i_m_otros_ejes = 1/4*m_m*(radio(2)^2+radio(1)^2)+1/12*m_m*long_p^3; % Inercia ...
     en los otros dos ejes (que no atraviesan el agujero)
131
132 end
133 %% Calculo del cdm del conjunto y masa total
134
135 if (n_eslabon ≠ 1)
136     d = 0.009; % distancia desde el centro de giro del radio hasta la base del ...
     radio
137 else
138     d = 0;
139 end
140

```

```

141 cdm_e = cdm_ef-d; % centro de masas del elemento fijo cuando esta en el conjunto
142 cdm_m = cdm_mp+h_e+h_c-d; % centro de masas del modulo pasivo en el conjunto
143 cdm_c1 = cdm_c+h_e-d; % centro de masas de la cola de milano unida al elemento fijo
144 cdm_c2 = cdm_c+h_e+h_c+long_p-d; % centro de masas de la cola de milano unida ...
     al motor
145 cdm_mot = cdm_mot_o+h_e+2*h_c+long_p-d; % centro de masas del modulo del motor ...
     en el conjunto
146
147 v_cdm = [cdm_e cdm_c1 cdm_m cdm_c2 cdm_mot]; % vector de centro de masas (para ...
     realizar el producto escalar)
148 masas = [m_e m_c m_m m_c m_mot]'; % vector de las diferentes masas (para el ...
     producto escalar)
149
150 masa = sum(masas); % Masa total del conjunto
151 cdm = (v_cdm*masas)/masa; % Centro del masas del conjunto
152
153 %% Calculo de la matriz de inercia del conjunto
154
155 inerciax = i_e(1,1)+m_e*(cdm-cdm_e)^2 + i_c(1,1)+m_c*(cdm-cdm_c1)^2 + ...
     i_m Otros_ejes+m_m*(cdm-cdm_m)^2 + i_c(1,1)+m_c*(cdm-cdm_c2)^2 + ...
     i_mot(1,1)+m_mot*(cdm-cdm_mot)^2;
156 inerciay = i_e(2,2)+m_e*(cdm-cdm_e)^2 + i_c(2,2)+m_c*(cdm-cdm_c1)^2 + ...
     i_m Otros_ejes+m_m*(cdm-cdm_m)^2 + i_c(2,2)+m_c*(cdm-cdm_c2)^2 + ...
     i_mot(2,2)+m_mot*(cdm-cdm_mot)^2;
157 inerciaz = i_e(3,3) + i_c(3,3) + i_m_eje + i_c(3,3)+ i_mot(3,3);
158
159 inercia = [inerciax 0 0; 0 inerciay 0; 0 0 inerciaz];

```

Listing 6: V3\_optimiza

```

1  function [radio,long_p] = ...
   V3_optimiza(long_e_c,momento_max,momento,fuerzas, tipo_articulacion)
2  %% ALGORITMO
3
4  % Se da como 1er parametro de entrada la longitud del eslabon de la viga
5  % junto con las colas de milano de los extremos.
6  espesor_colas = 0.005; % (m) espesor de las colas de milano
7  L = long_e_c-2*espesor_colas; % (m) Longitud de la viga (sin las colas) que se
8  % usa para el disenzo
9
10 % El 2do parametro de entrada es el momento maximo que es entregado por el
11 % servo que acciona dicho brazo, que es escogido entre 3 servos
12 % m_motor sera el momento maximo que da el motor de la seccion mas solicitada ...
   en Nm
13 if momento_max==0
14     m_motor=0.13; % (Nm) Da el valor del momento maximo del servo pequeno
15     momento(3,1)=m_motor;
16 end
17 if momento_max==1
18     m_motor=0.345; % (Nm) Da el valor del momento maximo del servo mediano
19     momento(3,1)=m_motor;
20 end
21 if momento_max==2
22     m_motor=1.21; % (Nm) Da el valor del momento maximo del servo dinamixel
23     momento(3,1)=m_motor;
24 end
25
26
27 % El tipo de articulacion viene a decir en que eje hace la rotacion el
28 % servo:
29 % Articulacion tipo rotacion radial = la rotacion se hace en el eje Z,
30 % estando la siguiente articulacion en el mismo eje en que rota (eje Z)
31 % Articulacion tipo rotacion axial = la rotacion se hace en el eje Z,
32 % estando la siguiente articulacion en otro eje distinto al que rota,
33 % usualmente el eje X.
34
35 if (tipo_articulacion==1) % 1 es el tipo de articulacion Axial
36     T=momento(1,1);
37     F=fuerzas(1,1);
38     momento_flector=(momento(2,1)^2+momento(3,1)^2)^0.5;
39 else % Seria tipo articulacion 0, la articulacion Radial
40     T=momento(3,1);
41     F=fuerzas(3,1);
42     momento_flector=(momento(1,1)^2+momento(2,1)^2)^0.5;
43 end
44
45 % Se fija ahora valores como el radio externo maximo del eslabon y el
46 % espesor de la viga minimo o la diferencia de radios minima R2 y R1, el
47 % momento de inercia minimo que debe tener la viga y la flecha v para la
48 % cual se ha disenado la viga

```

```

49 r2max = 0.018; % (m) Radio externo maximo del eslabon
50 e = 0.0025; % (m) Espesor minimo que se desea tener en el eslabon
51 E = 2.8e9; % Modulo de Young del PLA (Pa)
52
53
54 % Se definen los parametros de Tension de rotura UTS, factor de seguridad SF
55 % y el concentrador de tensiones para tension normal (tension tangencial
56 % seria menor pero se usa la misma en principio)
57 UTS=50*10^6; % (Pa) Es la Ultimate Tensile Stress o tension de rotura
58 SF=1.5; % Es el factor de seguridad para el que se va a disenar la viga
59 Kt=1.7; % Es el factor de concentracion de tension que se debe por la geometria ...
en el inicio de la viga
60 % con D/d=1.6 aprox y r/d=0.1, con lo que Kt lo cogemos como
61 % concentrador de tensiones tanto para tension tangencial como para
62 % tension normal
63
64 z=0; % Condicion que hace que el bucle se repita
65 v=L/500; % Definicion del desplazamiento vertical que sufre con el momento del ...
servo
66 zz=1; % Se usa si la tension de Von Mises provoca rotura, y con zz se usa para ...
disminuir el desplazamiento vertical
67 while(z==0)
68 r2=r2max;
69 I=momento_flector*L^2/(3*E*v); % (m^4) Momento de inercia buscado y minimo ...
necesario
70 Ko = 3*E*I/(L^3); % Rigidex (N/m)
71
72 % % Calculamos la geometria optima
73 Ko_t = 4*L^3*Ko/(3*pi*E);
74
75 r1 = (r2^4-Ko_t)^(1/4); % (m) Valor del radio interior optimo
76 % Si r1 cumple espesor (r2-r1)>e, nos quedamos con este radio interior
77
78 if (r2-r1 < e) % Evaluamos opcion de que el espesor sea menor que el impresible
79 % Calculamos la geometria optima, que en funcion del radio interior y
80 % el espesor e=0.0025 obtenemos la seccion con la inercia deseada
81 % encontrando las raices de la funcion siguiente en funcion del radio
82 % interior
83 func=@(r) I-pi*(r^3/100 + (3*r^2)/80000 + r/16000000 + 1/25600000000)/4;
84 r1=fzero(func, [-0.0025 0.0165]); % (m)
85
86 r2=r1+0.0025; % (m) Le damos el valor al radio menor
87 if (r1≤0)
88 r1=0.001;
89 end
90 end
91 %
92 if (r1 ≤ 0.001) % Ponemos un radio minimo (esto seria el caso en el que el ...
modulo seria lo mas chico posible)
93 r1 = 0.001;
94 r2 = 0.0035;

```

```
95     end
96
97     % Evaluamos la Tension Von Mises en la sección más solicitada
98     % (empotramiento), teniendo en cuenta la tensión normal y la tensión
99     % tangencial (causada por retorcer brazo)
100    tension_normal=momento_flector*r2/I+abs(F)/(pi*(r2^2-r1^2));
101    Itorsion=pi*(r2^4-r1^4)/2;
102    % tension_tangencial=16*T/(pi*(2*r2)^3*(1-(r1/r2)^4));
103    tension_tangencial=T*r2/Itorsion;
104    tension_VonMises=SF*((Kt*tension_normal)^2+3*(Kt*tension_tangencial)^2)^0.5;
105    if (UTS>tension_VonMises)
106        z=1; % Si la tensión de VM es menor que UTS, no rompe el brazo y nos ...
107        % vale las dimensiones para
108        % diseñar el brazo robótico
109    else
110        zz=zz+1;
111        v=L/(500+zz); % El brazo rompe, y disminuimos el desplazamiento de ...
112        % diseño para que
113        % la estructura sea más rígida y no rompa
114    end
115 end
116
117 %Calculo del redondeo
118 return
```

Listing 7: V3\_calcular\_pares\_3DOF\_rne

```

1  function [ torque_dinamico, wbase, momentos, fuerzas ] = ...
   V3_calcular_pares_3DOF_rne( R,m, datos_robot,servos )
2  %vel, acel1, acel2
3  %Funcion para calcular los pares necesario en el robot R para levantar una
4  %masa m. Para calcular los pares se evalua el robot en la peor posicion articular
5  %posible
6  %Entradas:
7  %R: robot
8  %m: masa a levantar
9  %Salidas:
10 %torque_total: vector de 1x3 en el que se especifica el par necesario en
11 %cada servo
12
13 Q=[0 -pi/2 0]; %Posicion mas desfavorable
14
15 %VELOCIDADES
16 rads_dinamixel= 6.178465545; %59 rpm
17 rads_medio= 6.974335683; %66.6 rpm
18 rads_pequeno= 8.7231555915; %83.3 rpm
19 %Para pasar a m/s %r * RPM * 0.10472
20
21 vel_max=[];
22
23 for n=1:3
24     if servos(n)==0
25         vel_max(1,n)=rads_pequeno;
26     elseif servos(n)==1
27         vel_max(1,n)=rads_medio;
28     elseif servos(n)==2
29         vel_max(1,n)=rads_dinamixel;
30     end
31 end
32
33 acel=[1 1 1]*5;
34
35 %acel = (R.accel(Q, vel, torque_max))';
36
37 [torque_dinamico,wbase,momentos,fuerzas]=R.V3_rne_dh_modificado(Q, vel_max/2, ...
   acel, [0 0 -9.81]',m);
38
39 %Cambio de unidades
40 torque_dinamico=(1/0.0980665)*torque_dinamico; %Para pasar de N*m a kg*cm
41
42 end

```

Listing 8: V3\_calcular\_pares\_3DOF\_estatico

```
1 function [ torque_estatico, wbase,momentos,fuerzas ] = ...
2 %Funcion para calcular los pares necesario en el robot R para levantar una
3 %masa m. Para calcular los pares se evalua el robot en la peor posicion articular
4 %posible
5 %Entradas:
6 %R: robot
7 %m: masa a levantar
8 %Salidas:
9 %torque_total: vector de 1x3 en el que se especifica el par necesario en
10 %cada servo
11
12 %CALCULAR EN ESTATICO
13
14 Q=[0 -pi/2 0]; %Posicion mas desfavorable
15
16 %Calculo del par, con las entradas de velocidad y aceleracion a cero
17 [torque_estatico,wbase,momentos,fuerzas]=R.V3_rne_dh_modificado(Q, [0 0 0], [0 ...
18 0 0], [0 0 -9.81]', m);
19 %Cambio de unidades
20 torque_estatico=(1/0.0980665)*torque_estatico; %Para pasar de N*m a kg*cm
21
22
23 end
```

Listing 9: V3\_elegir\_servos\_3DOF

```
1 function [ servos_finales ] = V3_elegir_servos_3DOF( torque, servos )
2 % Elegir tipo de servo en función del par requerido
3 % Entrada:
4 %torque_total: vector de 1x3 en el que se especifica el par necesario en
5 %cada servo
6 %Salida:
7 %servos_finales: vector de 1x3 en el que se especifica el tipo de módulo a
8 %utilizar, donde: servo_pequeno=0 servo_medio=1 dinamixel=2
9
10 %torque maximo para cada uno de los pares en kg-cm
11 torque_maximo_servo_pequeno=1.3;
12 torque_maximo_servo_medio = 4; %servo medio HK15268A
13 torque_maximo_dinamixel = 12.29;
14
15 servos_finales=servos;
16
17 %En el eslabón 1 siempre tendremos un servo medio (en la base)
18 servos_finales(1)=1;
19
20 torque_max=[0 0 0];
21
22 for i=1:3
23     if servos(1,i)==1
24         torque_max(i)=torque_maximo_servo_medio;
25     elseif servos(i)==2
26         torque_max(i)=torque_maximo_dinamixel;
27     end
28 end
29
30 for i=1:3
31     if abs(torque(i)) > torque_maximo_dinamixel
32         error('Se necesita un par de %f N·m en la articulación número %d . No se ...
33             puede conseguir un par tan grande con los servos disponibles',torque(i),i);
34     end
35
36 for i=1:3
37     if abs(torque(i)) > torque_max(i)
38         servos_finales(i)=servos_finales(i)+1;
39     end
40 end
```

Listing 10: V3\_rne\_dh\_modificado

```

1  %$SERIALLINK.RNE_DH Compute inverse dynamics via recursive Newton-Euler formulation
2  %
3  % Recursive Newton-Euler for standard Denavit-Hartenberg notation. Is invoked by
4  % R.RNE().
5  %
6  % See also SERIALLINK.RNE.
7  %
8  %
9  % verified against MAPLE code, which is verified by examples
10 %
11 %
12 % Ryan Steindl based on Robotics Toolbox for MATLAB (v6 and v9)
13 %
14 % Copyright (C) 1993-2011, by Peter I. Corke
15 %
16 % This file is part of The Robotics Toolbox for MATLAB (RTB).
17 %
18 % RTB is free software: you can redistribute it and/or modify
19 % it under the terms of the GNU Lesser General Public License as published by
20 % the Free Software Foundation, either version 3 of the License, or
21 % (at your option) any later version.
22 %
23 % RTB is distributed in the hope that it will be useful,
24 % but WITHOUT ANY WARRANTY; without even the implied warranty of
25 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
26 % GNU Lesser General Public License for more details.
27 %
28 % You should have received a copy of the GNU Lesser General Public License
29 % along with RTB. If not, see <http://www.gnu.org/licenses/>.
30 %
31 % http://www.petercorke.com
32 %
33 function [tau,wbase,momentos,fuerzas] = V3_rne_dh_modificado(robot, posicion, ...
    velocidad, aceleracion, a4, masa)
34 %
35 %
36 z0 = [0;0;1];
37 grav = robot.gravity;    % default gravity from the object
38 fext = zeros(6, 1);
39 n = robot.n;
40 %
41 % check that robot object has dynamic parameters for each link
42 for j=1:n,
43     link = robot.links(j);
44     if isempty(link.r) || isempty(link.I) || isempty(link.m)
45         error('dynamic parameters (m, r, I) not set in link %d', j);
46     end
47 end
48 %
49 % Set debug to:

```

```

50      % 0 no messages
51      % 1 display results of forward and backward recursions
52      % 2 display print R and p*
53
54      debug = 0;
55
56      if numcols(posicion) == 3*n,
57          Q = posicion(:,1:n);
58          Qd = posicion(:,n+1:2*n);
59          Qdd = posicion(:,2*n+1:3*n);
60          np = numrows(Q);
61          if nargin >= 3,
62              grav = velocidad(:);
63          end
64          if nargin == 4,
65              fext = aceleracion;
66          end
67      else
68          np = numrows(posicion);
69          Q = posicion;
70          Qd = velocidad;
71          Qdd = aceleracion;
72          if numcols(posicion) != n | numcols(Qd) != n | numcols(Qdd) != n | ...
73              numrows(Qd) != np | numrows(Qdd) != np,
74              error('bad data');
75          end
76          if nargin >= 5,
77              grav = robot.gravity
78          end
79          if nargin == 6,
80              fext(1:3,1) = masa*grav;
81          end
82      end
83
84      tau = zeros(np,n);
85      momentos=zeros(3,n);
86      fuerzas=zeros(3,n);
87      matriz_rotacion=zeros(n,3,3);
88      base_efector=eye(3,3);
89
90      for p=1:np,
91          q = Q(p,:)';
92          qd = Qd(p,:)';
93          qdd = Qdd(p,:)';
94
95          Fm = [];
96          Nm = [];
97          pstarm = [];
98          Rm = [];
99          w = zeros(3,1);
100         wd = zeros(3,1);

```

```

101      v = zeros(3,1);
102      vd = grav(:);
103
104      %
105      % init some variables, compute the link rotation matrices
106      %
107      for j=1:n,
108          link = robot.links(j);
109          Tj = link.A(q(j));
110          if link.RP == 'R',
111              d = link.d;
112          else
113              d = q(j);
114          end
115          alpha = link.alpha;
116          pstar = [link.a; d*sin(alpha); d*cos(alpha)];
117          if j == 1,
118              pstar = t2r(robot.base) * pstar;
119              Tj = robot.base * Tj;
120          end
121          pstar(:,j) = pstar;
122          Rm{j} = t2r(Tj);
123          matriz_rotacion(:,:,j)=Rm{j};
124          base_efector=base_efector*matriz_rotacion(:,:,j);
125          if debug>1,
126              Rm{j}
127              Pstarm(:,j)'
128          end
129      end
130
131      %
132      % the forward recursion
133      %
134
135      for j=1:n,
136          link = robot.links(j);
137
138          Rt = Rm{j}';      % transpose!!
139          pstar = pstar(:,j);
140          r = link.r;
141          %
142          % statement order is important here
143          %
144          if link.RP == 'R',
145              % revolute axis
146              wd = Rt*(wd + z0*qdd(j) + ...
147                          cross(w,z0*qd(j)));
148              w = Rt*(w + z0*qd(j));
149              %v = cross(w,pstar) + Rt*v;
150              vd = cross(wd,pstar) + ...
151                  cross(w, cross(w,pstar)) +Rt*vd;

```

```

152
153     else
154         % prismatic axis
155         w = Rt*w;
156         wd = Rt*wd;
157         vd = Rt*(z0*qdd(j)+vd) + ...
158             cross(wd,pstar) + ...
159             2*cross(w,Rt*z0*qd(j)) +...
160             cross(w, cross(w,pstar));
161     end
162
163     %whos
164     vhat = cross(wd,r') + ...
165         cross(w,cross(w,r')) + vd;
166     F = link.m*vhat;
167     N = link.I*wd + cross(w,link.I*w);
168     Fm = [Fm F];
169     Nm = [Nm N];
170
171     if debug,
172         fprintf('w: '); fprintf('%.3f ', w)
173         fprintf('\nwd: '); fprintf('%.3f ', wd)
174         fprintf('\nvd: '); fprintf('%.3f ', vd)
175         fprintf('\nvdbar: '); fprintf('%.3f ', vhat)
176         fprintf('\n');
177     end
178 end
179
180 %
181 % the backward recursion
182 %
183
184 fext = fext(:);
185 f=fext(1:3);
186 f = base_efector'*fext(1:3);           % force/moment on end of arm
187 nn = fext(4:6);
188
189 for j=n:-1:1,
190     link = robot.links(j);
191     pstar = pstarm(:,j);
192
193 %
194 % order of these statements is important, since both
195 % nn and f are functions of previous f.
196 %
197 if j == n,
198     R = eye(3,3);
199 else
200     R = Rm{j+1};
201 end
202 r = link.r;

```

```
203     nn = R*(nn + cross(R'*pstar,f)) + ...
204         cross(pstar+r',Fm(:,j)) + ...
205         Nm(:,j);
206     f = R*f + Fm(:,j);
207     if debug,
208         fprintf('f: '); fprintf('.3f ', f)
209         fprintf('\nn: '); fprintf('.3f ', nn)
210         fprintf('\n');
211     end
212
213     R = Rm{j};
214     if link.RP == 'R',
215         % revolute
216         tau(p,j) = nn)*(R'*z0) + ...
217             link.G^2 * link.Jm*qdd(j) + ...
218             abs(link.G) * friction(link, qd(j));
219     else
220         % prismatic
221         tau(p,j) = f)*(R'*z0) + ...
222             link.G^2 * link.Jm*qdd(j) + ...
223             abs(link.G) * friction(link, qd(j));
224     end
225     momentos(:,j)=R*nn;
226     fuerzas(:,j)=R*f;
227 end
228 % this last bit needs work/testing
229 R = Rm{1};
230 nn = R*(nn);
231 f = R*f;
232 wbase = [f'; nn'];
233 momentos;
234 fuerzas;
235 matriz_rotacion;
236 end
237 end
```

## Referencias

- [1] RoKiSim. <http://www.parallelmic.org/RoKiSim.html>. Published by Control and Robotics Lab École de technologie supérieure (Montreal, Canada).
- [2] P. Corke. Robotics toolbox. [http://www.petercorke.com/Robotics\\_Toolbox.html](http://www.petercorke.com/Robotics_Toolbox.html).
- [3] P. Corke. *Robotics Toolbox for MATLAB Release 9*, Feb. 2015.
- [4] A. Gil. ARTE. [http://arvc.umh.es/arte/index\\_en.html](http://arvc.umh.es/arte/index_en.html).
- [5] R. Pazmiño. *Prácticas de robótica utilizando MatLab*. Universidad Miguel Hernández. Departamento de Ingeniería, 2000.
- [6] H. F. Quintero, G. Calle-Trujillo, and A. Díaz. Selección de un servomotor y transmisión por el método de las potencias transitorias. *Scientia et Technica Año XII, No 30*, Vol.1, 2016.
- [7] J. O. Yugat. Selection methodology of mechanical drive system servomotor and transmission through transient power analysis. *Revista Técnica de la Facultad de Ingeniería Universidad del Zulia*, 2010.