

---

Desarrollo de herramientas en software para el manejo, monitoreo y programación de la nueva versión del humanoide Robonova

---

Stefano Papadopolo Cruz



UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería



Desarrollo de herramientas en software para el manejo,  
monitoreo y programación de la nueva versión del humanoide  
**Robonova**

Trabajo de graduación presentado por Stefano Papadopolo Cruz para  
optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

Vo.Bo.:

(f) \_\_\_\_\_  
Ing. Miguel Zea

Tribunal Examinador:

(f) \_\_\_\_\_  
Ing. Miguel Zea

(f) \_\_\_\_\_

(f) \_\_\_\_\_

Fecha de aprobación: Guatemala, de de .

---

## Prefacio

---

Mi curiosidad desde que era niño de cómo funciona las cosas, constante necesidad de seguir aprendiendo y la mentalidad de "que tan difícil puede ser.<sup>en</sup> todas las áreas de la vida me han llevado al deseo de ejercer como ingeniero. El crecimiento y aprendizaje que he tenido en esta carrera ha sido una oportunidad como ninguna otra de la cual he aprendido bastante y ha incrementado estas cualidades inquisitivas en mi vida.

El estudiar una licenciatura en ingeniería mecatrónica por cinco años, aprender acerca de diseños mecánicos, eléctricos y de control, además de programación en una amplia gama de lenguajes culmina aquí: en la robótica. Esta tesis cubre todas las áreas de lo aprendido en la carrera. Desde la revisión de diseños CAD y su ensamblaje y la creación y programación de circuitos simples con un microcontrolador, hasta la creación de un programa para el control y simulación de un robot humanoide.

En esta sección quiero aprovechar para agradecer a mi asesor el Ing. Miguel Zea y a todos mis compañeros graduandos. Sin su apoyo al momento de compartir ideas, discutir nuestras distintas opiniones y aprender y crecer juntos estos últimos años, este trabajo no hubiera sido posible. También quiero agradecer a mi pareja ya que de no ser por su constante presión, apoyo y ánimo, escribir esta tesis hubiera tomado una cantidad vergonzosa de tiempo.

---

## Índice

---

<b>Prefacio</b>	<b>III</b>
<b>Lista de figuras</b>	<b>VII</b>
<b>Lista de cuadros</b>	<b>VIII</b>
<b>Resumen</b>	<b>IX</b>
<b>Abstract</b>	<b>X</b>
<b>1. Introducción</b>	<b>1</b>
<b>2. Antecedentes</b>	<b>3</b>
2.1. Programación en RoboBASIC . . . . .	3
2.2. Programación de robots bailarines . . . . .	5
2.3. Atlas de Boston Dynamics . . . . .	6
2.4. Chorégraphe de Aldebaran Robotics . . . . .	7
<b>3. Justificación</b>	<b>10</b>
<b>4. Objetivos</b>	<b>11</b>
<b>5. Alcance</b>	<b>12</b>
<b>6. Marco teórico</b>	<b>13</b>
6.1. Biomecánica del movimiento humano . . . . .	13
6.1.1. Ciclo de Gait . . . . .	14
6.2. Cinemática de robots humanoides . . . . .	15
6.3. Robonova-1 . . . . .	16
6.3.1. Versión original . . . . .	16
6.3.2. Nueva versión (diseñada por Fernando José Caceres) . . . . .	19
6.4. PyBullet . . . . .	20
6.5. Robotics Toolbox de Peter Corke . . . . .	21
6.6. Formato JSON . . . . .	21

6.7. Formato URDF . . . . .	23
6.8. Onshape-to-robot . . . . .	23
6.9. PyQt . . . . .	23
6.10. Protocolo de comunicación TCP/IP . . . . .	25
6.11. ESP32 DevKit V1 . . . . .	25
<b>7. Generación Automática del URDF de la nueva versión del Robonova</b>	<b>27</b>
7.1. Diseño de ensamblaje . . . . .	27
7.1.1. Metodología . . . . .	27
7.1.2. Resultados . . . . .	28
7.2. Uso de onshape-to-robot . . . . .	29
7.2.1. Metodología . . . . .	29
7.2.2. Resultados . . . . .	30
7.3. Discusión . . . . .	30
<b>8. Programa en python para simulación y control de Robonova</b>	<b>31</b>
8.1. Primera versión del programa . . . . .	32
8.1.1. Metodología . . . . .	32
8.1.2. Resultados . . . . .	39
8.2. Segunda Versión del programa . . . . .	41
8.2.1. Metodología . . . . .	41
8.2.2. Resultados . . . . .	46
8.3. Discusión . . . . .	46
<b>9. Control en tiempo real de Robonova</b>	<b>47</b>
9.1. Metodología . . . . .	47
9.2. Resultados . . . . .	47
9.2.1. Programación final de servidor y controlador de servomotores en Arduino	47
9.2.2. Programación final de cliente en python . . . . .	49
9.3. Discusión . . . . .	50
<b>10. Conclusiones</b>	<b>51</b>
<b>11. Recomendaciones</b>	<b>52</b>
<b>12. Bibliografía</b>	<b>53</b>
<b>13. Anexos</b>	<b>56</b>
<b>14. Glosario</b>	<b>57</b>

---

## Lista de figuras

---

1.	Captura de pantalla de la interfaz Catch and Play de RoboBASIC . . . . .	4
2.	Captura de pantalla de la interfaz RoboScript de RoboBASIC . . . . .	4
3.	Captura de pantalla de la interfaz RoboReomocon de RoboBASIC . . . . .	5
4.	Comparación entre RoboBASIC y el ecosistema creado por investigadores de Drexel University . . . . .	6
5.	Robot Humanoide Atlas de Boston Dynamics . . . . .	7
6.	Robot Humanoide Aldebaran Robotics . . . . .	8
7.	Interfaz de usuario de Choregraphe . . . . .	9
8.	Modelo de juntas y eslabones de un sistema biomecánico [7] . . . . .	14
9.	Ciclo de Gait [8] . . . . .	15
10.	Representación cinemática de un robot humanoide [12] . . . . .	16
11.	Diseño original del Robonova-1 [14] . . . . .	17
12.	Numeración de los servomotores en la versión original del Robonova-1 [13] .	18
13.	Planos de diseño mecánico de los servomotores HSR-8498HB . . . . .	19
14.	Nueva versión del robot humanoide Robonova . . . . .	20
15.	Ejemplo de un objeto en formato JSON [21] . . . . .	22
16.	Ejemplo de un <i>array</i> en formato JSON [21] . . . . .	22
17.	Ejemplo de la sintaxis de un archivo URDF . . . . .	24
18.	Capas del protocolo de comunicación TCP/IP . . . . .	25
19.	Mapa de pines del ESP32 [28] . . . . .	26
20.	Configuración de junta en Onshape . . . . .	28
21.	Ensamblaje final de la nueva versión del robonova . . . . .	29
22.	Folder con piezas y URDF del Robonova . . . . .	30
23.	Primera versión de la interfaz de usuario del programa de control del Robonova	34
24.	Mundo Inicial en pyBullet . . . . .	36
25.	Modelo de Robonova en pyBullet . . . . .	37
26.	Simulación Robonova con posiciones de los servomotores cambiada en pyBullet	38
27.	Interfaz luego de cargar posiciones a una rutina . . . . .	39
28.	Captura de pantalla del programa y la simulación . . . . .	40
29.	Captura de pantalla de la simulación corriendo la rutina <i>walking</i> . . . . .	40

30.	Segunda versión de la interfaz de usuario del programa de control del Robonova	41
31.	Interfaz de usuario al crear una nueva rutina . . . . .	43
32.	Espacio para imágenes de la pose actual de la rutina . . . . .	45
33.	Conexión TCP entre servidor y cliente . . . . .	48
34.	Prototipo de piernas del Robonova siendo controlado programa . . . . .	50

---

## Lista de cuadros

---

- |    |   |    |
|----|---|----|
| 1. | Características del controlador MR-C3024 [15] . . . . . | 18 |
| 2. | Características del controlador ESP32 [28] . . . . .    | 26 |

---

## Resumen

---

Esta tesis plantea el diseño de una herramienta de software para el control, programación y monitoreo de la nueva versión del robot humanoide Robonova-1. Se diseñó un programa inspirado en aplicaciones de control de robots humanoides ya en el mercado (roboBASIC y Choreography). Para ello el programa, escrito en Python, utilizó PyQt5 para la programación de la interfaz gráfica y pyBullet para simular los movimientos del robot. En cuanto a la conexión con el robot físico, se utilizó el protocolo comunicación TCP/IP para conectar el cliente (programa) al servidor (robot). El resultado final es una aplicación que permite controlar los servomotores del Robonova de forma inalámbrica, crear rutinas en base a estos movimientos de los servomotores (además de contar con rutinas predefinidas) y visualizar los cambios de pose del Robonova en una simulación dentro de la aplicación. Además de esto, se obtuvo una manera fácil y confiable de generar URDFs de cualquier diseño de robot deseado.

---

## Abstract

---

This undergraduate thesis proposes the design of a software application for the control, programming and monitoring of the new version of the humanoid robot Robonova-1. The design of the program was inspired by other robot control applications already on the market (roboBASIC and Choregraphe). For that the program, written in Python, used the PyQt5 library to design and implement the graphic user interface and the pyBullet library to simulate and visualize the robot's movements on the application. The connection to the physical robot was done using the TCP/IP protocol where the client (application) connects to a server (robot). The final result is an application that allows the user to control the Robonova's servomotors wirelessly, create routines based on the positions of the servomotors (along with pre-made routines) and, visualize the changes on the pose of the Robonova in a simulation embedded on the program. On top of this, an easy reliable way to generate the URDF of any robotic design was achieved.

# CAPÍTULO 1

---

## Introducción

---

Este trabajo graduación se basa en la revitalización de los robots Robonova-1 de la Universidad del Valle de Guatemala. El objetivo general de este proyecto es diseñar una herramienta de software para monitorear y programar los Robonova de forma gráfica. El proyecto consiste en el diseño del software y coreografías básicas, la comunicación con el controlador del robot y el monitoreo en tiempo real del mismo.

El proyecto consiste de 3 distintos objetivos: diseño de la interfaz gráfica, diseño de rutinas y conexión en tiempo real al Robonova. Estos tres objetivos se presentarán en dos capítulos separados. El primero definiendo la programación, diseño y uso de la interfaz gráfica para el control de la simulación y creación de rutinas y el segundo será acerca de la conexión y el control del robot. La interfaz gráfica consistirá de dos ventanas. Una para visualizar la simulación y otra para controlar el controlarla y crear las rutinas, además de habilitar y deshabilitar la conexión al Robonova. La conexión al Robonova en tiempo real correrá en paralelo a la simulación, permitiendo observar los movimientos del robot tanto en el programa como en el mundo real.

Siguiente a esta introducción, en el capítulo 2, tendremos una sección de antecedentes. En esta sección se encuentran programas de control similares e información relacionada el control de un robot humanoide que se usaron como inspiración para el diseño del programa. En el capítulo 3, justificación, se expande en las razones por las cuales se está llevando a cabo este trabajo de graduación. Siguiendo en el capítulo 4, objetivos, se detallan específicamente las características con las que tiene que cumplir el programa para considerar su diseño exitoso. En el capítulo 5 se define el alcance del proyecto reduciendo la extensión del proyecto para mantener su enfoque en los objetivos establecidos. En el capítulo 6, marco teórico, se presenta la información que se necesitó para llevar a cabo el proyecto. Esta abarca desde las librerías de python y las características de los microcontroladores utilizados hasta investigación acerca de la biomecánica del movimiento humano necesaria para emular la caminata en las rutinas del robot. En el capítulo 6 se presenta como generar un URDF utilizando librerías de python. En el capítulo 7 se expande en el diseño de la aplicación de control del robot mostrando la metodología de ingeniería y diseño que se llevó a cabo para construirlo y el resultado final

obtenido. De forma similar, el capítulo 8 presenta lo mismo para el diseño del programa de control dentro del *cerebro* del Robonova. En el capítulo 9 se presentan las conclusiones obtenidas gracias a esta investigación y el capítulo 10, las recomendaciones para ampliar la misma.

# CAPÍTULO 2

---

## Antecedentes

---

Este proyecto de graduación se trabajará con los robos humanoides Robonova sin embargo, debido a que no se han trabajado proyectos con estos robots dentro de la Universidad del Valle de Guatemala, los antecedentes a continuación serán obtenidos de fuentes externas a la institución. Estos antecedentes incluyen: la programación original en RobotBASIC de los Robonova como fue diseñado por David Buckley, un proyecto de robots bailarines autónomos que utilizó a los Robonova como prototipo y las características de otros robots humanoides como NAO de Aldebaran Robotics y Atlas de Boston Dynamics.

### 2.1. Programación en RoboBASIC

Originalmente, los robots Robonova eran programados a través del IDE RoboBASIC. Como su nombre lo indica, la programación de estos robots se llevaba a cabo en el lenguaje de programación BASIC [1]. Para ello contaba con comandos generales de programación en BASIC además de comandos específicos para facilitar la programación de los Robonova, como lo es el control de los servomotores y la asignación de grupos de motores estipulada en el manual de RoboBASIC. Además de estos comandos para el control principal del robot, el programa también contaba con comandos para funciones adicionales como lo son comandos de sonido para identificar y generar diversas frecuencias y comandos de control para LCD.

Además de la programación de alto nivel en la que se trabaja, el software de RoboBASIC también cuenta con varias interfaces para facilitar aún más el control de los robots. La primera, llamada “Catch-and-Play”, permitía controlar cada servomotor de los Robonova en tiempo real a través de la interfaz gráfica que se muestra en la Figura 1.

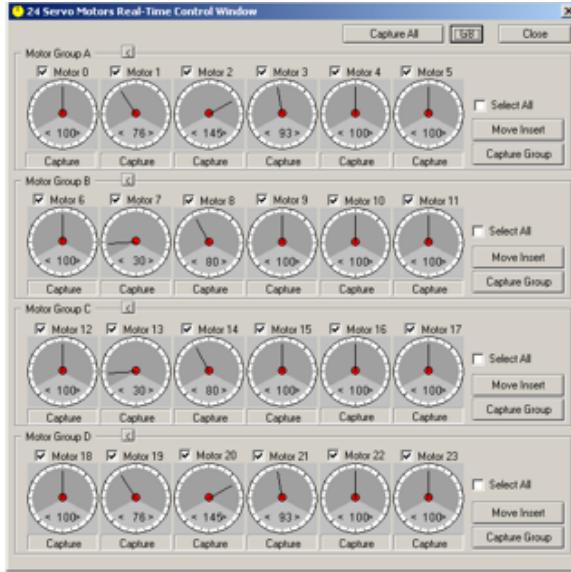


Figura 1: Captura de pantalla de la interfaz Catch and Play de RoboBASIC

Otra interfaz útil dentro de RoboBASIC era RoboScript. Esta podía considerarse como una combinación entre la interfaz de Catch-and-Play y la programación directa en BASIC. Contaba con comandos básicos necesarios para crear una rutina del robot además de poder modificar la posición de los servomotores a través de una interfaz gráfica que se muestra en la Figura 2.

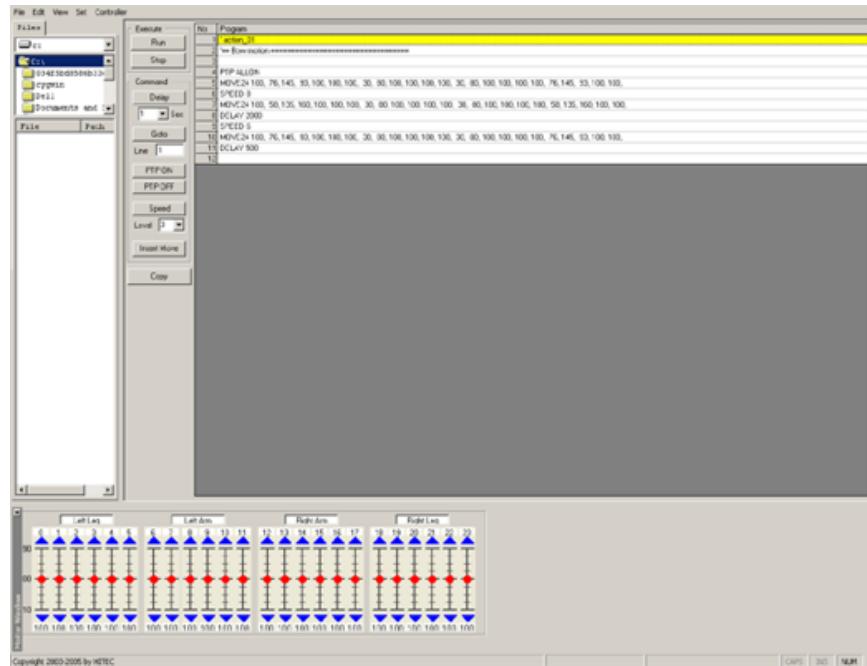


Figura 2: Captura de pantalla de la interfaz RoboScript de RoboBASIC

Por último, los Robonovas cuentan con un control remoto. Este puede ser utilizado para controlar directamente el movimiento del robot o para activar rutinas del mismo. Esta configuración es establecida a través de RoboRemocon, la última interfaz auxiliar de RoboBASIC. Cabe notar que no se lleva a cabo ningún tipo de programación en esta como se hacía en las anteriores, RoboRemocon solamente asigna las funciones y rutinas ya programadas con alguna de las herramientas anteriores y las asigna a un botón específico del control. La interfaz se muestra en la Figura 3.

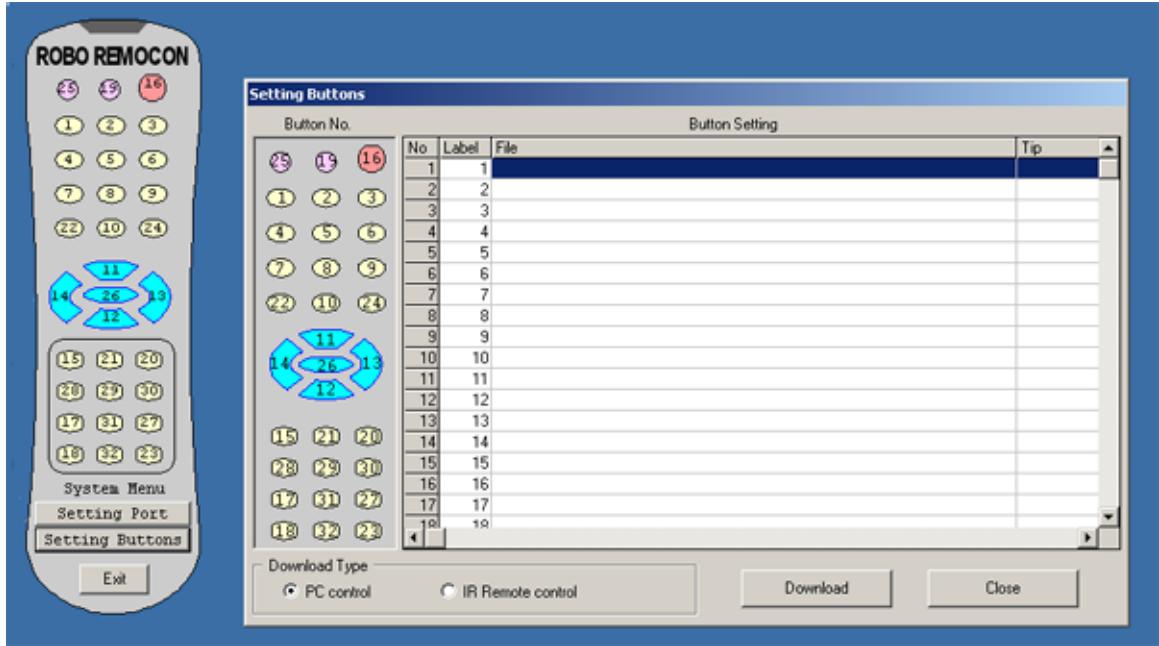


Figura 3: Captura de pantalla de la interfaz RoboRemocon de RoboBASIC

## 2.2. Programación de robots bailarines

En [2] se presenta un proyecto cuyo objetivo era crear un programa para que los robots Robonova-1 reaccionaran a señales de audio y bailen al ritmo de la música.

Los algoritmos de control de este proyecto se dividieron en 3 etapas: identificación del ritmo, la plataforma a utilizar (Robonova-1) y la generación de gestos y control. Para la primera etapa de identificación de ritmo, dado que el Robonova no contaba con el hardware necesario para el procesamiento de las señales, se realizó el programa en una computadora externa al robot la cual se comunicaba con el mismo por medio de Bluetooth. Este programa externo se encargaba tanto de el procesamiento de la música como la transmisión de los gestos al robot. El baile del Robonova consistía en una concatenación de diferentes gestos ya programados cuya velocidad de acción y de cambio entre gestos era regida por el ritmo analizado anteriormente.

Además de implementar el programa para hacer que el robot bailase, el equipo también creó un nuevo ecosistema para controlarlo en lugar de utilizar RoboBASIC. Lograron crear un mejor ecosistema de control que contaba con una latencia mucho menor a la obtenida al utilizar RoboBASIC (0.2 segundos) además de contar con una mayor exactitud en cuanto a las posiciones deseadas de los servomotores. Como se evidencia en la Figura 4.

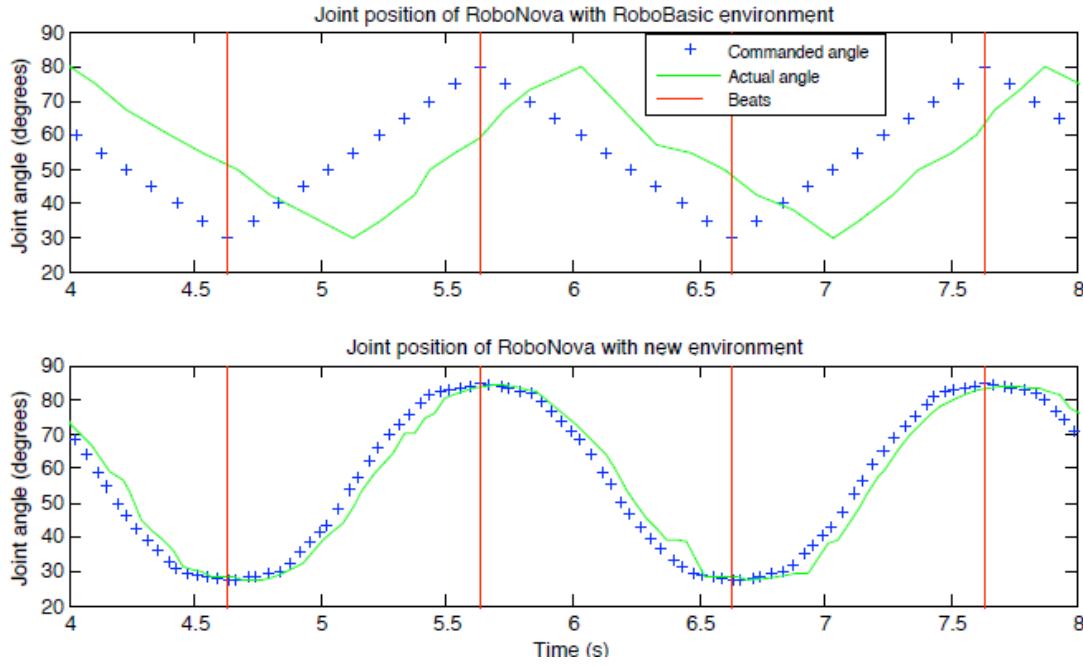


Figura 4: Comparación entre RoboBASIC y el ecosistema creado por investigadores de Drexel University

### 2.3. Atlas de Boston Dynamics

Una de las compañías más innovadoras en el área de la robótica es Boston Dynamics. En su repertorio de robots cuentan con varios robots cuadrúpedos (Spot, Big Dog, LS3, etc.), robots con ruedas (Handle) y robots humanoides como es el caso del robot Atlas. Con 28 grados de libertad y un sistema de control de estado del arte, Atlas es uno de los robots humanoides con mejor movilidad en la academia[3]. Con este robot, Boston Dynamics demuestra los posibles usos de robots humanoides y como estos podrían revolucionar la industria. Este se puede observar en la Figura 5.



Figura 5: Robot Humanoide Atlas de Boston Dynamics

## 2.4. Choregraphe de Aldebaran Robotics

Otro programa de control de un robot humanoide es Choregraphe de Aldebaran Robotics. Esta es la aplicación por defecto para la programación, control, monitoreo y creación de rutinas del robot humanoide NAO.

El robot NAO es uno de los modelos insignia de Aldebaran Robotics. Este robot humanoide cuenta con 25 grados de libertad los cuales le permiten un movimiento fluido y adaptable a su ambiente. También cuenta con varios sensores (de fuera, sonar, inercial, etc.) que le permiten recibir retroalimentación de su ambiente y así poder recalcular la dinámica del sistema para adaptar su movimiento. Además de estos sensores, el robot también cuenta con cámaras, micrófonos y bocinas para interactuar con el usuario.[4] Se puede observar el robot en la Figura 6



Figura 6: Robot Humanoide Aldebaran Robotics

A grandes rasgos el programa Choregraphe permite al usuario controlar al robot NAO de manera gráfica e interactiva. Para lograr esto, la interfaz gráfica de Choregraphe que se encuentra en la Figura 7 cuenta con varias herramientas. La principal herramienta es el panel de diagrama de flujo (C) [5]. Este es una linea del tiempo que conecta las diferentes acciones del NAO para crear una rutina cohesiva. Estas conexiones pueden ser en serie o paralelo, utilizando subrutinas básicas que se encuentran dentro del programa (B) como hablar o caminar, o bien, utilizar rutinas creadas por el usuario [6]. Además de esto, el programa también cuenta con una ventana que permite visualizar una simulación del robot NAO llevando a cabo la rutina (E).

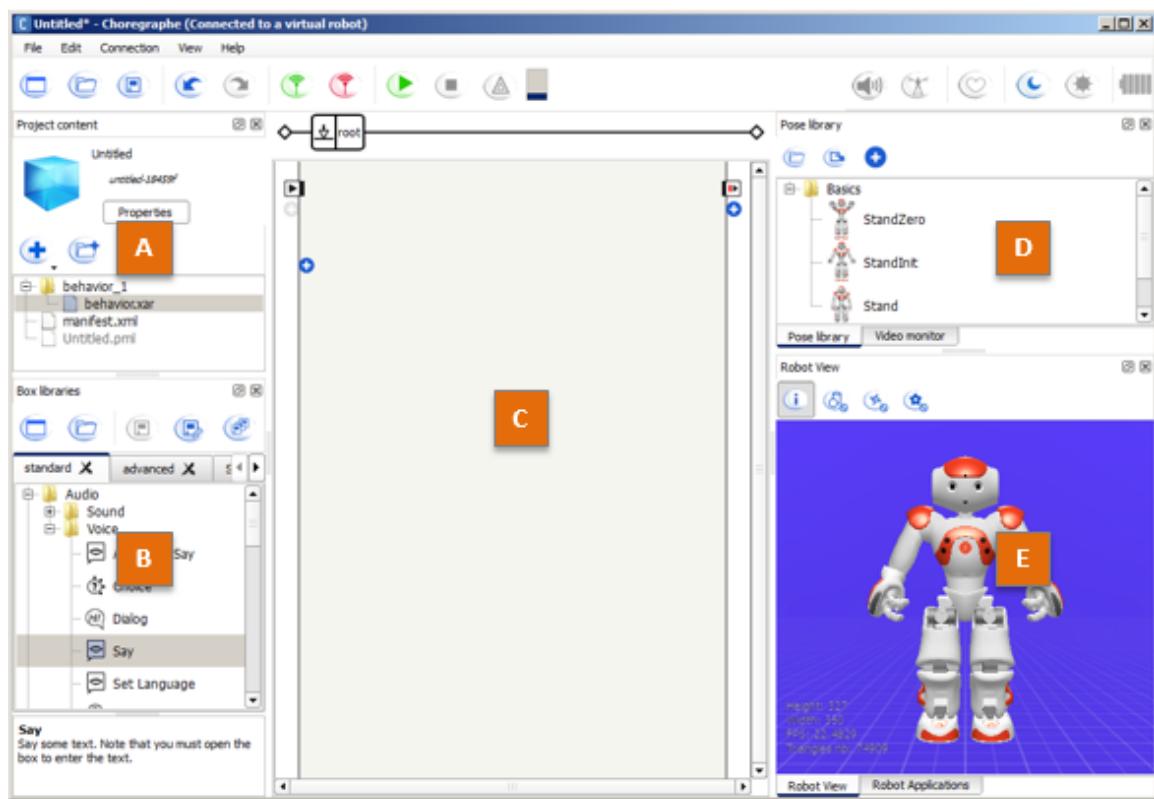


Figura 7: Interfaz de usuario de Choregraphe

## CAPÍTULO 3

---

### Justificación

---

En los últimos años se han dado bastantes avances en robótica gracias a la investigación y desarrollo que varias instituciones llevan a cabo utilizando robots humanoides. Desde la manufactura de piezas mecánicas más eficientes hasta la programación de sistemas de control más avanzados y robustos, el uso de estos robots en investigación ha aportado bastante en el área de ingeniería. Por esta razón, este proyecto se llevará a cabo con el fin de revitalizar los robots humanoides Robonova-1 disponibles en la Universidad del Valle de Guatemala. Lo que se desea es actualizar su interfaz de programación además de su conexión y control de forma inalámbrica. El contar con estas herramientas aplicadas a un prototipo como el Robonova mejorará la investigación y aprendizaje dentro de la universidad en las áreas anteriormente mencionadas y abriría el campo para el futuro control de robots humanoides más sofisticados.

Como se pudo observar en los antecedentes, la plataforma RoboBASIC que se utiliza por defecto para la programación de los Robonova, a pesar de contar con diversas interfaces para facilitar la creación de rutinas, no es tan amigable y ya no cuenta con soporte, con su última actualización siendo en mayo de 2008. Además de ello, no cuenta con monitoreo en software de el estado actual del Robonova lo cual es una característica importante para el control de robots. Lo que se busca entonces es crear una interfaz de programación similar a Coreographe (la interfaz utilizada por los robots NAO). Esta debe permitir al usuario observar el estado de los robots y crear las rutinas en una interfaz comprensible y amigable.

# CAPÍTULO 4

---

## Objetivos

---

### Objetivo general

Diseñar una herramienta de software para el monitoreo y programación de los robots humanoides Robonova-1 de forma gráfica.

### Objetivos específicos

- Conectar la herramienta de software con el robot de forma inalámbrica para su programación y monitoreo en tiempo real.
- Crear una librería de subrutinas para facilitar la creación de coreografías complejas.
- Crear una interfaz gráfica amigable y sencilla para el fácil desarrollo de coreografías de los robots.

# CAPÍTULO 5

---

## Alcance

---

Como fue determinado anteriormente, se desea producir un programa para el control y monitoreo de la nueva versión de los robots Robonova-1 de la Universidad del Valle. Ya que el diseño eléctrico y mecánico no se encuentra finalizado, el alcance de esta tesis se reduce al correcto funcionamiento del programa en una simulación además de conexiones básicas a prototipos del Robonova que se encuentren disponibles.

Cabe notar que dentro de la tesis se estará trabajando con diversas herramientas y conceptos científicos basados en investigación para lograr los objetivos deseados. Estas herramientas y conceptos incluyen:

- Motor físico pyBullet para simulación del robot
- Definición y análisis del ciclo de Gait de un ser bípedo.
- Uso de formato de intercambio de datos JSON
- Uso de formato de intercambio de datos URDF
- Diseño mecánico de las piezas de un robot humanoide en un programa CAD.
- Diseño eléctrico para la nueva versión del robot humanoide Robonova.

El desarrollo de los mismos se encuentra fuera del alcance de esta tesis.

Además de esto, la nueva versión del Robonova-1 se está trabajando en paralelo a esta tesis por lo cual el monitoreo de las posiciones de los servomotores (*feedback*) también se encuentra fuera del alcance.

# CAPÍTULO 6

---

## Marco teórico

---

### 6.1. Biomecánica del movimiento humano

Para lograr que un robot bípedo emule correctamente los movimientos de una persona, primero se debe empezar por analizar la mecánica de una persona en movimiento. El área de estudio que se enfoca en esto se denomina biomecánica. Esta se enfoca en modelar a los seres vivos por medio de representaciones físicas más sencillas como masas, eslabones, uniones, resortes, etcétera, para poder implementar un análisis cinemático y dinámico de su movimiento [7].

Como se puede observar en la Figura 8, un modelo simple para una persona (o robot humanoide) es un conjunto de eslabones unidos por diferentes tipos de uniones. Durante su movimiento, estos eslabones se moverán de acuerdo a los grados de libertad que cada unión permita. Para modelos biomecánicos sencillos, estas uniones generalmente son revolutas con un grado de libertad. Cabe notar que, a demás de las articulaciones, también se encuentra marcado el centro de masa del modelo al rededor del cual se llevan a cabo los cálculos.

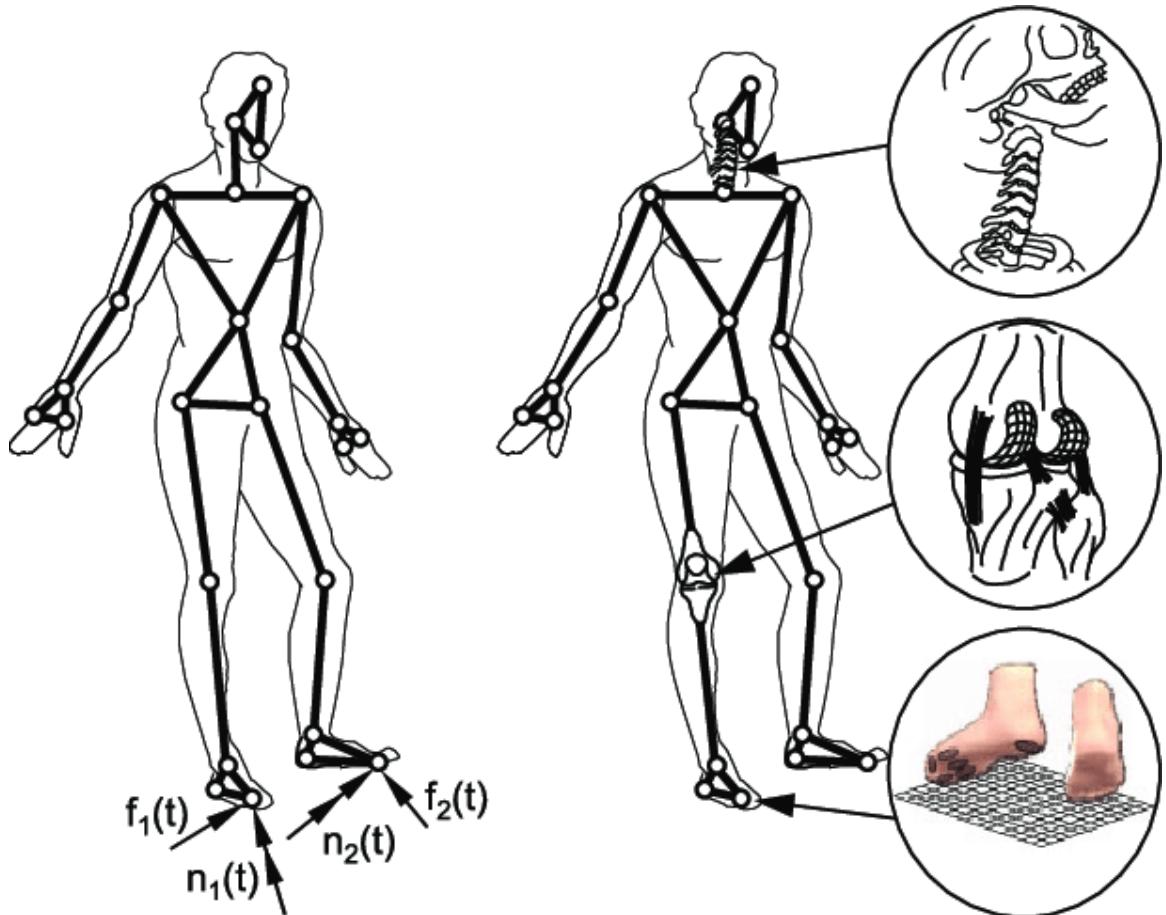


Figura 8: Modelo de juntas y eslabones de un sistema biomecánico [7]

### 6.1.1. Ciclo de Gait

Al llevar a cabo movimientos que se repiten una y otra vez como es el caso de caminar y correr, estos pueden ser analizados por medio de un ciclo de Gait como se observa en la Figura 9. Este ciclo separa los movimientos en distintas fases para su análisis. El ciclo de Gait de caminar puede ser de dos fases: la postura y el balanceo. Durante la fase de postura, la cual es el 60 % del ciclo de Gait [8], la pierna se encuentra en contacto con el piso. Este contacto se subdivide en 5 fases: *Heel strike*, ocurre cuando el talón entra en contacto con el piso; *loading response*, cuando este se prepara para cargar con todo el peso del cuerpo; *midstance*, donde todo el peso se encuentra sobre esa pierna; *heel off*, cuando el pie comienza a levantarse del piso; y por último, *toe off*, cuando ya se levantó por completo. Durante la fase de balanceo, como lo indica su nombre, la pierna se balancea con momentum hacia adelante para llegar a su nueva posición. Esta fase también puede ser dividida en 3 fases propias: aceleración, balanceo y desaceleración. Para un bípedo, el proceso de caminar consiste en un ciclo de Gait continuo en cada pierna, desfazados entre sí.

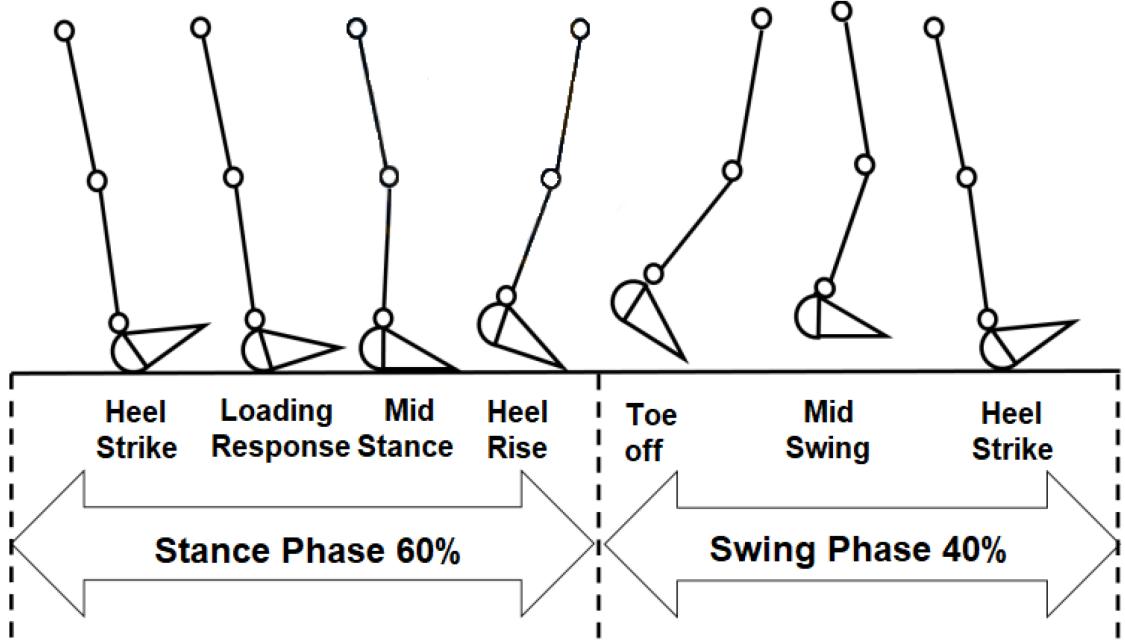


Figura 9: Ciclo de Gait [8]

## 6.2. Cinemática de robots humanoides

Un robot humanoide es un tipo específico de robot de base flotante cuya forma tiene inspiración en el cuerpo humano. Por ello, la mayoría de estos robots cuentan con 4 extremidades: 2 brazos y 2 piernas. En los robots de base flotante, generalmente, su base se encuentra en el centro de masa del mismo. Ya que no es posible actuar sobre la base directamente este tipo de robot moviliza su base de forma indirecta controlando sus extremidades. Para ello, se crean juntas virtuales entre el robot y un marco de referencia fijo como se muestra en la Figura 10 [9].

Para el control de las extremidades del robot se puede considerar cada una como un manipulador serial independiente saliendo de la base. Como se observa en [10], cada una de las extremidades tendrá su propia matriz de Denevit-Hartenberg en relación a la base flotante del robot. Al plantear el robot de esta manera, la cinemática directa e inversa de los brazos del mismo se vuelve trivial.

Este no es el caso para las piernas del robot. Debido a que estas deben sostener el peso entero del robot, además del control cinemático que se lleva a cabo, también deben tomarse en cuenta consideraciones dinámicas al controlar las piernas. En [11], se muestran dos modelos para tomar en cuenta estas consideraciones: el modelo LIMP (Linear Inverted Pendulum Model) y el modelo ZMP (Zero Moment Point). En el modelo LIMP, las piernas actúan como una distancia desde el pivote (el suelo) hasta el centro de masa del robot, simplificando así el análisis físico. Una vez modelado de esta forma el ZMP es el posicionamiento del centro de masa de tal manera que este no genere un torque en el plano xy evitando así que el robot pierda su estabilidad al momento de dar un paso.

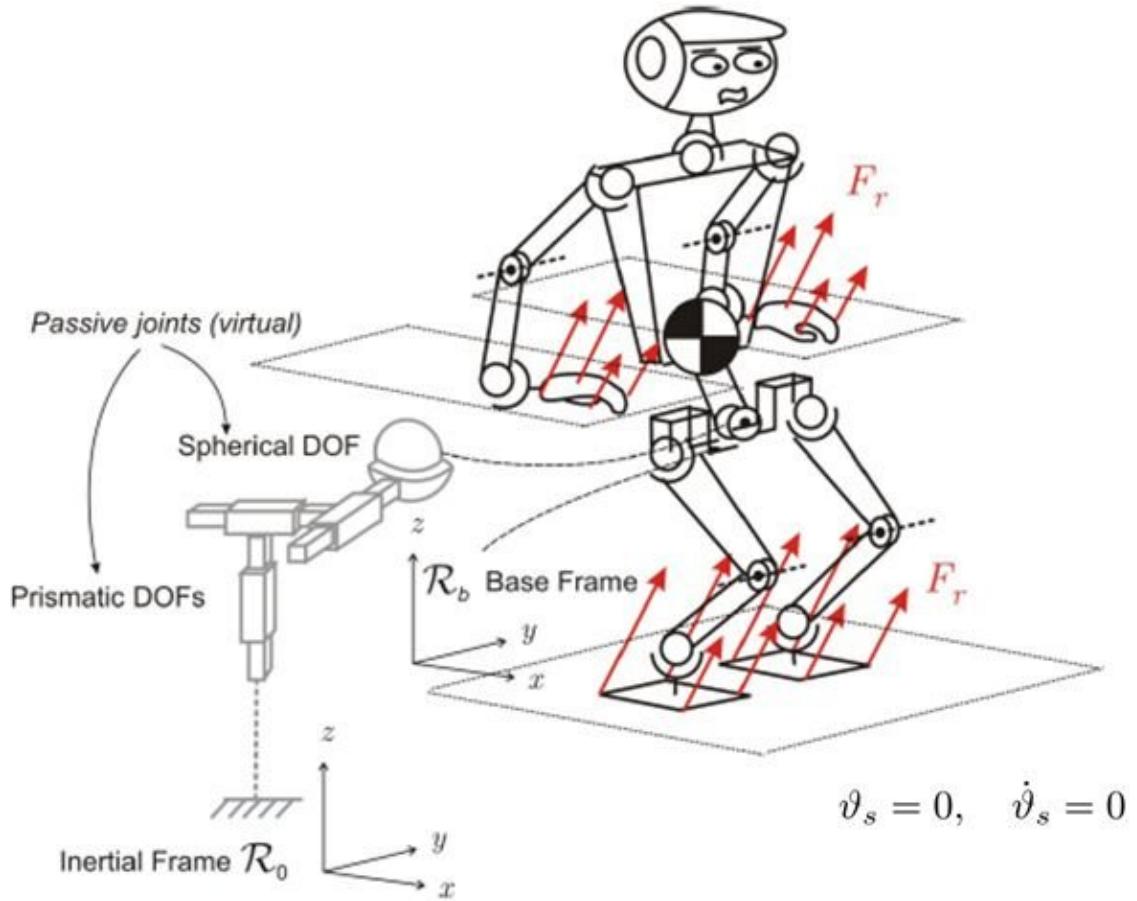


Figura 10: Representación cinemática de un robot humanoide [12]

### 6.3. Robonova-1

#### 6.3.1. Versión original

El robot humanoide Robonova-1 (Figura 11) fue usado como base para esta tesis. El robot, creado por, HITEC Robotics, cuenta 16 grados de libertad. Estos grados de libertad son controlados por servomotores que actúan como juntas revolutas en las articulaciones del robot. Como se observa en la Figura 12, Cuenta con 3 grados de libertad en cada brazo y 5 en cada pierna emulando las articulaciones de los codos, hombros, caderas, rodillas y tobillos de un ser humano. [13]



Figura 11: Diseño original del Robonova-1 [14]

El Robonova-1 es controlado por un controlador de robot MR-C3024 cuyas características se pueden observar en el cuadro 1. Este controlador permite programar al robot por medio de comunicación serial (UART) para cargar el programa actualizado al Robonova. También permite, mientras se mantenga la conexión, controlar y calibrar los servomotores utilizando RoboBASIC. Cabe notar que la placa, a pesar de utilizar solamente 16 servomotores, cuenta con espacio para 24 permitiendo agregar más servomotores o cualquier actuador que se controle por medio de PWM.

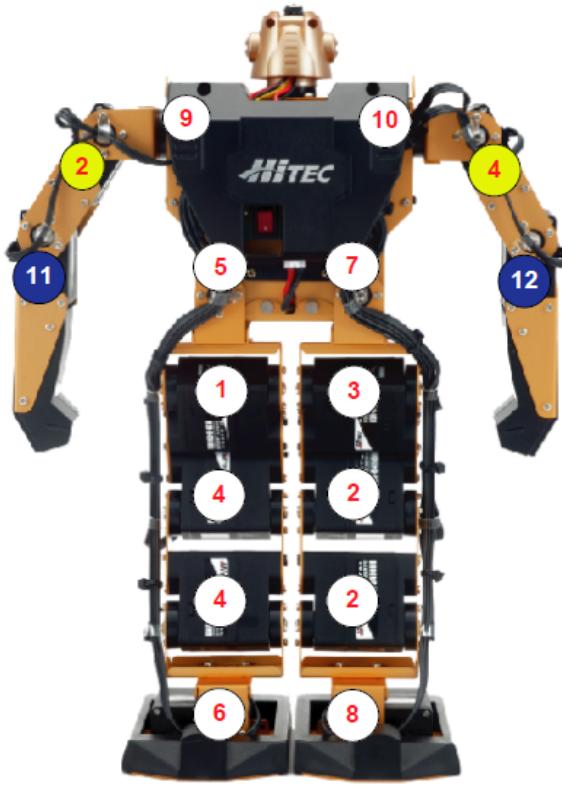


Figura 12: Numeración de los servomotores en la versión original del Robonova-1 [13]

Modelo	MR-C3024
CPU	Atmel ATMEGA128 8bit RISC
Puertos I/O	24
Control de Servomotores	24
Control PWM de motores DC	3
ADC	8 canales
Memoria	32 Kb
Sensores Ultrasónicos	12 canales
Receptor de Control IR	Sí
Receptor RF	Sí
Características comunes	LCD
	Piezo de 6 octavas (música, voz) Comunicación Serial RS-232 (UART)

Cuadro 1: Características del controlador MR-C3024 [15]

Los servomotores utilizados por en el diseño son los HSR-8498HB, también diseñados por HITECH Robotics. Estos servomotores, cuya hoja de datos se puede observar más a detalle [16], son servomotores con engranajes hechos de resina de alta resistencia. Estos servomotores pueden ser operados a 6 V o 7.4 V, llegando a generar un torque máximo de 9 Kg\*Cm (a 1480 mA). Para alimentar los servomotores y el controlador, el Robonova cuenta

con un paquete de baterías Ni-Mh de 6 V - 1,000 mAh.

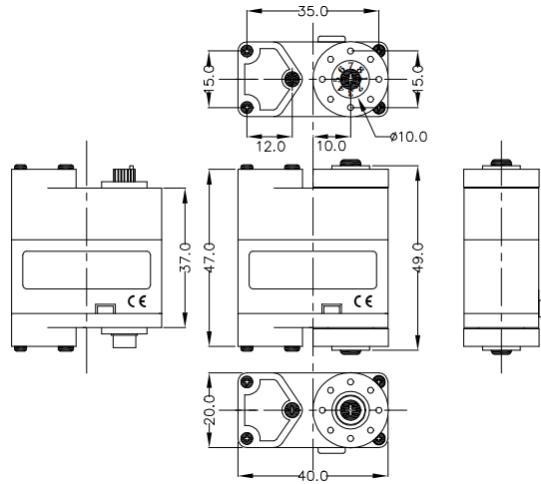


Figura 13: Planos de diseño mecánico de los servomotores HSR-8498HB

### 6.3.2. Nueva versión (diseñada por Fernando José Caceros)

Además de la versión original del Robonova-1, también se está trabajando en la Universidad del Valle de Guatemala una nueva versión del robot. Esta consiste en utilizar el diseño original como inspiración además de cualquier piezas que puedan ser reutilizadas dándole una actualización completa al diseño anterior.

Iniciando con el diseño mecánico, la nueva versión del Robonova tiene algunos cambios. Principalmente, el diseño de las piezas fue recreado desde cero en OnShape e impreso en PLA utilizando impresoras 3D. Como se observa en la Figura 14, la principal diferencia en los diseños es la falta de grados de libertad en los codos, sustituido por otro grado de libertad en los tobillos. Al estar reutilizando los servomotores originales del Robonova, Se decidió realizar este cambio para generar una emulación más cercana a la caminata de un ser humano.

Además de los cambios mecánicos, también se le hizo una modernización al sistema embebido utilizado para controlar al Robonova. En lugar de la placa MR-C3024, se ha utilizado un ESP32 DevKit V1 6.11 conectado a dos placas controladoras de servo PCA9685. Esta actualización al sistema embebido permite la comunicación por WiFi al programa para el control y programación del robot.



Figura 14: Nueva versión del robot humanoide Robonova

Cabe notar que el diseño de la nueva versión del Robonova aún se encuentra en proceso por lo cual puede estar sujeta a cambios.

#### 6.4. PyBullet

PyBullet es una librería para Python del *Bullet Physics engine*. Esta librería se enfoca en el estudio de detección de colisiones y dinámica de cuerpos rígidos. La librería cuenta con simulaciones de dinámica directa, dinámica inversa, detección de colisiones y intersección de rayos. Esta librería se utiliza principalmente la simulación de sistemas de robótica antes de su implementación física además de tener aplicaciones en videojuegos y realidad virtual.

Con el área de interés siendo simulaciones de robótica, PyBullet cuenta con herramientas para crear y analizar diversos tipos de robots. Además de contar con una librería de robots de ejemplo, PyBullet cuenta con comandos para cargar la información de un robot específico en diversos formatos como URDF, SDF, MJCF, entre otros. Una vez cargados los datos del robot se puede analizar y controlar dentro del espacio de simulación. PyBullet cuenta con funciones para obtener las propiedades físicas del robot además de funciones que le permiten al programador controlar el mismo.[17]

Además de esto, la librería también cuenta con funciones para el cálculo matemático de la cinemática y dinámica inversa de los robots, detección de colisiones, visión de computadora y realidad virtual.[17]

## 6.5. Robotics Toolbox de Peter Corke

Esta librería es una versión planteada para Python de la Robotics Toolbox para MATLAB de Peter Corke, y cuenta con herramientas para el análisis físico de robots. Utilizando todas estas herramientas, el programa puede modelar robots, analizar y crear trayectorias, llevar a cabo manipulación simbólica de variables, analizar la cinemática diferencial del robot e incluso la dinámica del mismo.[18]

La Robotics Toolbox ya cuenta con más de 30 modelos cinemáticos de diversos tipos de robots. Además de esto, es posible utilizarla para modelar robots propios de diversas maneras. Se puede generar la matriz de Denavit-Hartenberg con las dimensiones adecuadas del robot o también se pueden cargar los datos del mismo por medio de un *string ETS* o un archivo URDF. Además de los parámetros cinemáticos, también se pueden agregar otros parámetros como la masa, CoG, inercia de los motores, límites de las articulaciones, etcétera, utilizando palabras clave dentro de la programación.

En cuanto a las trayectorias, el Robotics Toolbox trabaja con polinomios de quinto orden para proporcionar un *jerk* continuo. Estas trayectorias proporcionan tanto la posición, velocidad y aceleración de los actuadores para llegar al punto deseado.

Como se estipuló anteriormente, la Robotics Toolbox también computa cinemática diferencial de los robots. Puede computar los Jacobianos, Hessianos y la manipulabilidad de todos los robots.

Por último, también es posible analizar la dinámica de los robots utilizando integración numérica. Esta se puede utilizar para calcular los términos de inercia, Coriolis y gravedad por medio de dinámica inversa.[18]

## 6.6. Formato JSON

El formato JSON (*JavaScript Object Notation*) es un formato de intercambio de datos ligero, basado en texto, independiente de lenguaje. Es considerado una alternativa ligera al formato XML [19]. Su compatibilidad con una gran variedad de lenguajes de programación lo hace un formato útil para el intercambio de datos entre programas escritos en diferentes lenguajes.

El formato utiliza llaves, corchetes, dos puntos y comas para ordenar los valores en dos diferentes tipos de estructuras: objetos y *arrays*. Un objeto es categorizado como un grupo de propiedades que a su vez son el par nombre-valor [20]. Un array es un conjunto de elementos, los cuales a su vez pueden ser objetos, valores o incluso otros arrays. En las figuras 15 y 16 se encuentran ejemplos de un objeto y un array de objetos en formato JSON, respectivamente.

```
{  
  "Image": {  
    "Width": 800,  
    "Height": 600,  
    "Title": "View from 15th Floor",  
    "Thumbnail": {  
      "Url": "http://www.example.com/image/481989943",  
      "Height": 125,  
      "Width": 100  
    },  
    "Animated": false,  
    "IDs": [116, 943, 234, 38793]  
  }  
}
```

Figura 15: Ejemplo de un objeto en formato JSON [21]

```
[  
  {  
    "precision": "zip",  
    "Latitude": 37.7668,  
    "Longitude": -122.3959,  
    "Address": "",  
    "City": "SAN FRANCISCO",  
    "State": "CA",  
    "Zip": "94107",  
    "Country": "US"  
  },  
  {  
    "precision": "zip",  
    "Latitude": 37.371991,  
    "Longitude": -122.026020,  
    "Address": "",  
    "City": "SUNNYVALE",  
    "State": "CA",  
    "Zip": "94085",  
    "Country": "US"  
  }  
]
```

Figura 16: Ejemplo de un *array* en formato JSON [21]

## 6.7. Formato URDF

El formato URDF (*Unified Robotics Description Format*) es un formato de intercambio de datos basado en XML utilizado para modelar ensamblajes robóticos de múltiples piezas [22]. La sintaxis de este formato cuenta con etiquetas específicas para definir cada parte del robot: *robot*, *link*, *joint* y extras.

La etiqueta *robot* debe ser utilizada al inicializar la definición del robot. Dentro de esta se escribirán todas las demás etiquetas al momento de diseñar el robot. La segunda etiqueta, *link*, da las características físicas de los eslabones del robot. Estas pueden ser visuales (*visual*), configurando los parámetros *geometry*, *origin* y *material*; de colisión (*collision*), configurando sus parámetros de *geometry* y *origin*; e iniciales (*inertial*), configurando los parámetros *mass*, *origin* e *inertia*. La tercera etiqueta, *joint*, configura las juntas del robot las cuales son: su nombre (*name*), tipo de junta (*type*), eslabones conectados (*parent/child*), origen (*origin*), ejes (*axis*) y límites (*limits*) [23]. Cada etiqueta también cuenta con otros parámetros para configuraciones más específicas del robot. En la Figura 17 se puede observar un ejemplo de la sintaxis de un archivo URDF.

## 6.8. Onshape-to-robot

Onshape-to-robot es una librería creada por Rhobah, un grupo de investigadores de la Universidad de Bordeaux. Esta herramienta permite importar robots diseñados en Onshape CAD a formatos de descripción como URDF o SDF para poder utilizarlos dentro de simuladores físicos como pyBullet. [24] Esta librería crea el URDF en base a las dimensiones de los eslabones y los tipos de juntas utilizados en el ensamblaje general de Onshape. Además de generar el URDF, la librería también genera los archivos .stl de todas las piezas del robot.

## 6.9. PyQT

PyQT es una librería de python que permite crear aplicaciones QT. Estas aplicaciones QT consisten de una interfaz de usuario utilizando diversos tipos de entradas para que el usuario interactúe con el programa de python subyacente.[25]

```

<robot name>
    <link name>
        <inertial>
            <origin xyz("0 0 0") rpy("0 0 0") />
            <mass value />
            <inertia ixx iyy izz ixy ixz iyz />
        </inertial>
        <visual name>
            <origin xyz("0 0 0") rpy("0 0 0") />
            <geometry>
                <box size />
                <cylinder radius length />
                <sphere radius />
                <mesh filename scale("1") />
            </geometry>
            <material name>
                <color rgba("0.5 0.5 0.5 1") />
                <texture filename />
            </material>
        </visual>
        <collision name>
            <origin xyz("0 0 0") rpy("0 0 0") />
            <geometry>
                <box size />
                <cylinder radius length />
                <sphere radius />
                <mesh filename scale("1") />
            </geometry>
        </collision>
    </link>
    <joint name type>
        <origin xyz("0 0 0") rpy("0 0 0") />
        <parent link />
        <child link />
        <axis xyz("1 0 0") />
        <calibration rising />
        <calibration falling />
        <dynamics damping("0") friction("0") />
        <limit lower+ upper+ effort velocity />
        <mimic joint multiplier("1") offset("0") />
        <safety_controller soft_lower_limit("0") ...
            ... soft_upper_limit("0") k_position("0") k_velocity("0") />
    </joint>
</robot>

```

Figura 17: Ejemplo de la sintaxis de un archivo URDF

## 6.10. Protocolo de comunicación TCP/IP

El protocolo de comunicación TCP/IP es un conjunto de normas estandarizadas para el envío y recepción de datos entre máquinas y aplicaciones. Cada máquina debe seguir las normas para garantizar la comunicación correcta de los mensajes. El protocolo TCP/IP se puede visualizar en capas como se observa en la Figura 18.

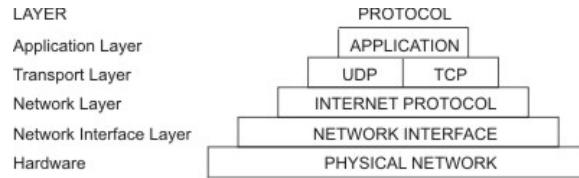


Figura 18: Capas del protocolo de comunicación TCP/IP

### 1. Capa de acceso a la red:

Gestiona la infraestructura física de la comunicación. En esta capa se incluyen los cables Ethernet, redes inalámbricas, tarjetas de interfaz de red, etc.

### 2. Capa de red:

Controla el flujo y enrutamiento del tráfico para garantizar el envío de datos.

### 3. Capa de transporte:

Proporciona conexión de datos fiable. Se encarga de empaquetar los datos y confirmar al destino y al remitente que se han recibido con éxito.

### 4. Capa de aplicación:

Es la capa más cercana al usuario. Permite al usuario acceder a los datos recibidos y preparar los datos que serán enviados [26].

## 6.11. ESP32 DevKit V1

El ESP32 DevKit es una placa de desarrollo del microcontrolador ESPWROOM32. Este microcontrolador, compatible con Arduino IDE, cuenta con una gran disponibilidad de diversas útiles para cualquier programador. Principalmente, cuenta con módulos WiFi y Bluetooth para programación y conexión inalámbrica [27]. Las características del microcontrolador se encuentran en la Tabla 2.

Característica	Cantidad
Vin	4 - 12 V
GPIO	34
Pull-up & Pull-down Internos	Todos los puertos GPIO
LED	GPIO2
UART	3 Canales
ADC	18 canales
SPI	4 Canales
DAC	2 canales
Sensor Táctil	10 Canales
I2C	2 Canales
PWM	16 Canales
CAN/TWAI	1 Canal
Interfaz JTAG	Sí
Interrupciones Externas	Todos los puertos GPIO
Ethernet MAC	Sí

Cuadro 2: Características del controlador ESP32 [28]

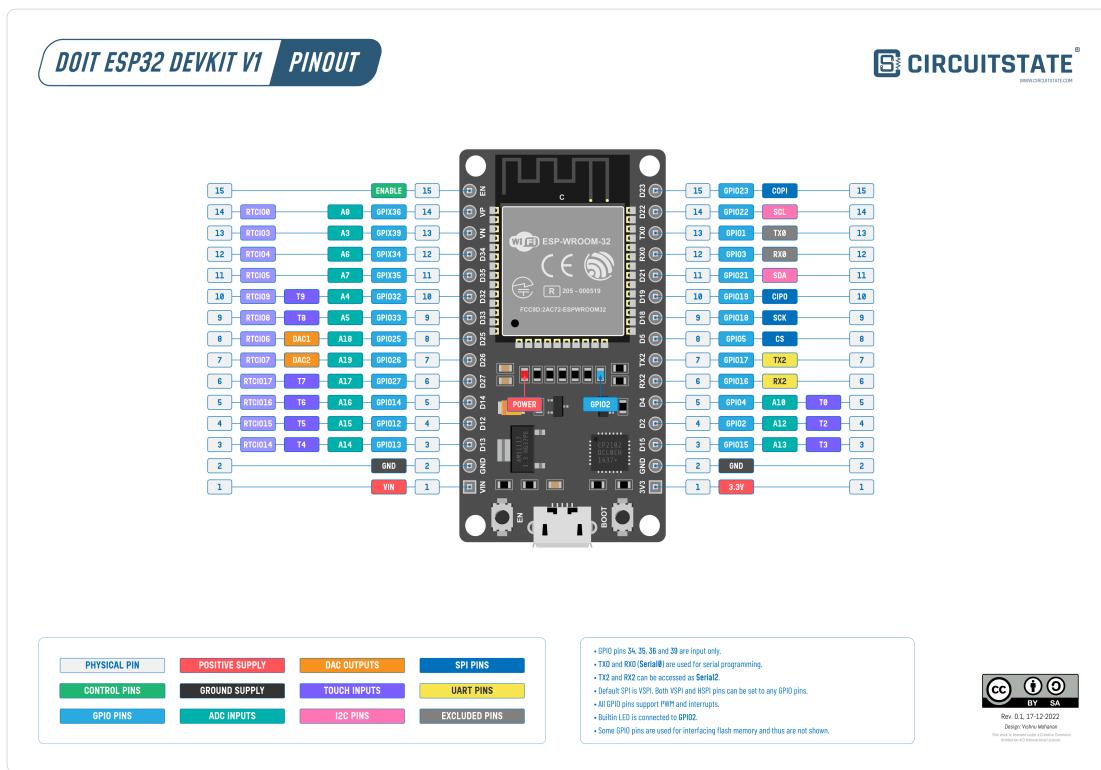


Figura 19: Mapa de pines del ESP32 [28]

# CAPÍTULO 7

---

## Generación Automática del URDF de la nueva versión del Robonova

---

### 7.1. Diseño de ensamblaje

#### 7.1.1. Metodología

Para poder utilizar la librería de onshape-to-robot descrita en 6.8 se debe crear un ensamblaje del robot cumpliendo condiciones específicas. Las piezas a ensamblar en Onshape fue proporcionado por Fernando José Caceros, como se presentó en la sección 6.3.2.

La primera condición es que los grados de libertad del robot se deben encontrar definidos en el ensamble principal, no en subensambles. Estos deben ser nombrados *dof\_NombreDeLaJunta*. El prefijo *dof\_* le permite al programa onshape-to-robot reconocer y nombrar la junta en el URDF.

Además de esto, las juntas pueden ser restringidas desde Onshape como se muestra en la Figura 20. Las restricciones o desfaces que se lleven a cabo al configurar la junta en onshape serán aplicados al generar el URDF.

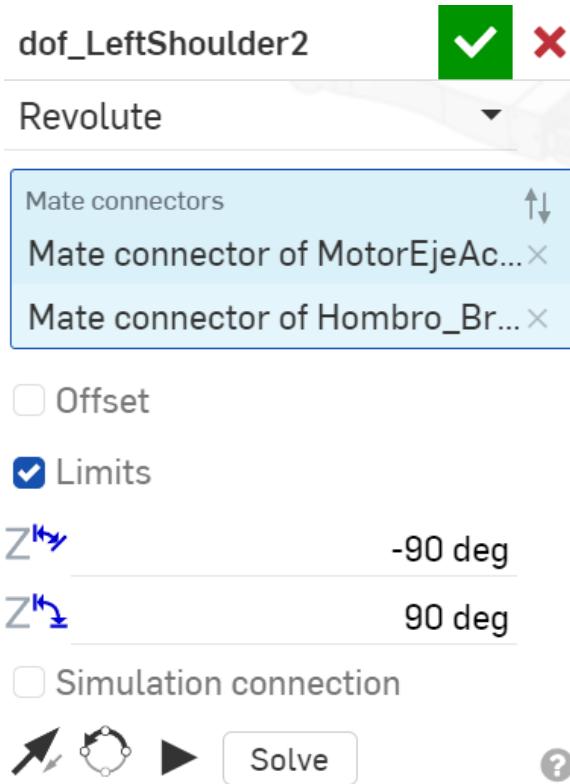


Figura 20: Configuración de junta en Onshape

Por último, la posición inicial del robot será la posición en la cual se encuentra el ensamblaje al momento de generar el URDF.

### 7.1.2. Resultados

Una captura de pantalla del ensamblaje final se observa en la Figura 21. Cabe notar que en el recuadro rojo se observan todos los grados de libertad escritos en el formato establecido en la metodología. Este ensamblaje final cuenta con todos los nombrados y restringidos a ángulos que un ser humano podría lograr.

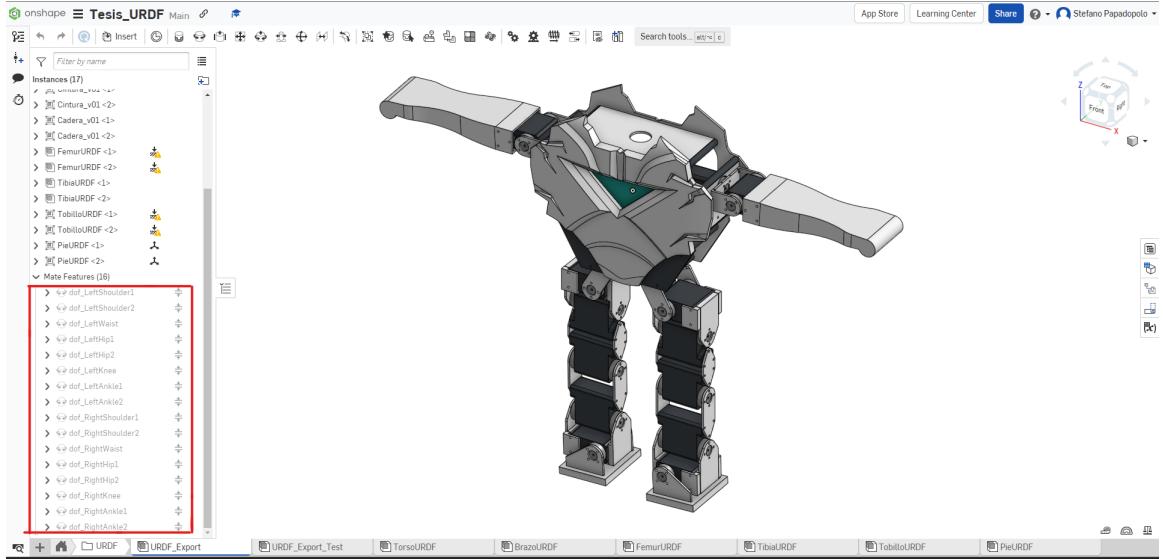


Figura 21: Ensamblaje final de la nueva versión del robonova

## 7.2. Uso de onshape-to-robot

### 7.2.1. Metodología

Una vez se tiene el ensamblaje listo en Onshape, se puede utilizar la librería de python onshape-to-robot para generar el URDF. Para lograr esto, primero se debe hacer una configuración inicial del ordenador para darle accesos a las llaves API de Onshape y así poder conectarse a los servidores y extraer el ensamblaje<sup>1</sup>. Para más información acerca de como lograr esto se puede hacer referencia a [24].

Una vez lograda la conexión inicial, se debe crear un archivo de configuración llamado *config.JSON*. Este archivo se utiliza para que la librería reconozca que archivo extraer y como extraerlo. El archivo debe contar con dos objetos: identificación del documento, para determinar que archivo se está buscando en los servidores de Onshape y el formato de salida, el cual será URDF. Además de estos objetos, el archivo *config.JSON* también puede contener objetos adicionales para una configuración más específica. Entre estos se encuentran: el nombre del ensamble, identificación de la versión, configuraciones iniciales del robot, dinámica del sistema, entre otros.

Con el archivo *config.JSON* creado y dentro de una carpeta propia, el último paso es correr el programa. Para ello, se debe abrir una ventana terminal en el directorio donde se encuentra la carpeta y escribir el comando **onshape-to-robot nombre\_de\_carpeta**. Una vez ejecutado el comando, la librería genera automáticamente archivos .part y .stl de los eslabones del robot además de un URDF con su descripción.

---

<sup>1</sup>El programa Onshape-to-Robot solamente puede ser utilizado en Linux. Esto debido a una integración innata del programa con ROS y que se debe configurar las claves dentro del archivo .basch para conectar al sistema operativo con la API de Onshape.

## 7.2.2. Resultados

Una vez el programa termina de correr, se puede observar que en la carpeta donde se encontraba la el archivo *config.JSON* ahora cuenta con los archivos mencionados anteriormente y un archivo URDF con el nombre *robot.urdf*. Esto se ve reflejado en la Figura 22

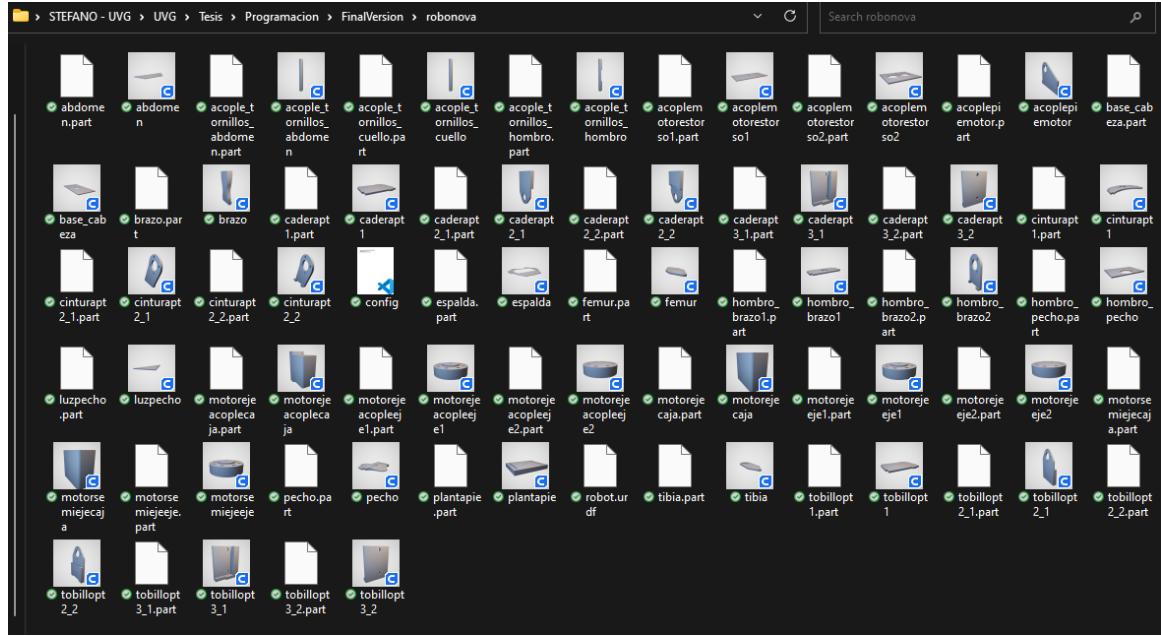


Figura 22: Folder con piezas y URDF del Robonova

## 7.3. Discusión

Para poder utilizar el programa de control, es imperativo contar con el URDF del robonova ya que de este se extrae la información para poblar la interfaz gráfica. Este podría ser definido manualmente como se explicó en 6.7 pero mientras mayor sea la cantidad y complejidad de las piezas y la sintaxis y longitud del archivo se vuelve demasiado difícil. Además de la restricción en el diseño de piezas escritas manualmente lo cual limita las geometrías posibles. El utilizar esta librería permite crear esas piezas complejas además de optimizar el tiempo y los recursos del diseñador extrayendo los STLs y creando el URDF directo del CAD.

Cabe notar que una desventaja de obtener el URDF de esta manera es que restringe al diseñador del robot a utilizar Onshape para diseñar y ensamblar el robot o, por lo menos, importar las piezas diseñadas a Onshape y ensamblarlo ahí. A pesar de ello, la facilidad que proporciona en la creación del URDF de robots, sobre todo tan complejos como el Robonova, vuelve este inconveniente despreciable a comparación.

## CAPÍTULO 8

---

### Programa en python para simulación y control de Robonova

---

Partiendo desde los antecedentes, el objetivo del nuevo programa para el control de la nueva versión del Robonova fue poder recrear lo hecho por David Buckley en el programa de RoboBASIC [1] original e incluir funcionamiento adicional inspirado en la programación de los robots NAO y otras fuentes. Además de las funciones ya existentes en los antecedentes, se incluyó una simulación en pyBullet en el programa. Esto fue con el afán de facilitar la logística de programación de rutinas del robot. El incluir la simulación, también permite expandir el uso del programa para crear rutinas con diversos robots, no solamente el Robonova.

Específicamente, el programa debe contar con las siguientes características:

- Tener una interfaz gráfica intuitiva y amigable.
- Visualizar una simulación los movimientos del Robonova desde el programa.
- Creación, edición y reproducción de rutinas básicas.
- Conectarse y controlar los servomotores del Robonova en tiempo real (esto será expandido en el capítulo 9).

En este capítulo se presenta la metodología llevada a cabo para lograr esto y los resultados obtenidos.

## 8.1. Primera versión del programa

### 8.1.1. Metodología

#### Interfaz Gráfica

Una parte fundamental del programa es la interfaz gráfica. Existen diversas opciones para diseñar una interfaz gráfica en python. Existen varias librerías para crear interfaces gráficas cuya sintaxis es bastante similar: *tkinter*, *kivy*, *PyQt5*, etc. Para el diseño de esta aplicación se decidió utilizar *PyQt5*, principalmente por su extensa y ordenada documentación además de accesos a la API de Qt para mayor control en el diseño y configuración de la interfaz.

*PyQT5* cuenta con diversas clases para poblar la interfaz con las herramientas que utilizará el usuario. Las clases que se utilizaron fueron: *QPushButton* para crear los botones de pruebas y de configuración de rutinas; *QCheckbox*, para conectar o desconectar el ESP32 que controla el robot; *QDials*, para las diales que controlan los servomotores del robot; y *QPlainTextEdit*, para crear un cuadro de texto que le dé información al usuario. Para la creación de las diales, de igual forma que en la simulación, se crearon funciones para automatizar su creación. Se utilizaron las mismas funciones *getNumJoints* y *getJointInfo* para crear la cantidad de diales exactas para controlar todos los servomotores. Esto con el afán de poder utilizar el programa con más robots. Se puede observar la interfaz en 23. Una vez creados, todos estos *widgets* son conectados a sus funciones específicas para enviar o recibir información al programa. Las funciones son:

- *getData*

Conectada al botón *Get Data*. Esta función toma los valores actuales del *array* de valores de servomotores y los guarda en una segunda variable. Una vez guardados, imprime un mensaje en el cuadro de texto indicando que se guardaron los valores con éxito.

- *sendData*

Conectada al botón *Send Data*. Toma la lista de valores guardada por la función anterior y guarda los mismos en las variables de las diales y de la posición de servomotores en la simulación. Después de eso, imprime un mensaje en el cuadro de texto indicando que se han cargado con éxito.

- *connectRobonova*

Conectada al checkbox *Connect to Robonova*. Al ser activado sube la bandera de conexión al robot Robonova real. Más información acerca de la conexión en tiempo real en 9.

- *updateDial*

Conectada a las diales. Se activa cada vez que una de las diales cambia su valor. Guarda ese valor en una variable que luego es introducida a la etiqueta de valor y a la posición en la lista *servoValues* que controla la posición de los servomotores en la simulación y en el robot.

- *newCoreo*

Conecada al botón *New choreography*. Esta función usa las librerías PyQt5 y os para crear y seleccionar un nuevo archivo .csv donde será guardad la rutina. Utilizando la función *getSaveFileName* de la clase *QFileDialog* se puede crear el archivo (si este ya existe se sobrescribe) y con *os.path.basename* se extrae el nombre del archivo y se guarda en una variable. Esta variable es utilizada en la escritura al .csv en las funciones posteriores.

- *loadCoreo*

Conecada al botón *Load choreography*. Muy similar a la función anterior, utiliza las librerías de PyQt5 y os. A diferencia de la anterior, esta utiliza la función *getOpenFileName* de la clase *QFileDialog*. Esta permite seleccionar (no crear) uno de los archivos ya existentes.

- *addCoreo*

Conecada al botón *Add current position to choreography*. Como lo dice su nombre, esta función consiste en agregar la posición actual a la coreografía. Para ello primero hay que mover las diales que controlan los servomotores para colocar al robot en la posición deseada y luego presionar el botón. La manera en la que la función guarda la posición es bastante sencilla. Primero se abre el archivo .csv que fue seleccionado con alguna de las dos funciones anteriores. Una vez abierto, se utiliza la librería .csv para añadir a la siguiente fila la lista que tiene guardados los valores de todos los servomotores.

- *removeCoreo*

Conecada al botón *Remove last saved position from choreography*. Remueve la última fila del archivo .csv activo borrando así la última posición de la coreografía. Esto se logra leyendo todo el archivo .csv y guardando los valores en una lista utilizando la función *readlines*. Luego, se elimina la última fila de esta lista utilizando la función *pop* y se sobrescribe esta lista ya modificada en el archivo .csv.

- *clearCoreo*

Conecada al botón *Clear choreography*. Como lo dice su nombre, limpia por completo el archivo .csv para poder empezar la rutina desde cero. Muy similar a la función anterior, se sobrescribe una lista en el archivo .csv actual. La única diferencia es que en este caso se sobrescribe una con una *string* vacía para "borrar"todos los contenidos del archivo.

- *deleteCoreo*

Conecada al botón *delete choreography*. De igual forma que se puede borrar la coreografía desde el explorador de archivos, también se puede borrar desde el mismo programa. Utilizando la función *remove* de la librería *os*, esta función toma el archivo .csv activo (que fue seleccionado con la función *newCoreo* o *loadCoreo* y lo elimina del sistema.

- *playCoreo*

La función más importante del programa. Conecada al botón *Play choreography*. Esta función coloca los servomotores en las posiciones del archivo de la coreografía una tras

otra en el orden en el que fueron guardadas. Al igual que en la función *removeCoreo*, el primer paso es leer todo el archivo de la coreografía y guardar los valores en una lista. Luego, se coloca la posición de los servomotores en las posiciones estipuladas por la fila actual de la lista una por una hasta llegar a la última fila. Existe un intervalo de tiempo estipulado al pasar de una posición a otra para darle tiempo a las revolutas de la simulación (y al Robonova físico) de llegar a la posición deseada.

Además del funcionamiento básico para mostrar las posiciones estipuladas en el rutina una tras otra, esta función cuenta con dos configuraciones adicionales conectadas a las checkboxes: *Smooth trajectory* y *Repeat choreography*.

Al activar la configuración de *Smooth trajectory* se toman las posiciones guardadas en el archivo de la coreografía y se hace una interpolación lineal entre las posiciones. Esto genera un movimiento más fluido lo cual será beneficiosos para el control del Robonova ya que evitará movimientos extremadamente bruscos que puedan llegar a dañar los servomotores o las juntas del robot.

La segunda configuración, *Repeat choreography*, inicia un bucle en el cual el robot se mueve de una posición de la coreografía a la siguiente y al terminar regresa al inicio. Este bucle no se detiene hasta que se desactive esta opción. Esta opción permite visualizar de manera más fácil las rutinas sin tener que estar activándolas una y otra vez y, más importante aún, permite movimientos repetitivos como caminar y saludar — movimientos básicos que se podían hacer en RoboBASIC y se desean replicar en este programa.

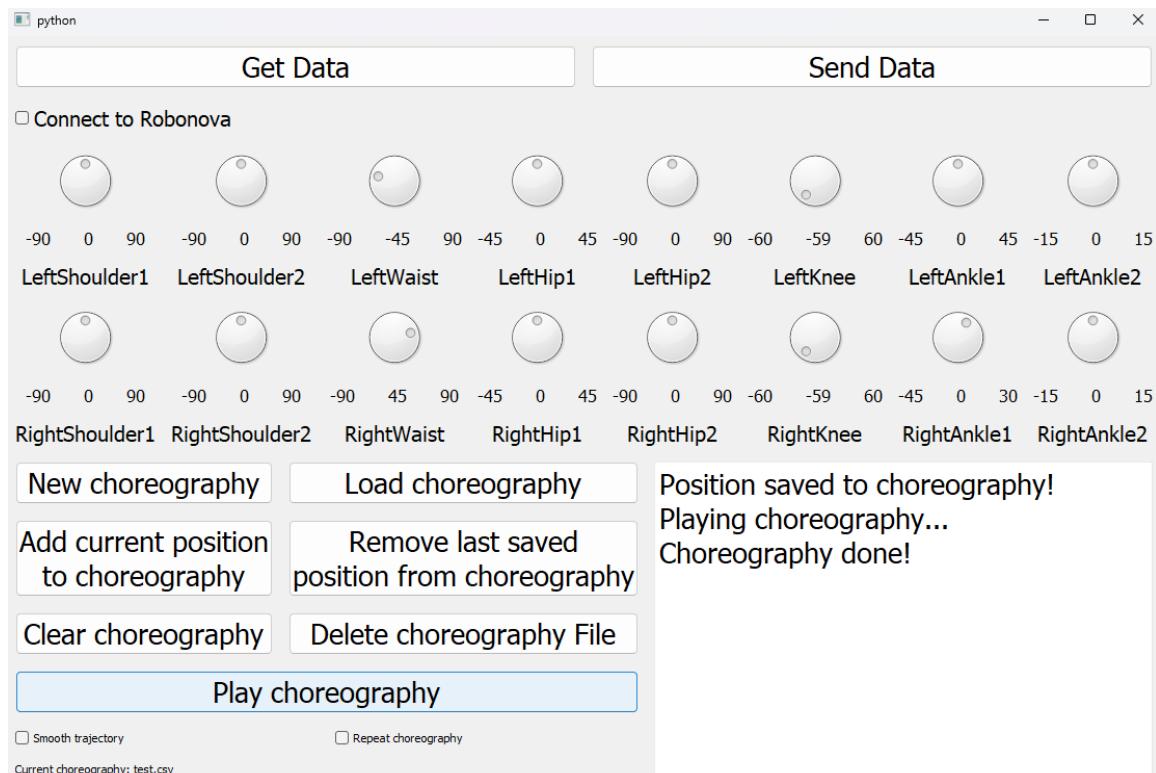


Figura 23: Primera versión de la interfaz de usuario del programa de control del Robonova

## Simulación

Para lograr un programa de control similar a Choregraphe mostrado en la sección 2.4 se requiere una manera de visualizar al Robonova dentro del programa. Para ello, se decidió que el mejor camino sería renderizarlo en un Motor físico. Esto permitiría ver una aproximación al movimiento real del Robonova además de la facilidad de creación de rutinas sin necesidad de tenerlo conectado.

Existen bastantes motores físicos con buena documentación en una amplia cantidad de lenguajes de programación. Lo primero a considerar en que utilizar fue su fácil integración con python para poder conectar la simulación con la interfaz de control fácilmente. Esto reduce las opciones a *Gazebo* y *pyBullet*. Al final se decidió utilizar pyBullet debido a que la librería integrada al motor físico cuenta con bastantes herramientas para el análisis cinemático y dinámico de robots. Por el momento no es necesario nada más que un control de sus juntas basado en cinemática directa (lo cual también es posible en *Gazebo* pero, si se desea expandir el alcance de esta tesis a incluir el sistema dinámico del robot varias funciones ya incluidas en pyBullet de dinámica y cinemática inversa podrían ser de utilidad. Estas funciones además de el uso de cuerpos rígidos articulados para su renderización lo vuelven la primera opción para aplicaciones en robótica.

La construcción de la simulación en pyBullet del robot puede separarse en tres etapas: inicialización del mundo, inicialización del robot y control del robot dentro de la simulación.

La inicialización del mundo consiste en conectar la simulación a la interfaz gráfica de pyBullet y reiniciar la simulación para asegurar que esta inicie correctamente. Además de esto, también se hacen configuraciones de cámara de la interfaz de usuario, configuraciones físicas como las aceleraciones y de simulación como el tiempo entre cuadros de la simulación. Para finalizar, se carga un plano para hacer el piso de la simulación. Este mundo inicial puede ser observado en 24.

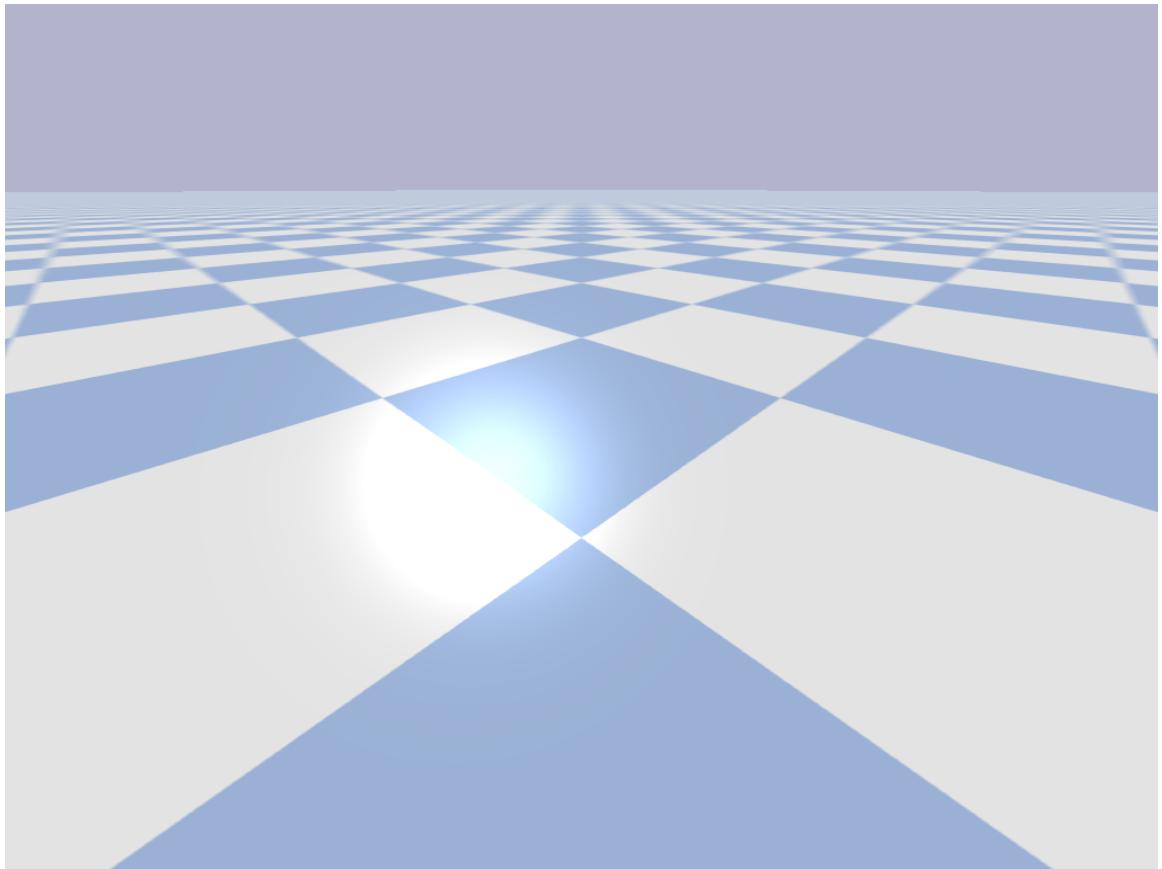


Figura 24: Mundo Inicial en pyBullet

Una vez iniciado el mundo, se carga el modelo del robot. Este puede ser observado en 25. En este caso, se cargó el URDF del Robonova. Se puede extraer la información del robot (número de juntas, nombres de las juntas, tipos de juntas, límites, etc.) utilizando las funciones de pyBullet *getNumJoints* y *getJointInfo*. Esta información es utilizada por el programa para determinar automáticamente las juntas y sus límites y crear las diales necesarias para el control de las juntas. Esto permite que la simulación pueda ser utilizada con cualquier URDF que se introduzca, no solo el Robonova.

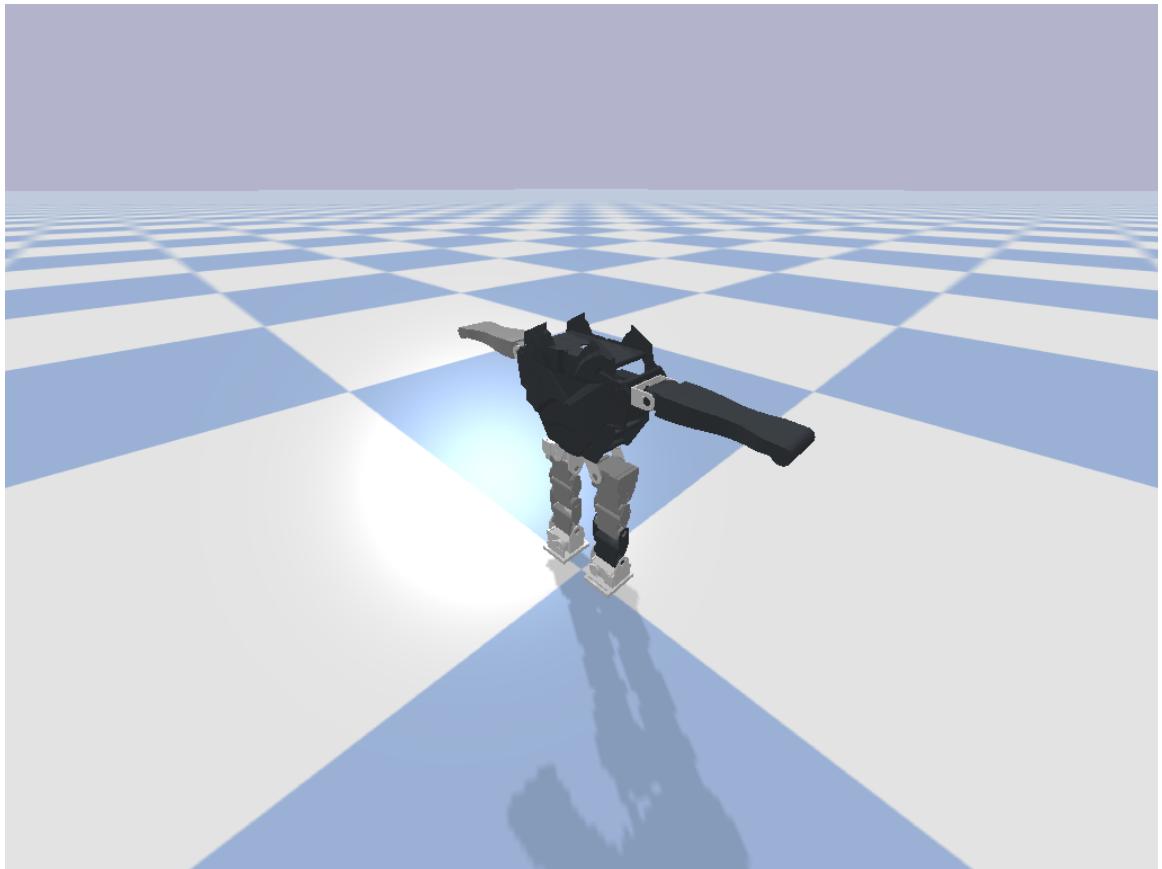


Figura 25: Modelo de Robonova en pyBullet

Por último, el control del robot dentro de la simulación se lleva a cabo con la función `setJointMotorControlArray`. Al trabajar con un robot humanoide con juntas a base de servomotores, se introduce a esta función un *array* con las posiciones angulares deseadas de los servomotores. Para poder observar la simulación a lo largo del tiempo se crea un bucle infinito con la función `stepSimuation` y la anteriormente mencionada de `setJointMotorControlArray` para poder observar y cambiar las posiciones de las juntas del robot. Una imagen del Robonova con las posiciones de sus servomotores ya manipuladas se puede observar en 26.



Figura 26: Simulación Robonova con posiciones de los servomotores cambiada en pyBullet

## Rutinas

Una vez se ha logrado el correcto funcionamiento de la interfaz y la simulación, el único elemento restante de este programa es la creación de rutinas básicas. Para ello, se utiliza la interfaz gráfica del programa, activando las funciones definidas en la sección 8.1.

La rutina que se trabajó en esta sección es una rutina básica de caminata del Robonova. Como se observa en la figura 27, se agregaron varias posiciones a la rutina *walking.csv*. Estas posiciones fueron calibradas cambiando las posiciones de los diales en la interfaz hasta que se visualizara la posición deseada en la simulación del Robonova. Una vez obtenida esta posición, se presionó el botón de guardar y se prosiguió a cambiar a la siguiente posición. Las posiciones fueron determinadas en base a lo discutido en la sección 6.1.1 acerca del ciclo de Gait de un ser humano.

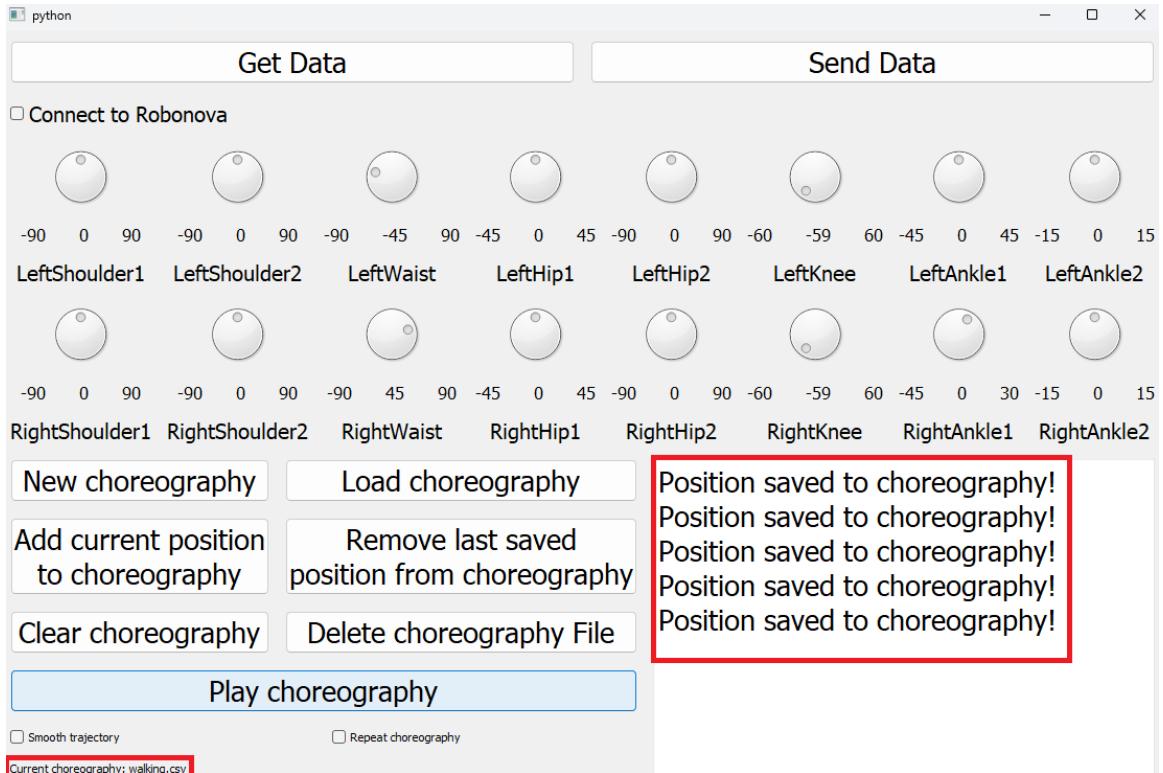


Figura 27: Interfaz luego de cargar posiciones a una rutina

Luego de posicionar al Robonova en los puntos deseados para la rutina, esta puede ser activada presionando el botón *Play Choreography*.

### 8.1.2. Resultados

En la figura 28 se observa como se ven la interfaz y la simulación a nivel de usuario. En otras palabras, el programa final que este utilizaría. Al trabajar en una sola ventana con botones bien etiquetados y un cuadro de texto para informar al usuario de las salidas y los errores, se observa una interfaz limpia y amigable, bastante similar a la de Choregraphe. Al cambiar las posiciones de las diales, el usuario puede observar los cambios instantáneamente en la simulación, dándole la retroalimentación visual de que cambios está causando en el robot incluso sin necesidad de tenerlo conectado de nuevo emulando correctamente a Choregraphe.

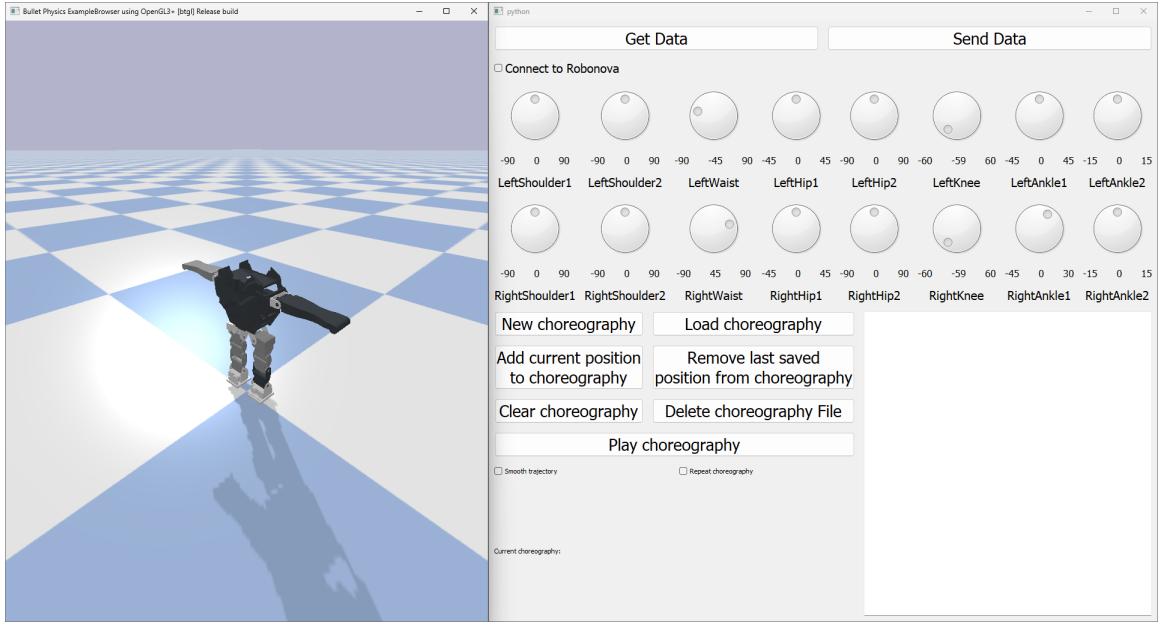


Figura 28: Captura de pantalla del programa y la simulación

Además de esto, el programa cuenta con una rutina de ejemplo llamada *walking* donde el Robonova da dos pasos. Esta rutina se puede activar y repetir indefinidamente para hacer que el Robonova camine en linea recta. Este movimiento se observa en la Figura 29.



Figura 29: Captura de pantalla de la simulación corriendo la rutina *walking*

## 8.2. Segunda Versión del programa

Una vez logrado el funcionamiento correcto de la primera versión de la interfaz gráfica, se iteró sobre esta para crear con funcionalidades más complejas además de ser más amigable con el usuario al momento de crear y editar rutinas el cual es uno de los principales objetivos de esta tesis. El resultado final se puede observar en la figura 30

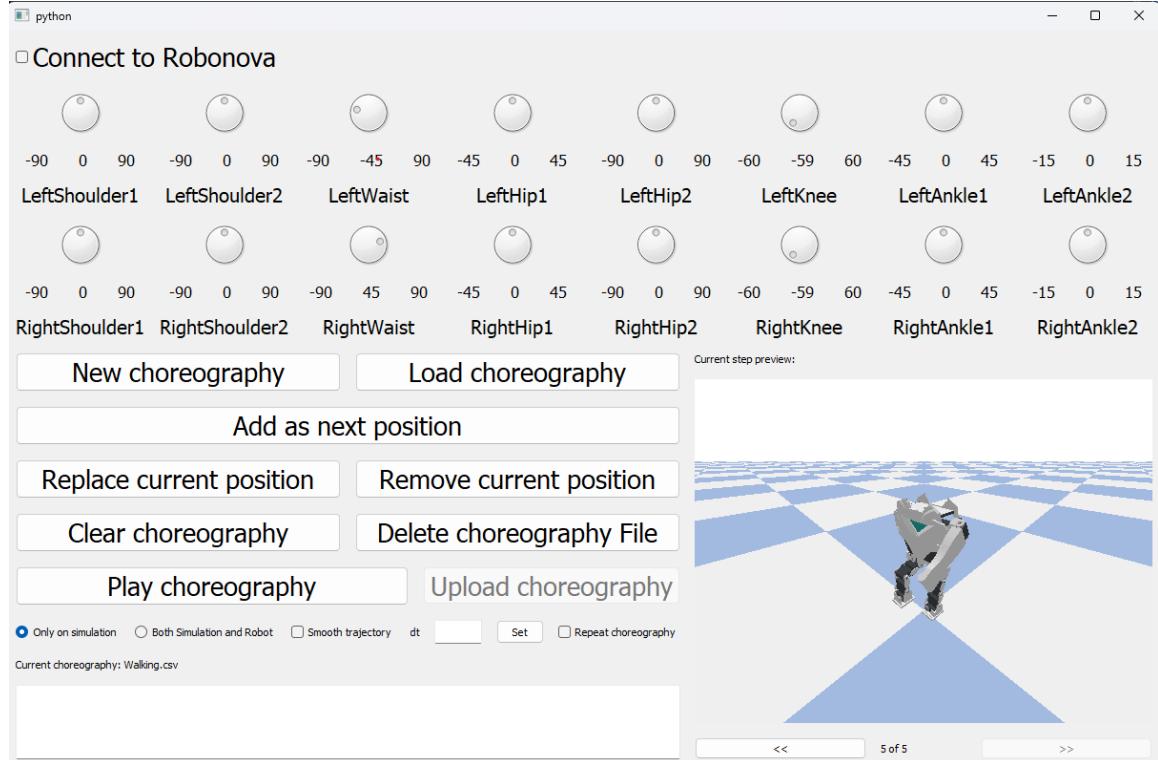


Figura 30: Segunda versión de la interfaz de usuario del programa de control del Robonova

### 8.2.1. Metodología

#### Interfaz gráfica

Como se puede observar comparando las Figuras 23 y 30, los dos programas se enfocan en cumplir los objetivos planteados de control del Robonova y la creación de rutinas con el programa en la Figura 30 ampliando sobre la primera iteración.

Comparando las figuras de arriba hacia abajo lo primero que se observa es la falta de los botones *Get data* y *Send data*. Como se describió en la sección anterior, estos botones tenían como funcionamiento guardar una posición específica del Robonova y cargarla a la simulación, respectivamente. Durante el proceso de rediseño se tomó la decisión de remover estos botones. Al comparar el programa con los vistos en los antecedentes como 2.4 y 2.1 se observó que estos no cuentan con esta funcionalidad. Al final fueron removidos ya que, con los botones para creación y modificación de coreografía, esta funcionalidad era redundante.

Siguiendo la comparación, el otro punto importante que se le adicionó a la segunda versión de la interfaz fue el cuadro de imágenes. De la mano con los botones que se encuentran justo abajo del mismo, este espacio muestra las diferentes posiciones que se utilizaron para crear las rutinas del Robonova facilitando así su diseño y modificación. La programación de esta sección consistió de 3 etapas: renderización y guardado de imágenes, programación para acceder y mostrar las imágenes en la interfaz gráfica y por último la actualización de los botones de control de rutinas para la integración de las imágenes en el programa.

Durante la actualización de los botones se modificaron las funciones (y se agregaron unas nuevas). Los cambios fueron los siguientes:

- *getData* (Removida)
- *sendData* (Removida)
- *connectRobonova* (Sin modificación)
- *updateDial* (Sin modificación)
- *newCoreo* (Modificada)

Además de crear el archivo .csv como se describió anteriormente, al crear el mismo ahora se crea un directorio con el mismo nombre. Este directorio se utiliza como *path* donde se guardará el archivo .csv de la rutina además de archivos de imágenes .png de los diferentes pasos de la rutina. Esto se logró utilizando la librería de *os* de Python y la variable del nombre del archivo que se estaba creando. Además de esto, al crear una nueva rutina, el espacio para imágenes ahora muestra la etiqueta *No preview* (Figura 31) ya que no hay posiciones guardadas en la rutina en este momento. Esto se logró adicionar a la interfaz creando dos *widgets QLable* más de la librería PyQt5.

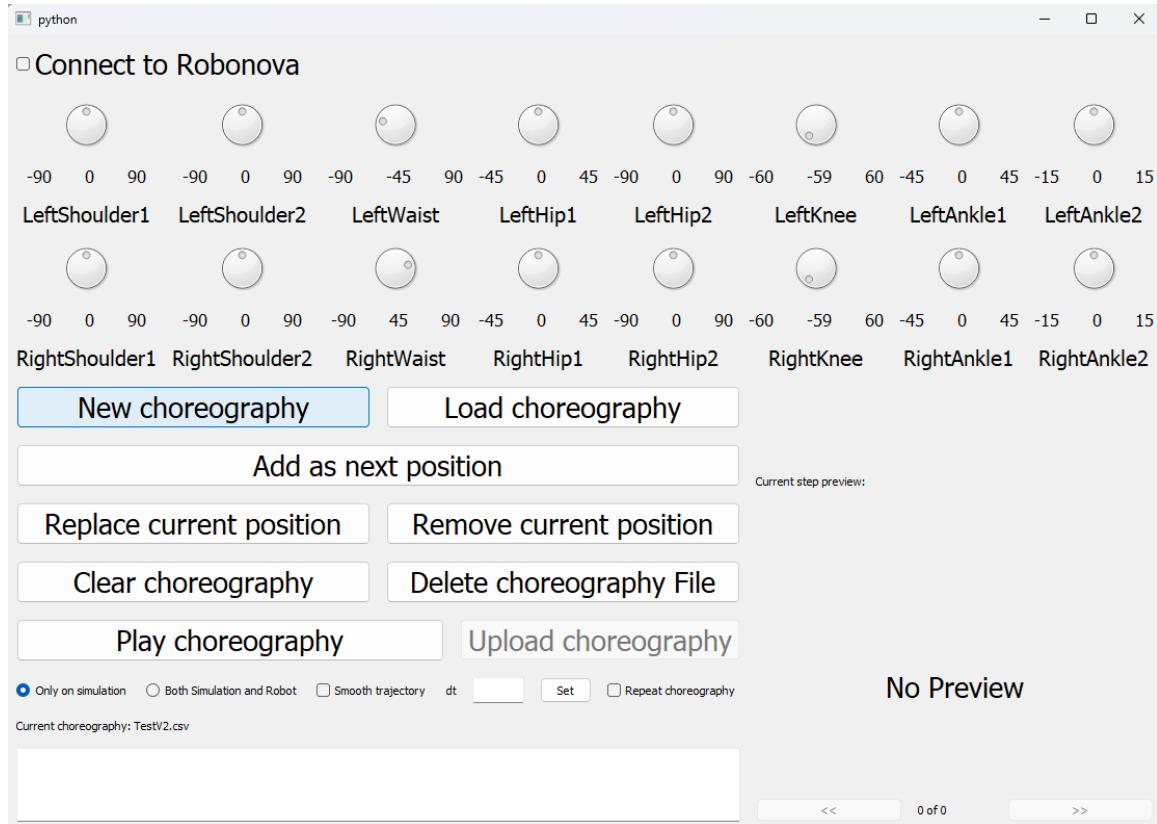


Figura 31: Interfaz de usuario al crear una nueva rutina

- *loadCoreo* (Modificada)

Al igual que la función anterior, esta función fue modificada para tomar en cuenta las imágenes de las posiciones de la rutina. Para ello además de abrir el *QFileDialog* como se explicó en la sección anterior, también se crearon variables en base al número de líneas del archivo .csv y funciones para modificar los *QLabels* de el número de pasos y mostrar las imágenes que se encuentran en el directorio.

- *addCoreo* (Modificada) (Nuevo nombre: *addPosition*)

Esta función tuvo que reprogramarse completamente. Esto se debe a que, en la versión anterior, consistía de utilizar la función *append* al archivo .csv para agregar la posición actual a una nueva fila del archivo. Ya que se deseaba que se deseaba que se pudieran agregar posiciones no solo al final de la rutina, este camino no serviría. Para lograr lo deseado, primero se leyó el archivo .csv y se guardaron los valores en una variable tipo lista. Luego utilizando la función *insert* de Python y una variable que lleva el conteo de en que posición se encuentra actualmente, se agregó la posición actual como la siguiente en el listado. Por último, se reescribió el archivo .csv con la lista actualizada. Además de esto, la simulación renderiza una imagen de la posición actual y la guarda en el directorio de la coreografía.

- *removeCoreo* (Modificada) (Nuevo nombre: *removePosition*)

A esta función solamente se le agregó la funcionalidad para reducir la cuenta de las variables que llevan el paso actual y el total de pasos de la coreografía además de

utilizar la librería *os* de Python para eliminar la imagen del paso que se está borrando y reescribir el nombre de las siguientes para mantener la continuidad.

- *replacePostion* (Nueva)

Muy similar a la función *addPosition* pero ahora, en lugar de insertar una nueva posición a la lista, simplemente remplaza la posición actual (tanto en la lista como en la imagen).

- *clearCoreo* (Modificada)

Al ser un botón que tiene posibilidad de hacer cambios no deseados e irrecuperables a las coreografías, ahora cuando se presiona el botón *Clear choreography* aparece un *pop-up* para confirmar si se desea proseguir con la función. A parte de esto, la funcionalidad sigue siendo la misma que la de

- *deleteCoreo* (Modificada)

Al igual que la función anterior, se agregó un *pop-up* para confirmar que si se desea eliminar por completo la rutina.

- *playCoreo* (No modificada)

- *whereToPlay* (Nueva)

Debajo del botón *Play Choreography* se agregaron dos botones para seleccionar en donde se desea observar la rutina: solo en la simulación o, una vez se encuentre conectado, en el Robonova real. Dependiendo de la selección que se haga, se levanta una bandera que es usada en 9 para llevar a cabo la rutina en el robot.

- *uploadCoreo* (Nueva)

Conectada al botón *Upload choreography*. Como lo dice su nombre, esta función carga la coreografía entera a el ESP32 del Robonova-1. Para ello la función lee el archivo .csv donde está guardada la coreografía y arregla los datos en un *array* de dos dimensiones donde una dimensión es la articulación y la otra es el paso en la coreografía. Esto luego es cargado a un diccionario con cada articulación el cual se serializa a un JSON y luego se envía por WiFi a el Robonova-1.

- *Set dt* (Nueva)

Esta función, conectada a los botones *set dt* y *smooth coreography* permite interpolar la rutina para crear una rutina más fluida. Utilizando funciones de la librería *numpy* se toman los datos de la coreografía y se les realiza una interpolación lineal con la cantidad de puntos estipulada. Esta interpolación luego es guardada en la variable de coreografía y puede ser simulada o cargada con las funciones anteriores

- *nextImg* (Nueva)

Utilizando la librería *os*, esta función permite acceder a las imágenes guardadas en el directorio de la coreografía. Aumentando una variable que determina el paso en el que se encuentra muestra la imagen correspondiente.

- *prevImg* (Nueva)

Utilizando la librería *os*, esta función permite acceder a las imágenes guardadas en el directorio de la coreografía. Disminuyendo una variable que determina el paso en el que se encuentra muestra la imagen correspondiente.

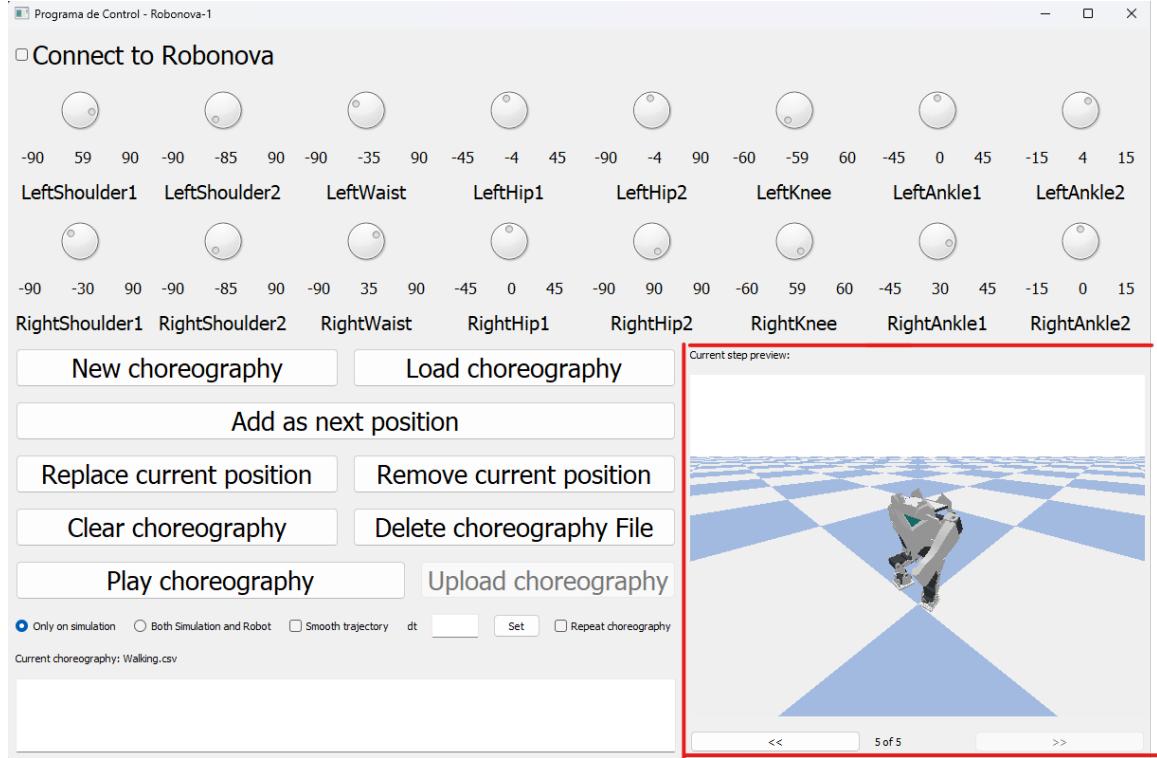


Figura 32: Espacio para imágenes de la pose actual de la rutina

## Simulación

En cuanto a cambios o adiciones en la simulación solamente se agregó la renderización de las imágenes. En [17] se pueden observar todas las funciones de la librería *pyBullet*. Utilizando la función *getCameraImage* se genera un *array* de dimensiones *altoxanchoxRGBA* de la imagen la cual después puede ser guardada o mostrada en la interfaz como se observa en la Figura 32.

Además de renderizar las imágenes cada vez que es agregue o reemplace una posición en la rutina también se llevó a cabo el intento de mantener una renderización constante en la interfaz creada con PyQt5. Esto con el afán de crear un programa más visualmente atractivo y amigable al reducir todo a una sola ventana en lugar de dos como se encontraba en el momento (una para la interfaz de control y otra para la simulación en *pyBullet*). Este intento no fue exitoso ya que si se decide llevar a cabo la simulación de una manera directa (sin interfaz de *pyBullet*), al momento de renderizar la imagen utilizando la función *getCameraImage* esta se renderiza con el CPU en lugar de el GPU que se utiliza cuando si se usa la interfaz de *pyBullet*. La renderización de imágenes sobre todo si se mantienen constantemente es una acción que toma bastantes recursos computacionalmente. Por esta razón cuando se trataba de renderizar todo dentro de una sola interfaz el programa se volvía demasiado lento. También se trató de crear un nuevo hilo donde correría la renderización pero esto no llevó a una mejora lo suficientemente buena para que el programa corriera satisfactoriamente. Por esta razón se decidió mantener las dos interfaces separadas permitiendo así que la GPU rendierice la simulación mientras que el CPU corre el programa.

## Rutinas

En cuanto al diseño de rutinas, utilizando las nuevas funcionalidades de la interfaz como la visualización de los pasos además de poder agregar los pasos no solo al final o reemplazarlos si es necesario se diseñaron más rutinas de prueba. Entre ellas se creó una rutina *hello* donde el robonova saluda al usuario con el brazo izquierdo y la rutina *ArmTest* la cual se utilizó para probar los servomotores de los brazos una vez se conectó al Robonova-1 real.

### 8.2.2. Resultados

## 8.3. Discusión

Como se puede observar con todas las funciones creadas y durante el uso del programa, este es un actualización más que adecuada al programa de control original del Robonova, RoboBASIC.

Además de cumplir con todas las funciones de roboBASIC (control de servomotores por medio de diales, creación y carga de rutinas, enviar posiciones específicas, etc.) el programa también cuenta con características adicionales que facilitan la logística de programación de rutinas del Robonova. Como primer punto se encuentra la simulación en pyBullet incorporada en el programa. Esta simulación permite crear las rutinas y ver una aproximación bastante acertada al movimiento real del Robonova sin necesidad de tener el robot a la mano todo el tiempo. Además de esto, las funciones adicionales agregadas como la repetición de rutina y el suavizado de trayectorias permiten un mayor control en la creación de rutinas tanto para el usuario como para el programador. Al utilizar el suavizador de trayectorias, el usuario puede crear rutinas con pocos puntos de referencia y obtener una trayectoria fluida. De igual forma se puede decidir no utilizar esta opción y generar movimientos más rápidos y explosivos con el Robonova que no eran posibles con el programa de control original. Adicional a esto, la función para cargar las rutinas al Robonova permite hacer modificaciones en tiempo real de las rutinas o cambiar completamente de una rutina a otra sin necesidad de conectar el Robonova al computador como era el caso en el programa original. Por último, la programación de la simulación y la interfaz de usuario se hicieron de forma tan general que permite trabajar con cualquier robot, contingente a que se tenga su URDF y los STLs de sus eslabones.

Gracias a todo esto, podemos determinar que, el programa fue un éxito cumpliendo con todos los objetivos deseados y más.

# CAPÍTULO 9

---

## Control en tiempo real de Robonova

---

### 9.1. Metodología

Con el fin de controlar al robot Robonova de forma inalámbrica, se utilizaron TCP/IP Sockets para conectar el robot como servidor a un cliente en python a través del módulo WiFi de un ESP32 [3]. Una vez creado y cargado el programa de el servidor y control de servomotores en el ESP32 se desea poder controlar al Robonova inalámbricamente desde el programa de Python y poder enviar las rutinas al mismo sin necesidad de conectarlo cada vez que se cree o modifique una de ellas.

### 9.2. Resultados

#### 9.2.1. Programación final de servidor y controlador de servomotores en Arduino

Para crear el servidor se utilizó la librería *Wifi.h* de Arduino. Dentro del programa se guardaron los valores de SSID, contraseña y puerto de conexión a la red wifi utilizada para crear la conexión entre servidor y cliente y se creó una variable tipo *wifiServer*. Dentro del *setup* se conectó el ESP32 a la red WiFi definida anteriormente con la función *WiFi.Begin(ssid,password)*. Luego se inició un bucle que no se detiene hasta que la conexión haya sido establecida para asegurarse que el microcontrolador se conecte antes de proseguir. Una vez conectado a la red WiFi se inicializa el servidor con la función *wifiServer.begin()*. Por último, al estar trabajando con una IP dinámica, se imprime la dirección IP del dispositivo para utilizarla en la configuración del cliente.

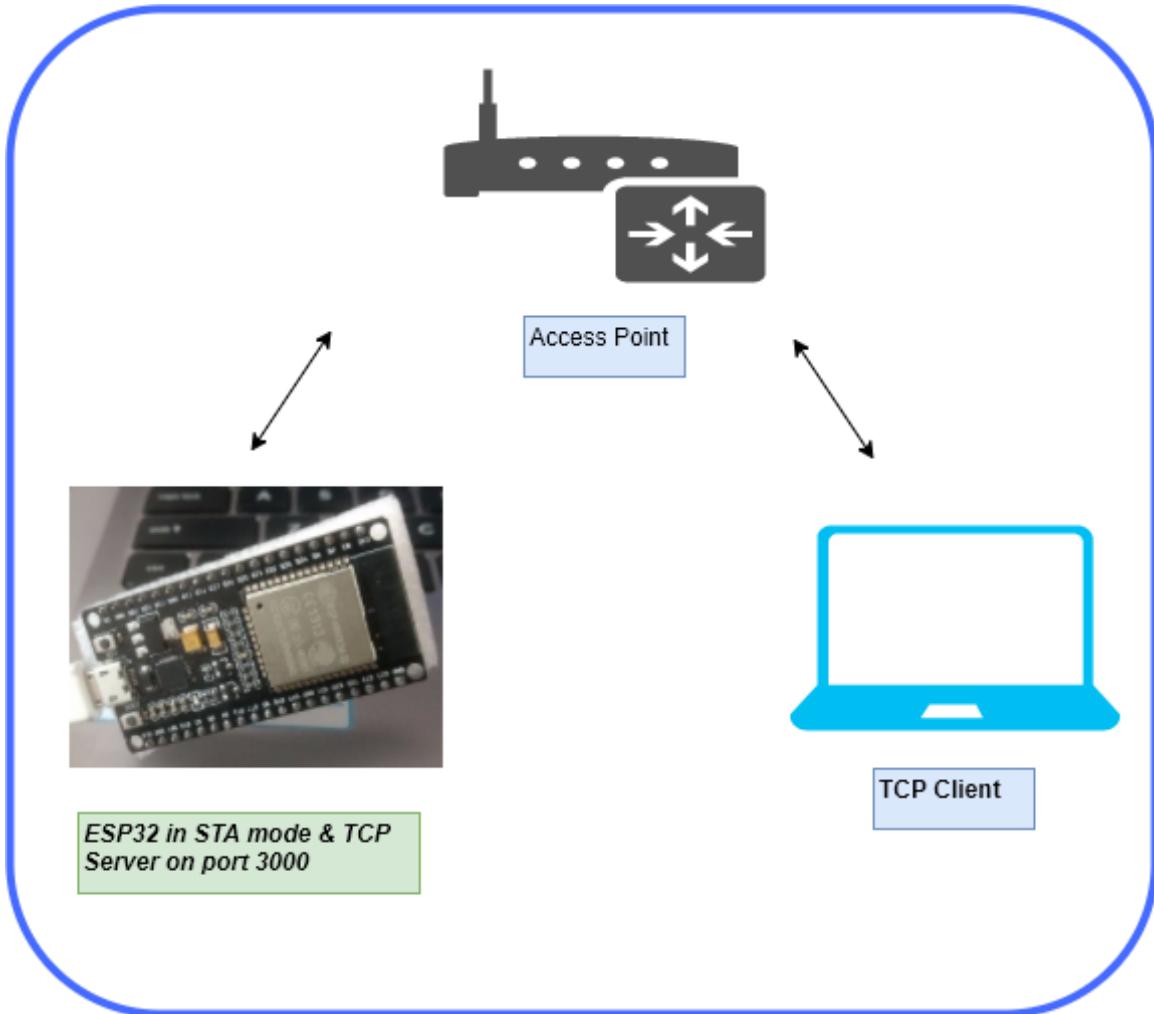


Figura 33: Conexión TCP entre servidor y cliente

Además de contar con el servidor activo, el programa del ESP32 también contiene funciones para extraer las posiciones de los servomotores de un JSON enviado por el cliente. Utilizando la librería *Adafruit\_PWMSServoDriver.h*, el ESP32 envía los valores extraídos del JSON a dos placas controladoras de servomotores para posicionarlos acorde a estos valores.

Para cargar y reproducir las rutinas de movimiento creadas en el programa, el archivo JSON que envía el cliente se modificó dependiendo de la función que se esté llevando a cabo. A demás de enviar las posiciones al momento de hacer un cambio, también se envían dos banderas para determinar las sección del código que se esté trabajando. Estas banderas se activan si el cliente envía la señal para cargar o reproducir una rutina respectivamente.

Si el servidor recibe un mensaje del cliente con ambas banderas apagadas actualiza las posiciones de los servomotores con las recibidas. Esta es una funcionalidad de control directo donde al mover los diales en el cliente, se actualizan las posiciones de los servomotores.

Si el cliente envía un mensaje con la bandera para cargar coreografía (sucede cuando se presiona *Upload Choreography*) entonces se ejecuta una función para guardar el mensaje

en distintas variables tipo array para cada articulación. Estos arrays que son extraídos del JSON son las diferentes posiciones en las que se colocan los servomotores a lo largo de la coreografía.

Por último, si el cliente envía un mensaje con una bandera para reproducir la rutina (sudece al presionar el botón *Play choreography*) se ejecuta una función donde las posiciones de los servomotores cambian de una en una en el orden provisto por los arrays anteriores.

### 9.2.2. Programación final de cliente en python

Debido a que ya existe un programa en python para el control de servomotores (en simulación) y creación de rutinas como se ve en 28 no se creó uno nuevo para la comunicación con el Robonova, solamente se agregó la programación para crear el socket TCP y enviar datos. Para ello se utilizaron las librerías *socket* y *json*. La primera para conectar el cliente (programa en python) a el servidor (ESP32 en el Robonva) y la segunda para crear un archivo .JSON para enviar los datos.

La creación del cliente consiste en la creación de una variable tipo *socket*. Una vez se quiera iniciar la conexión se utiliza la función *connect((ip,port))* de la librería *socket* donde se introducen como argumentos la dirección IP del servidor (que fue determinada en la sección anterior) y el puerto por el cual se hará la conexión. Cabe notar que para la conexión suceda exitosamente tanto el cliente como el servidor deben estar conectados a la misma red.

Una vez conectado al servidor, se envían las posiciones de los servomotores en la simulación a el ESP32 para que este, con su programación incorporada coloque los servomotores en las posiciones deseadas. Para estandarizar la comunicación y facilitar el envío y recepción de los datos, se crea una biblioteca en el programa del cliente que luego es guardada en un archivo .JSON que, como se explicó anteriormente, al llegar al ESP32 es deserializado y utilizado para el posicionamiento de los servomotores. Para evitar errores durante la comunicación y reducir la utilización de recursos, este archivo .JSON únicamente se genera y se envía cada vez que exista un cambio en la posición de alguno de los servomotores del robot.

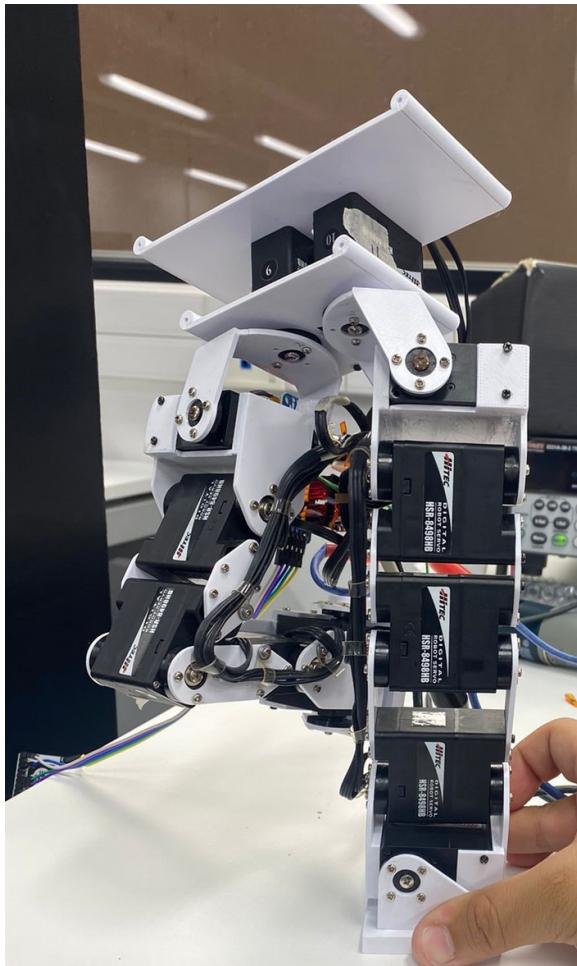


Figura 34: Prototipo de piernas del Robonova siendo controlado programa

### 9.3. Discusión

La principal ventaja de esta conexión entre el programa de control y el Robonova es facilidad de creación, modificación y carga de rutinas al Robonova. Ya que el programa que se encuentra en el ESP32 del nuevo Robonova solamente se encarga de conectarse por WiFi al cliente y controlar los servomotores de a cuerdo a lo que se le envía (es decir, no tiene las rutinas cargadas en su propia memoria), no es necesario conectar el Robonova a la computadora y cargar las nuevas rutinas cada vez que se quiera agregar o modificar una de ellas como era el caso al trabajar con roboBASIC. Además de esto, al encontrarse conectado por Wifi todo el tiempo, también es posible llevar a cabo las demás funciones que se podían hacer en roboBASIC como el control directo de los servomotores sin necesidad de conectar el Robonova o cambiar su programación.

En conclusión, este capítulo se enfocó en como programar y trabajar con el Robonova directamente. Tomando como base todas las funciones de roboBASIC creado por David Buckley, este programa logra llevar a cabo todas las diferentes funciones deseadas para controlar el robot aprovechando los avances en la tecnología y comunicación que se han logrado desde que este salió al mercado.

# CAPÍTULO 10

---

## Conclusiones

---

- Se ha creado una interfaz amigable para el control, programación, monitoreo y la creación de rutinas de la nueva versión del robot humanoide Robonova.
- La conexión en tiempo real al Robonova se logró exitosamente teniendo una latencia baja y, gracias a los servomotores utilizados y programación de control de los mismos, una resolución bastante alta.
- La interfaz creada permite al usuario crear rutinas sencillas y fluidas basado en posiciones específicas deseadas dentro de la misma.
- El agregar una simulación al programa de control del Robonova facilita la logística de programación de rutinas del mismo al no tener que conectarlo con cada edición de rutina.

# CAPÍTULO 11

---

## Recomendaciones

---

- Hacer el modelo dinámico del Robonova para mejorar su emulación de movimientos humanos. Al incluir la dinámica del sistema en el modelo, el Robonova podrá hacer movimientos más complejos de una forma autónoma. Las librerías de pyBullet y Robotics Toolbox incluyen bastantes funciones para facilitar la simulación y análisis de este modelo dinámico.
- Expandir las herramientas de edición de rutinas del programa. Agregar imágenes a la interfaz y botones para de una posición a la siguiente dentro de una rutina. Hacer esta la posición .<sup>a</sup>ctiva" que pueda ser reemplazada por la que se deseé colocar. Esto sería ampliando lo que se tiene en el programa hasta ahora que es solamente agregar o eliminar posiciones al final de la rutina que se está trabajando.

## CAPÍTULO 12

---

### Bibliografía

---

- [1] D. Buckley. “RoboBasic-v2.2.61.” (5 de feb. de 2015), dirección: <http://davidbuckley.net/DB/RoboNova/RoboBasic-v2.10.htm#Appendix1> (visitado 15-04-2023).
- [2] D. Grunberg, R. Ellenberg, I. Kim, J. Oh, P. Oh e Y. Kim, “Development of an Autonomous Dancing Robot,” 2010.
- [3] Boston Dynamics. “Atlas™,” Boston Dynamics. (), dirección: <https://www.bostondynamics.com/atlas> (visitado 29-04-2023).
- [4] Aldebaran Robotics. “NAO the humanoid and programmable robot | Aldebaran.” (), dirección: <https://www.aldebaran.com/en/nao> (visitado 30-09-2023).
- [5] Aldebaran Robotics. “Menus, Panels and Toolbar in a glance — Aldebaran 2.1.4.13 documentation.” (), dirección: <http://doc.aldebaran.com/2-1/software/choregraphe/interface.html> (visitado 02-10-2023).
- [6] Aldebaran Robotics, *Aldebaran Choregraphe Webinar*, 8 de dic. de 2014. dirección: <https://www.youtube.com/watch?app=desktop&v=2byYfgypse8> (visitado 02-10-2023).
- [7] C. Quental, J. Folgado, J. Ambrósio y M. Silva, “A simple controller to overcome the lack of correlation between forward and inverse dynamic analysis of human motion tasks,” *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics*, vol. 230, oct. de 2015. DOI: 10.1177/1464419315608336.
- [8] S. S. Ekka. “Gait definition, its phases & abnormal gait.” Running Time: 59 Section: Health & Fitness. (16 de oct. de 2016), dirección: <https://physiosunit.com/gait-definition-phases-of-cycle-explained/> (visitado 20-05-2023).
- [9] M. Zea, *Cinemática de robots de base flotante, fuerzas de contacto y el modelo de fricción de Coulomb*, 2023.
- [10] N. Kofinas, “Forward and inverse kinematics for the NAO humanoid robot,” 2012.
- [11] K. Erbatur y O. Kurt, “Natural ZMP trajectories for biped robot reference generation,” *IEEE Transactions on Industrial Electronics*, vol. 56, n.º 3, págs. 835-845, mar. de 2009, ISSN: 0278-0046. DOI: 10.1109/TIE.2008.2005150. dirección: <http://ieeexplore.ieee.org/document/4633623/> (visitado 22-05-2023).

- [12] F. Moro, L. Sentis, J. Park et al., “Whole-Body Control [TC Spotlight],” *IEEE Robotics & Automation Magazine*, vol. 24, págs. 12-14, 1 de sep. de 2017. DOI: 10.1109/MRA.2017.2722258.
- [13] Meena Sealathan. “RoboNova Manual-Eng-V1.50.pdf.” (), dirección: [http://davidbuckley.net/DB/RoboNova/RoboNova\\_files/RoboNova%20Manual-Eng-V1.50.pdf](http://davidbuckley.net/DB/RoboNova/RoboNova_files/RoboNova%20Manual-Eng-V1.50.pdf) (visitado 06-05-2023).
- [14] Superrobotica. “Robonova 1 robot humanoide.” (25 de mayo de 2023), dirección: <http://www.superrobotica.com/robonova.htm> (visitado 04-10-2023).
- [15] HITEC. “MR-C3024 Spec English Version 1.pdf.” (), dirección: [http://davidbuckley.net/DB/RoboNova/RoboNova\\_files/MR-C3024%20Spec%20English\\_Version%201.pdf](http://davidbuckley.net/DB/RoboNova/RoboNova_files/MR-C3024%20Spec%20English_Version%201.pdf) (visitado 03-10-2023).
- [16] HITEC. “General Specifications of HSR-8498HB.” (29 de mayo de 2023), dirección: [http://davidbuckley.net/DB/RoboNova/RoboNova\\_files/HSR-8498HB\\_GENERAL\\_SPECIFICATION\\_050623.pdf](http://davidbuckley.net/DB/RoboNova/RoboNova_files/HSR-8498HB_GENERAL_SPECIFICATION_050623.pdf) (visitado 04-10-2023).
- [17] E. Coumans e Y. Bai, “PyBullet quickstart guide,” 2022.
- [18] P. Corke y J. Haviland, “Not your grandmother’s toolbox – the robotics toolbox re-invented for python,” en *2021 IEEE International Conference on Robotics and Automation (ICRA)*, Xi’an, China: IEEE, 30 de mayo de 2021, págs. 11 357-11 363, ISBN: 978-1-72819-077-8. DOI: 10.1109/ICRA48506.2021.9561366. dirección: <https://ieeexplore.ieee.org/document/9561366/> (visitado 21-05-2023).
- [19] “JSON (JavaScript Object Notation).” (11 de ago. de 2022), dirección: <https://www.loc.gov/preservation/digital/formats/fdd/fdd000381.shtml> (visitado 30-09-2023).
- [20] “Working with objects - javascript.” (25 de sep. de 2023), dirección: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_objects) (visitado 04-10-2023).
- [21] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” Internet Engineering Task Force, Request for Comments RFC 8259, dic. de 2017, Num Pages: 16. DOI: 10.17487/RFC8259. dirección: <https://datatracker.ietf.org/doc/rfc8259> (visitado 30-09-2023).
- [22] MATLAB. “URDF Primer - MATLAB & Simulink.” (2023), dirección: <https://www.mathworks.com/help/sm/ug/urdf-model-import.html> (visitado 30-09-2023).
- [23] Josh Newmans. “Getting ready for ROS part 7: Describing a robot with URDF,” Articulated Robotics. (25 de oct. de 2021), dirección: <https://articulatedrobotics.xyz/ready-for-ros-7-urdf/> (visitado 30-09-2023).
- [24] Rhoban. “Onshape-to-robot documentation.” (2020), dirección: <https://onshape-to-robot.readthedocs.io/en/latest/> (visitado 01-09-2023).
- [25] The Qt Company Ltd. “Qt for Python.” (2023), dirección: <https://doc.qt.io/qtforpython-6/> (visitado 01-09-2023).
- [26] Danielle Bodnar. “¿qué es TCP/IP?” ¿Qué es TCP/IP? (4 de jun. de 2021), dirección: <https://www.avg.com/es/signal/what-is-tcp-ip> (visitado 04-10-2023).
- [27] ESPRESSIF. “ESP32-DevKitC Board I Espressif.” (2023), dirección: <https://www.espressif.com/en/products/devkits/esp32-devkitc> (visitado 30-09-2023).

- [28] VISHNU MOHANAN. “DOIT ESP32 DevKit v1 wi-fi development board – pinout diagram & arduino reference – CIRCUITSTATE electronics.” (20 de dic. de 2022), dirección: <https://www.circuitstate.com/pinouts/doit-esp32-devkit-v1-wifi-development-board-pinout-diagram-and-reference/> (visitado 04-10-2023).

## CAPÍTULO 13

---

### Anexos

---

Para hacer referencia al código de este programa vaya al enlace: <https://github.com/pap19836/Tesis>

# CAPÍTULO 14

---

## Glosario

---

**CAD** *Computer Asisted Desing.* 30

**Motor físico** Software de computadora que provee una simulación aproximada de sistemas físicos como dinámica de cuerpos rígidos, dinámica de fluidos, etc. Usualmente utilizada en la creación de videojuegos.. 35

**OnShape** Programa gratuito CAD basado en el navegador.. 19

**PLA** El ácido poliláctico, también conocido como PLA, es un monómero termoplástico derivado de fuentes orgánicas renovables como el almidón de maíz o la caña de azúcar. 19

**PWM** *Pulse Width Modulation.* 17

**XML** *Extensible Markup Language.* 21