UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Aprendizaje por Refuerzo Seguro para enseñar a un robot humanoide a caminar más rápido

Autor: Daniel Acera Bolaños
Tutor: Fernando Fernández Rebollo
Director: Francisco Javier García Polo

Leganés, 15 de julio de 2013

Agradecimientos

Tras años de continuo trabajo, con este proyecto concluye una etapa importante en mi vida. Por ese motivo, me gustaría agradecer especialmente el apoyo de mi familia que han estado ayudando y apoyando durante este tiempo.

Además, agradecer a mis compañeros de clase los buenos momentos que hemos pasado durante estos años, y con especial agradecimiento, a mis compañeros de prácticas, Cristian y Fran, por ayudarme y soportarme durante estos años.

Por último, agradecer a todos los profesores que durante estos años me han hecho aprender cosas nuevas y especialmente, al tutor y al director de este proyecto, que me han ayudado en la finalización de esta etapa.

Resumen

Enseñar a un robot humanoide a caminar es un problema abierto y desafiante. Los comportamientos clásicos de caminar habitualmente requieren la puesta a punto de muchos parámetros de control (longitud de paso, velocidad, frecuencia, etc). Encontrar una configuración inicial o básica de estos parámetros no es complicado, pero optimizarla para un objetivo (por ejemplo, caminar rápido) no es tan sencillo, ya que puede hacer caer al robot humanoide provocando daños, en caso de una optimización incorrecta. En este proyecto, se propone usar técnicas de aprendizaje por refuerzo seguro para mejorar el comportamiento de caminar de un robot humanoide que permite caminar más rápido que la configuración predefinida. El aprendizaje por refuerzo seguro asume la existencia de una política segura que permite aprender una nueva, la cual se representa con un enfoque basado en casos. Los algoritmos de aprendizaje por refuerzo seguro aplicados son *PI-SRL* (***P**olicy **I**mprovement throught **S**afe **R**einforcement **L**earning*) y *PR-SRL* (***P**olicy **R**euse for **S**afe **R**einforcement **L**earning*).

Palabras Clave:

Caminar, *NAO*, PISRL, PRSRL, Aprendizaje por Refuerzo Seguro, ROS, NAOqi, NAOsim, simulación, robot humanoide

Abstract

Teaching a humanoid robot to walk is an open and challenging problem. Classical walking behaviors usually require the tuning of many control parameters (step size, speed, frequency, etcetera). To find an initial or basic configuration of such parameters could not be so hard, but optimizing them for some goal (for instance, to walk faster) is not easy because, when defined uncorrectly, may produce the fall of the humanoid, and the consequent damages. In this paper we propose the use of Safe Reinforcement Learning for improving the walking behavior of a humanoid that permits the robot to walk faster than with a pre-defined configuration. Safe Reinforcement Learning assumes the existence of a safe policy that permits the humanoid to walk, and probabilistically reuse such policy to learn a new one, which is represented following a case based approach. The Safe Reinforcement Learning algorithms used are, *PI-SRL* (***P****olicy* ***I****mprovement throught* ***S****afe* ***R****einforcement* ***L****earning*) y *PR-SRL* (***P****olicy* ***R****euse for* ***S****afe* ***R****einforcement* ***L****earning*).

Keywords:

Walking, *NAO*, PISRL, PRSRL, Safe Reinforcement Learning, ROS, NAOqi, NAOsim, simulation, Humanoid Robot

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

En este capítulo se introduce la motivación del proyecto, se definen los objetivos en la realización del trabajo y se muestra la estructura general de este documento.

1.1. Introducción

En la actualidad, los robots son utilizados en múltiples ámbitos de la vida cotidiana, y siendo objeto de diversas líneas de investigación. Una de ellas es buscar comportamiento rápidos y estables para tareas en entornos complejos.

Los comportamientos que realizan los robots son muy variados, como pueden ser desplazarse, coger objetos, visualizar el entorno u orientarse en una habitación, entre otros. En los robots humanoides, el comportamiento de caminar es uno de los temas de investigación más interesantes y una importante área de aplicación para múltiples campos.

Las técnicas de aprendizaje automático han sido utilizadas para mejorar múltiples comportamientos de los robots, pero en el caso del caminar bípedo se disponen de múltiples aplicaciones. Por ejemplo, Mereçli y Veloso [1] utilizan un proceso de caminar bípedo formado por dos fases basado en el aprendizaje por demostración. Este método aprende correcciones de movimientos basado en una retroalimentación correctiva proporcionada por el humano, mientras camina automáticamente usando un algoritmo simplificado del caminar. Por otro lado, Farchy et al.[2] proponen un algoritmo optimizado iterativo para andar rápido, pero este proceso necesita aprender un comportamiento en el simulador, probarlo en el robot real, modificar el simulador con las correctas imperfecciones y así sucesivamente, lo cual requiere mucho tiempo.

Por otro lado, el comportamiento de caminar bípedo ha sido tratado con

técnicas de aprendizaje por refuerzo, por ejemplo para generar un patrón optimizado de caminar[3], o para caminar rápido y estable[4]. Sin embargo, todos estos comportamientos no contemplan explícitamente el riesgo de caída del robot, lo cual puede ser un gran riesgo para la integridad del robot.

Para contemplar este riesgo de caída en los comportamientos de caminar bípedo se introducen los algoritmos de aprendizaje por refuerzo seguro, que permiten explorar entornos peligrosos de forma segura y eficiente[5]. En este trabajo se utilizarán dos técnicas de aprendizaje por refuerzo seguro, como son *PI-SRL*[6] (***P**olicy **I**mprovement throught **S**afe **R**einforcement **L**earning*) y *PR-SRL*[7] (***P**olicy **R**euse for **S**afe **R**einforcement **L**earning*).

El robot utilizado es el robot *NAO*, desarrollado por la empresa francesa *Aldebaran Robotics*. Es un robot humanoide que dispone de un comportamiento predefinido de caminar de forma bípeda, y que realiza un movimiento estable y omnidireccional.

En este proyecto, este comportamiento predefinido de caminar será mejorado mediante la aplicación de técnicas de aprendizaje por refuerzo seguro (*PI-SRL* y *PR-SRL*), las cuales permitirán obtener caminatas más rápidas basándose en dicho comportamiento predefinido.

Por último, para el desarrollo de este comportamiento se dispone de un sistema operativo o librería para robots, denominada *ROS* (**R**obot **O**perating **S**ystem), que será la utilizada en este proyecto.

1.2. Objetivos

El objetivo de este proyecto es la utilización de técnicas de aprendizaje por refuerzo seguro para mejorar el comportamiento de caminar predefinido en el robot *NAO*. Esta mejora consiste en obtener un comportamiento más rápido pero asegurando la estabilidad del robot, es decir, evitando que el robot sufra caídas. Para obtener esta mejora, se utilizará el comportamiento de caminar predefinido en el robot *NAO*.

Para ello se realizará un software que permita la aplicación de los algoritmos utilizando el comportamiento predefinido en el robot *NAO*, ya sea en simulación o con el robot real. El desarrollo se realizará sobre el software para robots *ROS*, detallado en la sección 2.7.

Para cumplir el objetivo principal se han definido otros objetivos parciales:

- Estudiar la plataforma *ROS*, para conocer las librerías y mecanismos de abstracción disponibles para el desarrollo de comportamientos en robots.

- Estudiar el robot *NAO*, para conocer los mecanismos disponibles para la redefinición del comportamiento de caminar.

- Analizar paquetes disponibles en *ROS*, que tengan relación con algoritmos de aprendizaje y con el robot *NAO*.

- Diseñar la arquitectura, incluyendo la definición de los algoritmos de aprendizaje adaptándolos al comportamiento de caminar del robot *NAO*.

- Implementar la arquitectura diseñada, utilizando las librerías de *ROS*.

- Realizar experimentos que permitan evaluar y comparar el funcionamiento de los algoritmos de aprendizaje por refuerzo seguro en el comportamiento de caminar del robot *NAO*.

- Validar que los algoritmos utilizados obtienen un comportamiento que permite al robot *NAO* caminar más rápido que con sus comportamientos predefinidos.

1.3. Estructura del documento

En esta sección se detalla la estructura utilizada para la realización de este documento. Ésta contiene 6 capítulos, que se describen a continuación.

1. **Introducción y objetivos**: en este primer apartado se expone la motivación para la realización de este proyecto y los objetivos establecidos para el proyecto.

2. **Estado de la cuestión**: en este apartado se describe el contexto en el que se encuentra este proyecto. En primer lugar, describe la evolución de la robótica desde su origen. A continuación, se describe el robot *NAO* y sus comportamientos de caminar predefinidos. Después, se detallan los algoritmos de aprendizaje por refuerzo seguro, *PI-SRL* (***P**olicy **I**mprovement throught **S**afe **R**einforcement **L**earning*) y *PR-SRL* (***P**olicy **R**euse for **S**afe **R**einforcement **L**earning*), que son los utilizados en este proyecto. Por último, se describe la plataforma *ROS*.

3. **Arquitectura para aprender el comportamiento de caminar de un robot *NAO***: en este apartado se describe la estructura general y diseño de la arquitectura.

4. **Experimentos**: en este apartado se incluyen los experimentos realizados mediante los algoritmos *PI-SRL* y *PR-SRL* para el comportamiento de caminar, con diferentes configuraciones y su validación en el robot real.

5. **Conclusiones**: en este apartado se incluyen las conclusiones obtenidas con la realización de este proyecto y las líneas futuras de investigación.

6. **Apéndices**: se incluyen 6 apéndices:

   - **Gestión del proyecto**: incluye las fases del proyecto, la planificación, los medios utilizados y el presupuesto del proyecto.
   - **Mensajes y servicios**: incluye la descripción detallada de todos los mensajes y servicios declarados para la comunicación en *ROS*.
   - **Comunicación detallada de los nodos**: incluye la descripción detallada de la comunicación que realizan cada uno de los nodos definidos en *ROS*.
   - **Instalación y configuración del entorno**: describe la instalación y configuración de los diferentes software necesarios.
   - **Simulador *NAOsim***: incluye una breve descripción del simulador *NAOsim* y de su instalación.
   - **Ejecución del software**: detalla los pasos a realizar para ejecutar el software desarrollado.

Capítulo 2

Estado de la cuestión

En este capítulo se describe el marco de trabajo en el cual queda encuadrado este proyecto.

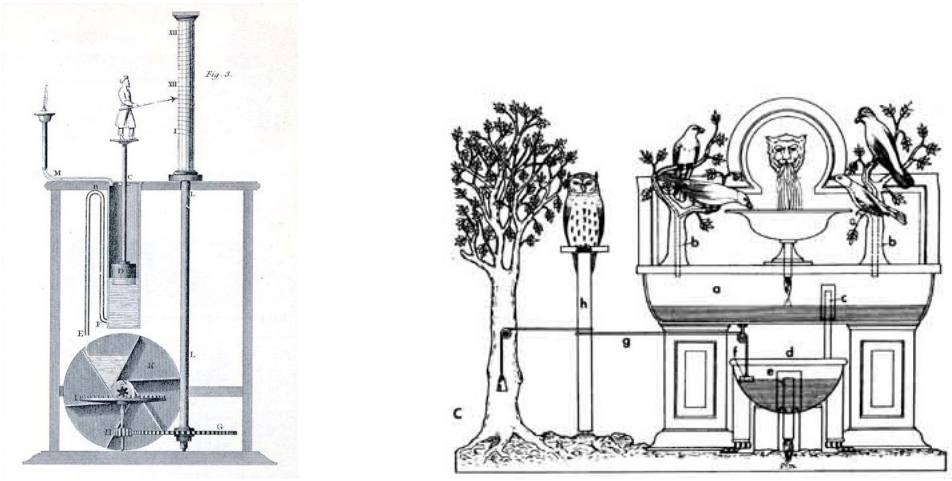
Para ello se comienza con la historia de la robótica desde su origen hasta la actualidad. A continuación, se describe el robot *NAO*, siendo el utilizado en este proyecto. Después, se analiza el mecanismo de caminar del robot *NAO* y los comportamientos base predefinidos. Posteriormente, se introducen los algoritmos de aprendizaje por refuerzo seguro, *PI-SRL* (***P****olicy* ***I****mprovement throught* ***S****afe* ***R****einforcement* ***L****earning*) y *PR-SRL* (***P****olicy* ***R****euse for* ***S****afe* ***R****einforcement* ***L****earning*), que son los utilizados en este proyecto para mejorar los comportamientos bases anteriores. Por último, se introduce el software utilizado para el desarrollo del proyecto, como es *ROS*.

2.1. Historia de la Robótica

La robótica es la rama de la tecnología que se dedica al diseño, construcción, operación y aplicación de los robots. En sus origenes, hace miles de años, se les conocía como autómatas y no es hasta el año 1921, cuando el escritor checo Karel Capek utilizó el término "Robot" por primera vez, que deriva de la palabra checa *robota* que significa trabajo forzoso o servidumbre, en su obra *Rossum's Universal Robots* (*R.U.R*)[8]. En esta obra se utiliza el término "Robot" para denominar a dos seres artificiales con apariencia humana que respondían a las órdenes de su dueño. A partir de esta obra, se comienza a utilizar este término en detrimento de autómatas[10].

Siguiendo con el origen de la robótica, en 1942 Isaac Asimov definió, en su obra *"Runaround"*[9], el término "Robótica" como la ciencia que estudia a los robots. Además, definió las *Tres Leyes de la Robótica*, que definen reglas de comportamiento de un robot hacia un ser humano[11].

El primer autómata se fechan en el año 1500 a.C., siendo una estatua del rey de Etiopía, que emitía sonidos al amanecer cuando los rayos del sol le iluminaban. Entre el 300 y 270 a.C se construye una clepsidra o reloj de agua, figura 2.1a, y un órgano que emite los sonidos por impulsos de agua[10]. En los siguientes años, Herón de Alejandría realiza un tratado de autómatas donde se describen múltiples juguetes capaces de moverse por si solos de forma repetida, como son aves que vuelan, gorjean y beben (figura 2.1b)[10].



(a) (b)

Figura 2.1: (a) Clepsidra o Reloj de agua[15], (b) Aves de Hero de Alejandría[12]

Durante la Edad Media y el Renacimiento, se realizan indicaciones para la construcción de sistemas mecánicos antropomórficos (Villard d'Honnecourt, 1235)[12] y se crea un león mecánico (Leonardo Da Vinci, 1500) que abría el pecho con la garra y mostraba el escudo de armas del rey, Además, Leonardo Da Vinci diseñó un dispositivo mecánico parecido a un caballero mecánico (que nunca fue construido), capaz de incorporarse, agitar los brazos, mover la cabeza (teniendo un cuello flexible) y, abrir y cerrar la mandíbula (figura 2.2a)[10].

En el siglo XVIII, se crearon autómatas que replicaban al ser humano, realizando una serie de movimientos simples. Estas máquinas fueron asumiendo tareas de ayuda al hombre y acabaron repercutiendo en la concepción del mundo y de los seres automáticos. Uno de los más famosos y completos constructores de androides automatizados, Jacques de Vaucanson, realizó un autómata flautista capaz de tocar melodías siguiendo una partitura y un pato mecánico de más de 400 piezas móviles, capaz de graznar y comer de la mano de una persona, completando de forma total la digestión (figura 2.2b).

(a) (b)

Figura 2.2: (a) Caballero mecánico de Leonardo Da Vinci[15], (b) Pato de Jacques de Vaucanson[16]

A finales de este siglo, se aportaron dos conceptos importantes de robótica, como son el automatismo con repetición de una tarea preprogramada y la precisión del mecanismo funcional basado en cilindros o discos giratorios[10].

A principios del siglo XIX, Joseph Marie Jacquard realiza una aportación fundamental a la robótica al diseñar un sistema de funcionamiento automático para telares (figura 2.3). Para ello, se utilizaba un cartón multiperforado que permite tipificar algunas tareas y repetirlas de manera idéntica, incluidas por *IBM* en sus primeras computadoras[10].

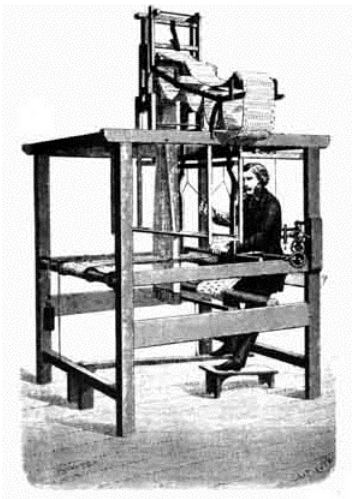
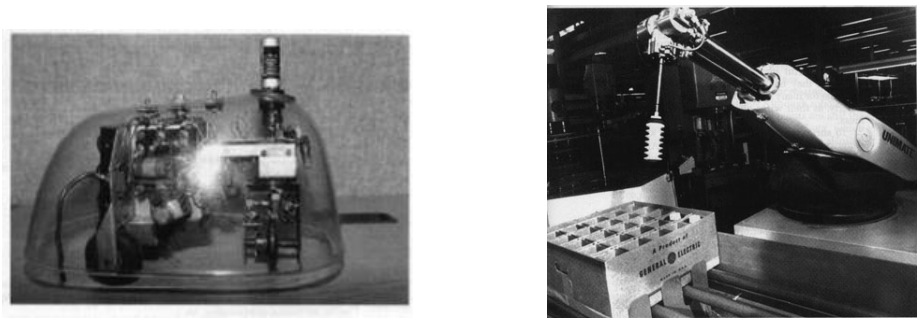


Figura 2.3: Telar de Jacquard[12]

En el primer tercio del siglo XX, se comienza la construcción de los robots modernos debido a los avances científicos y técnicos en el ámbito de las matemáticas y de la física teórica. Ésto unido a los avances en la computación es

el impulso para el desarrollo de máquinas cercanas al ideal de automatismo y autonomía[10] y mediados de siglo se produce el auge de la robótica moderna, construyéndose múltiples modelos de robots. Los hitos más relevantes son[11][14]:

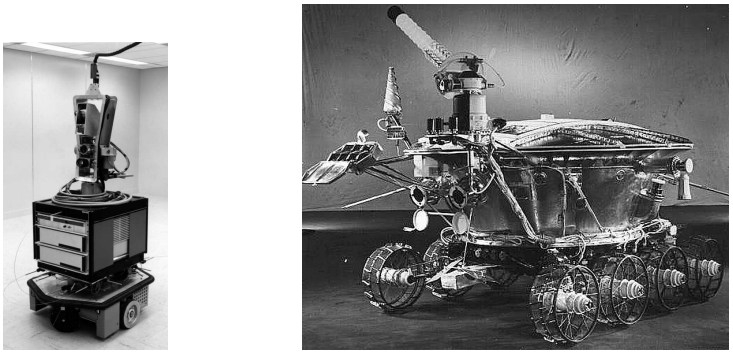
- En el año 1938, el primer brazo articulado (o manipulador) para pintura al spray, lo que representó la introducción de los robots en una cadena de producción.
- En el año 1953, se exhibe un robot con comportamientos biológicos simples, llamado Elsei, siendo los primeros robots electrónicos autónomos (figura 2.4a).



(a) (b)

Figura 2.4: (a) Elsei[15], (b) Unimate[15]

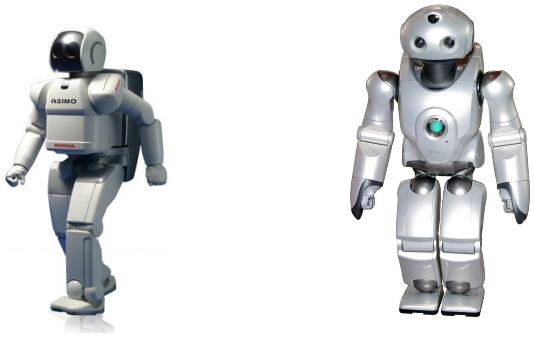
- En el año 1956, George Devol y Joseph Engelberger construyeron un robot llamado Unimate (figura 2.4b). Este robot se trataba de un computador junto con un manipulador, de este modo la máquina era manejada para realizar tareas de manera automática. En 1962, se instaló en una planta de *General Motors*, convirtiéndose en el primer robot industrial.
- En el año 1970, *SRI* International (***S**tanford **R**esearch **I**nstitute*) fabricó el primer robot móvil capaz de realizar sus propias decisiones, mediante la aplicación de inteligencia artificial. El robot se llamaba Shakey (figura 2.5a).
- En el año 1971, la Unión Soviética explora la superficie de la luna con el vehículo Lunokhod 1 (figura 2.5b). Este es el primer robot por control remoto para aterrizar en la Luna.
- En el año 2000, la empresa japonesa Honda Motor, presentó un robot humanoide capaz de caminar de manera bípeda y de interactuar con



(a) (b)

Figura 2.5: (a) Shakey[15], (b) Lunokhod 1[17]

las personas, llamado ASIMO (figura 2.6a). Este robot es resultado del programa de desarrollo e investigación humanoide iniciado en el año 1986[14].



(a) (b)

Figura 2.6: (a) ASIMO[19], (b) QRIO[18]

En los primeros años del siglo XXI, los nuevos robots desarrollados se utilizan en misiones de exploración en otros planetas, como es el caso del robot Spirit y Opportunity, que han sido utilizados para la exploración de la superficie marciana[11]. También, en esos años aparecieron otros robots humanoides, como es el caso del primer robot humanoide comercial completamente autónomo capaz de correr, llamado QRIO (figura 2.6b) y años después el robot *NAO*, detallado en la sección 2.2.

2.2. Robot *NAO*

El contenido de esta sección incluye el origen y los ámbitos de utilización del robot *NAO* en la vida cotidiana actual. Además, incluye una breve descripción de los componentes hardware y software del robot.

2.2.1. Robot *NAO*: Origen

El robot *NAO* es un robot humanoide autónomo y programable desarrollado por la empresa francesa *Aldebaran Robotics*[23]. Su desarrollo comenzó en el año 2004 con el proyecto *NAO*, y que llevó a la producción de la primera versión del robot en el año 2008, tras un total de seis prototipos entre esos años.

Desde esa primera versión, el robot *NAO* se ha ido introduciendo en diferentes ámbitos de la vida cotidiana. El primero de ellos fue en el año 2008, convirtiéndose en el robot oficial de la competición internacional de fútbol para robots, *Robocup* (*Robot Soccer World Cup*). Esta competición promueve la investigación en el ámbito de la robótica y de la inteligencia artificial, y tiene como objetivo la creación de sistemas cooperativos de multi-agentes y multi-robots en entornos dinámicos.

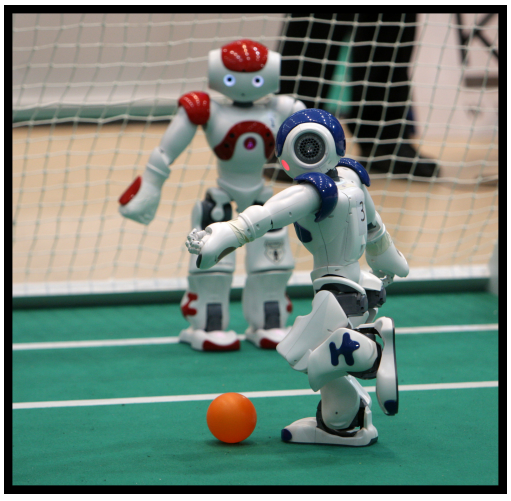


Figura 2.7: Robot *NAO* en la *Robocup*

En ese mismo año, *Aldebaran Robotics* lanza una edición académica del robot para universidades, institutos y laboratorios de investigación. Esta edición orienta al robot al ámbito la investigación, y años más tarde (2011), el código fuente de control del robot se liberó, pasando a ser software de códi-

go abierto. Desde ese año, más de 200 instituciones académicas de todo el mundo utilizan este robot.

En el año 2011, se anuncia el lanzamiento de un nuevo robot *NAO* (*Nao Next Gen*), con mejoras en el hardware y software como la incorporación de cámaras HD, sistemas de anticolisión y mecanismos de caminar a alta velocidad.

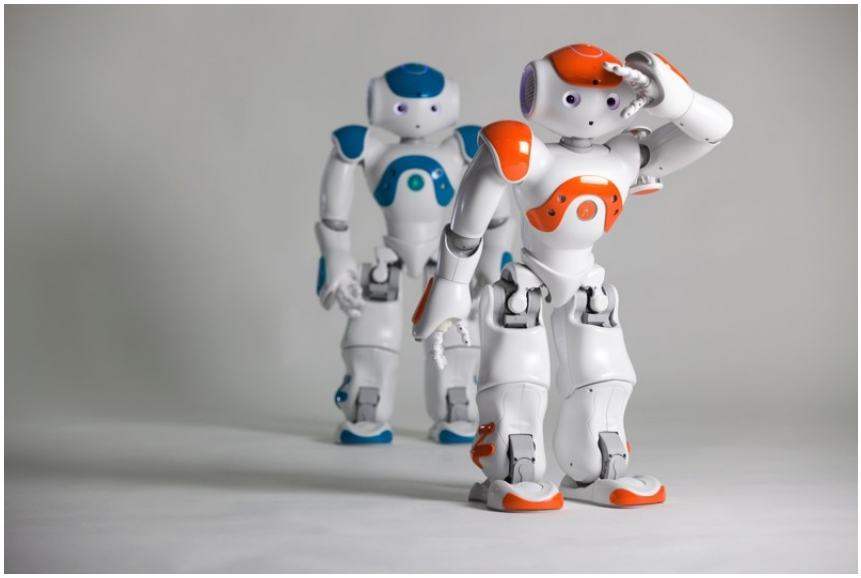


Figura 2.8: Nueva versión del robot *NAO* (*Nao Next Gen*)[25]

En los últimos años se ha incorporado a causas sociales, como es ayudar a niños autistas con la inclusión de juegos y aplicaciones para atraer su atención y ayudarlos a mejorar su sociabilidad[26]. Además, se ha llevado a cabo el proyecto *Romeo*, que introduce al robot *NAO* en la ayuda a personas dependientes[27].

2.2.2. Robot *NAO*: Hadware

El robot *NAO* tiene una altura de 57.3 cm, una anchura de 27.5 cm y una profundidad de 31.1 cm (figura 2.9)[21].

Las especificaciones técnicas del robot son, un procesador *x86 AMD GEODE 500MHz CPU* con 256MB *SDRAM* y 2GB de memoria *flash*, para versión 3.X del robot. En el caso de la versión 4.0 del robot (*Nao Next Gen*), dispone de un procesador *ATOM Z530 1.6GHz CPU* con 1GB de *RAM* y 2GB de memoria flash, disponiendo adicionalmente de 4 a 8GB de memoria dedicada para el usuario. En relación a la batería, permite una autonomía aproximada de 90 minutos para un uso normal y 60 minutos para un uso activo.

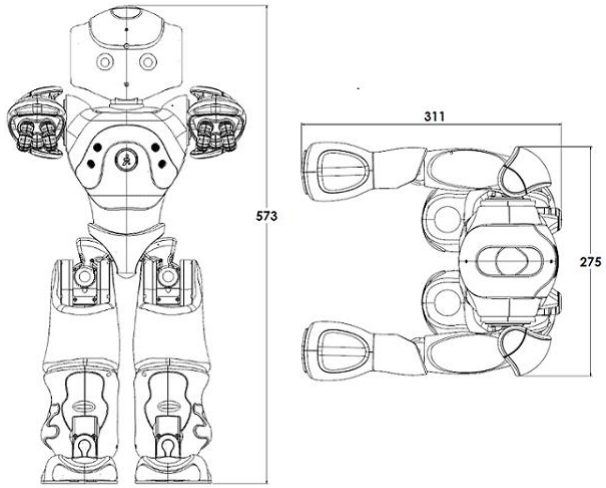


Figura 2.9: Dimensiones del robot *NAO*[21]

En relación a la movilidad de las diferentes partes del robot, dispone de 26 grados de libertad (figura 2.10), que se corresponden con motores eléctricos y actuadores. Estos motores y actuadores se localizan:

- 2 en la cabeza.
- 4 en cada brazo.
- 1 en cada mano.
- 6 en cada pierna.

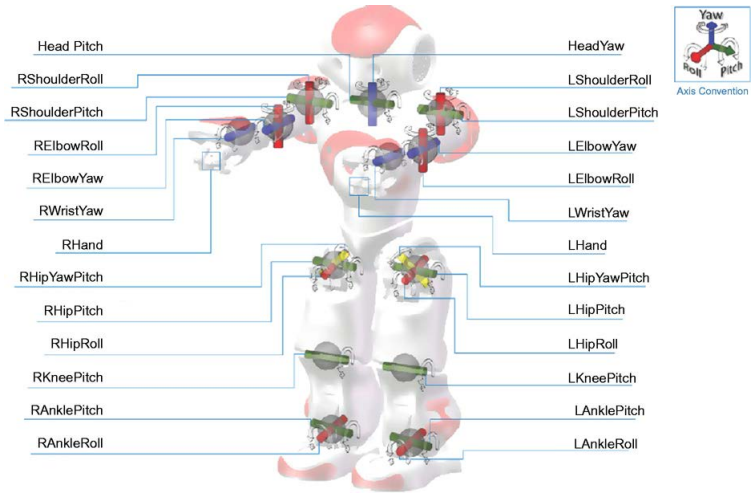


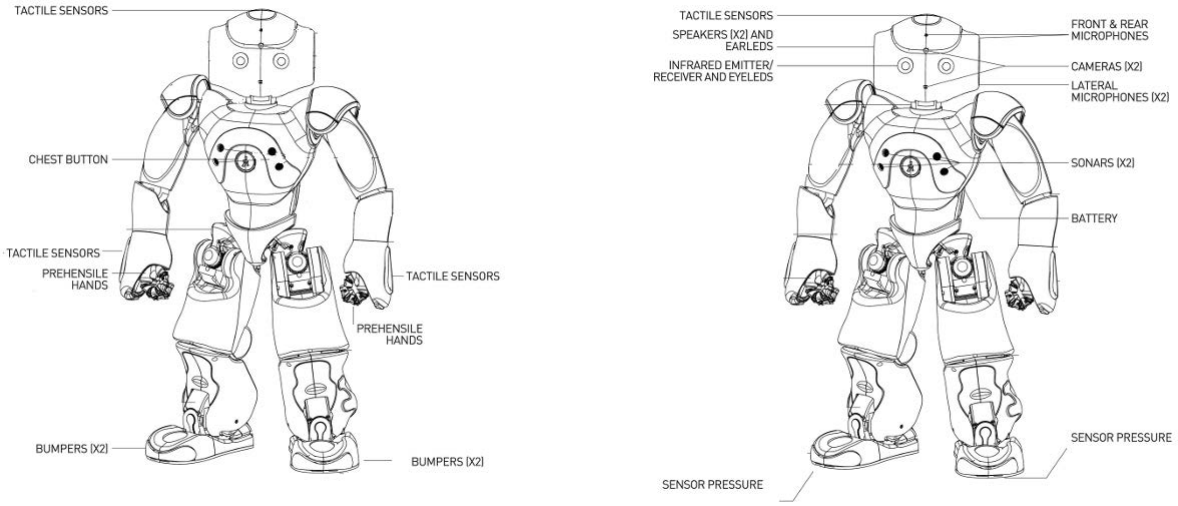
Figura 2.10: Motores y actuadores del robot *NAO*[22]

En relación a la recopilación de información del entorno, dispone de diferentes tipos de sensores (figura 2.11a), como son:

- 14 sensores táctiles o de contacto, 3 de ellos en la cabeza, 3 en cada mano, 2 en cada pie (*bumpers*) y uno en el torso.
- 8 sensores de presión.
- 2 emisores de señales de ultrasonidos (*"sonars"*).
- 2 receptores de señales de ultrasonidos (*"sonars"*).
- Un sistema inercial, compuesto por 2 giroscopios y un acelerómetro, con su propio procesador incorporado en el torso.

Por último, para la interacción entre el robot y una persona física (figura 2.11b), dispone de:

- 2 cámaras.
- 4 micrófonos.
- 2 emisores de señales infrarrojas.
- 2 receptores de señales infrarrojas.
- 2 altavoces.



(a) (b)

Figura 2.11: (a) Sensores táctiles del robot *NAO*. (b) Sensores de presión y elementos de interacción del robot *NAO*[20]

2.2.3. Robot *NAO*: Software

El principal componente software del robot *NAO* es un software embebido llamado *NAOqi* [24]. Este software proporciona un *framework* rápido, seguro, confiable, multiplataforma y distribuido, que permite mejorar las funcionalidades del robot. Además, proporciona una comunicación homogénea entre los diferentes módulos, y una programación que se puede realizar en diferentes sistemas operativos (*Windows*, *Mac OS* o *Linux*) y sobre diferentes lenguajes de programación (*C++*, *Python*, *Urbi* o *.Net*).

Esta librería define 4 principales módulos que permiten la interacción con los elementos hardware del robot:

- ***Motion***: proporciona los métodos que facilitan la realización de movimientos del robot. Además, implementa mecanismos de seguridad en el movimiento, como es manejar las caídas o evitar colisiones.
- ***Audio***: contiene los elementos software relacionados con el audio del robot.
- ***Vision***: contiene los elementos software relacionados con la visión del robot.
- ***Sensors***: contiene los métodos que facilitan el acceso a la información de los sensores del robot *NAO*.

2.3. Comportamiento de caminar en el *NAO*

En este apartado se detalla brevemente los comandos de alto nivel que permiten realizar un movimiento omnidireccional y robusto ante cualquier obstáculo. A continuación, se describe las fases para realizar el movimiento y el algoritmo utilizado por el robot *NAO* para el cálculo de las trayectorias de los pasos.

2.3.1. Comandos de alto nivel

En este apartado se detallan los comandos de alto nivel que realizan el movimiento omnidireccional y robusto del robot *NAO*. Estos comandos son declarados en el módulo *Motion*, comentado en el apartado 2.2.3. Se dispone de 2 maneras de controlar el movimiento[33]:

Las configuraciones predeterminadas para los pasos (*feetGaitConfig*, *leftFootGaitConfig* y *rightFootGaitConfig*) se muestran en la tabla 2.1.

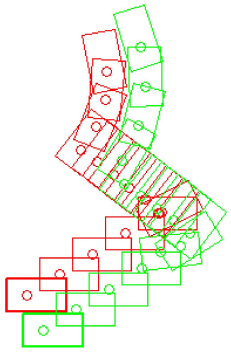


Figura 2.12: Movimiento omnidireccional[33]

- Realiza un movimiento a una posición $< x, y >$ y una orientación en radianes ($theta$), respecto a la actual. Además incluye la configuración de los pasos, $feetGaitConfig$.

walkTo(float x, float y, float theta, ALValue feetGaitConfig)

- Realiza un movimiento acorde a una velocidad normalizada $< x, y, theta >$, una frecuencia de paso $frequency$ y la configuración de los pasos para cada pierna, $leftFootGaitConfig$ y $rightFootGaitConfig$.

  setWalkTargetVelocity(float x, float y, float theta, float frequency, ALValue leftFootGaitConfig, ALValue rightFootGaitConfig)

Nombre (unidad)	Mínimo	Defecto	Máximo
MaxStepX(mt)	0.001	0.040	0.080
MinStepX(mt)	-0.040	-0.040	-0.040
MaxStepY(mt)	0.101	0.140	0.160
MaxStepTheta(rad)	0.001	0.349	0.524
MaxStepFrequency(n,ul)	0	1	1
MinStepPeriod(seg)	0.42	0.42	0.42
MaxStepPeriod(seg)	0.6	0.6	0.6
StepHeight(mt)	0.005	0.020	0.040
TorsoWx(rad)	-0.122	0	0.122
TorsoWy(rad)	-0.122	0	0.122
FootSeparation(mt)	0.1	0.1	0.1
MinFootSeparation(mt)	0.088	0.088	0.088

Tabla 2.1: Valores de los parámetros de caminar del robot *NAO*

Adicionalmente se dispone de dos métodos que permiten un mayor control sobre el movimiento, ya que proporcionan los pasos exactos a realizar por el robot.

- Establecer una lista de pasos, $legName$ y $footSteps$, mediante la utilización de tiempos concretos para su realización, $timeList$. Además se añade una condición para eliminar los pasos anteriores pendientes de ejecución, $clearExisting$.

  setFootSteps(vector <string> legName, ALValue footSteps, vector<float> timeList, bool clearExisting)

- Establecer una lista de pasos, $legName$ y $footSteps$, mediante la utilización de una velocidad normalizada para la realización, *fractionMaxSpeed*. Además se añade una condición para eliminar los pasos anteriores pendientes de ejecución, $clearExisting$.

  setFootStepsWithSpeed(vector<string> legName, ALValue footSteps, vector <float> fractionMaxSpeed, bool clearExisting)

2.3.2. Descripción del proceso de caminar

En el proceso de caminar, cada paso está compuesto por dos fases, *"double leg"* y *"single leg support"* (figura 2.13).

La primera fase (*"double leg"*) se produce cuando el robot dispone de los dos pies apoyados en el suelo. La segunda fase (*"singe leg support"*) representa el movimiento de uno de los pies, es decir, uno de los pies se encuentra en el suelo apoyado y el otro se encuentra en el aire realizando el desplazamiento.

La duración de cada una de las fases con respecto al tiempo de paso es de un tercio para la primera fase (*"double leg"*) y dos tercios para la segunda fase (*"singe leg support"*).

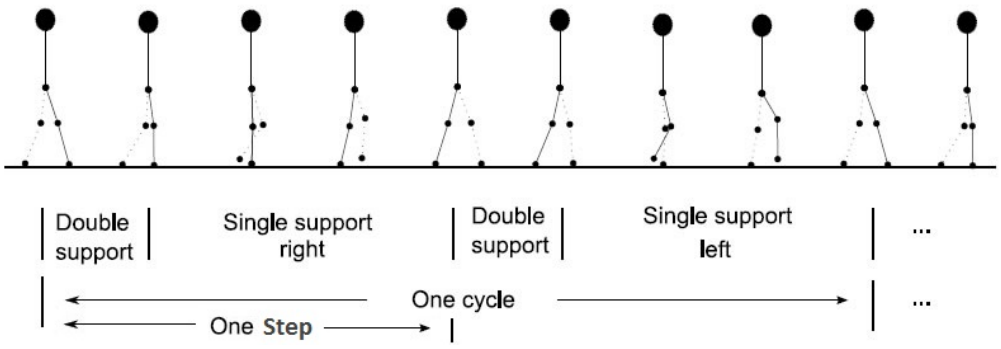


Figura 2.13: Fases en el mecanismo de caminar[30]

2.3.3. Determinación de la trayectoria

El cálculo de la trayectoria de cada paso está basado en el uso de un modelo de péndulo invertido[31] con modificaciones realizadas con programación cuadrática[32]. Para la aplicación de este modelo se describen dos mecanismos, uno de ciclo abierto (figura 2.14) y otro de ciclo cerrado (figura 2.17)[28].

La principal diferencia entre ambos mecanismos es la retroalimentación de los sensores del robot tras la ejecución de la trayectoria. Esta retroalimentación permite realizar un sistema más robusto ante perturbaciones del entorno, como obstáculos en el suelo o desniveles. En resumen, el mecanismo de bucle abierto realiza movimientos más estáticos mientras que el de bucle cerrado, movimiento más dinámicos.

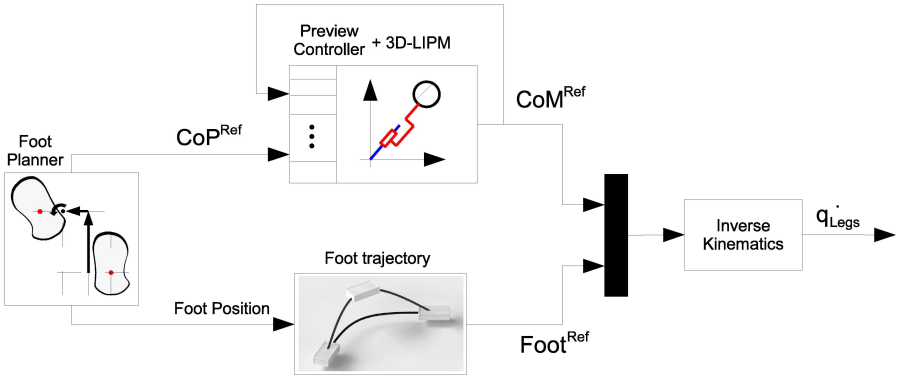


Figura 2.14: Mecanismo de generación de trayectorias con ciclo abierto[28]

El primer mecanismo (figura 2.14) está formado por 4 componentes, *"Foot Planner"*, *"Foot Trayectory"*, *"Preview Controller + 3D-LIPM"* (***Three-Dimensional Linear Inverted Pendulum Mode***) e *"Inverse Kinematics"*.

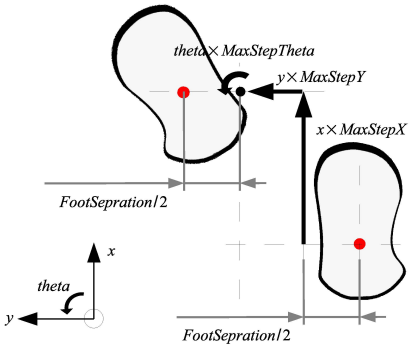


Figura 2.15: Planificador de la trayectoria de un paso[28]

El primer componente (*"Foot Planner"*), se encarga de obtener la mejor colocación de los pies de manera independiente del comando de alto nivel utilizado. En relación a los comandos descritos en la sección 2.3.1, para el comando *walkTo* y *setWalkTargetVelocity* se encarga de obtener la mejor colocación para los pasos (figura 2.15) y en el caso de los comandos *setFootSteps* y *setFootStepsWithSpeed*, se encarga de recortar los pasos para asegurar que un movimiento elíptico y sin colisiones.

Este componente utiliza un conjunto de parámetros en la planificación que permiten delimitar los pasos (tabla 2.1), teniendo en cuenta las capacidades físicas del robot.

Como se observa en la figura 2.14, este componente genera dos salidas:

- **Trayectoria del centro de presión (CoP^{Ref})**, también denominado *ZMP* (***Z**ero **M**oment **P**oint*), que se corresponde con el punto de contacto entre los pies del robot y el suelo (figura 2.16). El CoP se mueve al siguiente soporte durante la primera fase del paso ("double leg") y en la segunda fase del paso ("single leg support") se coloca en medio del área de soporte del pie.

- **Trayectoria del pie ($Foot^{Ref}$)**, la posición cartesiana de los pies calculada a través de una interpolación cúbica, usando un *spline* cúbico, que permite garantizar una trayectoria suave respecto a la velocidad lineal y de rotación.

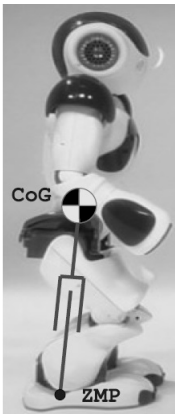


Figura 2.16: Localización del centro de masa (CoG) y del centro de presión (ZMP)[29]

El componente *"Preview Controller + 3D-LIPM"*, es el encargado de calcular la trayectoria del centro de masa (CoM^{Ref}), utilizando las trayectorias

del centro de presión (CoP^{Ref}) obtenidas por el *"Foot Planner"* y con la reutilización de las trayectorias calculadas con anterioridad.

El cálculo de la trayectoria está basada en el modelo de péndulo lineal inverso ($LIPM$)[31], que proporciona una aproximación general para computar el centro de presión (ZMP o CoP) teniendo en cuenta la inercia producida por el movimiento de las diferentes partes del cuerpo. Al modelo $LIPM$ se le incorpora el concepto de *Preview Controller*, que se encarga de anticipar la localización del siguiente paso. Para ello utiliza muestras de la trayectoria del CoP para computar la posición del CoM. Para resolver este problema se utiliza programación cuadrática[32].

Por último, el componente (*"Inverse Kinematics"*) es el encargado de reflejar las trayectorias de los pies, obtenidas por los componentes anteriores, en el robot aplicando la intensidad necesaria a los actuadores.

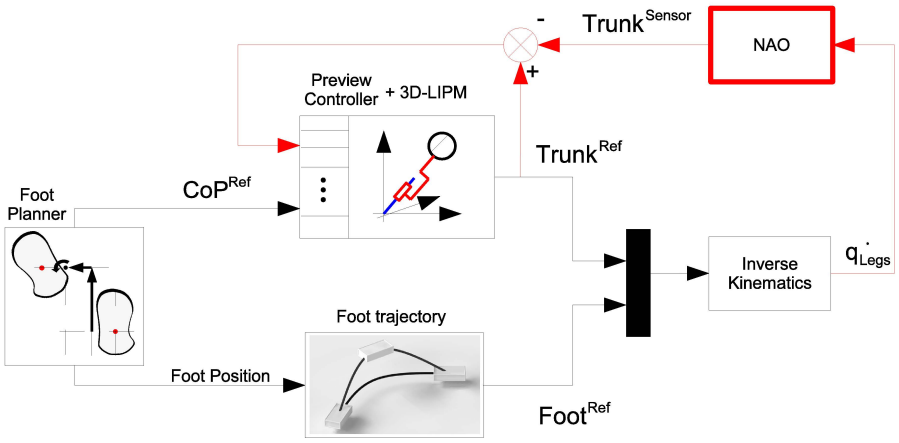


Figura 2.17: Mecanismo de generación de trayectorias con ciclo cerrado[28]

Los componentes descritos hasta este punto, se encuentran en ambos algoritmos de cálculo de trayectoria. En el caso del bucle cerrado (figura 2.17), se basa únicamente en la posición real del tronco basada en la información aportada por los sensores de las articulaciones. Con esta posición, la principal idea es la inclusión del error de la posición del tronco en el cálculo de trayectorias ($Trunk^{error} = Trunk^{Ref} - Trunk^{Sensor}$), pero no es sencillo debido a defectos mecánicos del robot, así que se incluye un umbral al error para mitigar estos defectos.

Por último, cabría indicar que en este proyecto se aplica técnicas de aprendizaje automático para seleccionar los pasos, realizando una función similar al componente *"Foot Planner"*. Para ello se utiliza la parametrización máxima y por defecto mostrada en la tabla 2.1.

2.4. Aprendizaje automático

El aprendizaje automático es una rama de la Inteligencia Artificial, cuyo objetivo es la resolución de problemas mediante la utilización del conocimiento adquirido en problemas resueltos con anterioridad y similares al actual.

2.4.1. Tipos de algoritmos

En el aprendizaje automático se distinguen diferentes tipos de algoritmos que se agrupan en función de la salida que producen. Estos algoritmos son:

- **Aprendizaje supervisado**: se dispone de ejemplos de entrenamiento donde se conoce la entrada y la salida que produce, siendo el principal objetivo de este tipo de aprendizaje obtener la función que sea capaz de predecir la salida ante cualquier entrada válida. Esta función puede producir una salida numérica en un problema de regresión, o discreta en un problema de clasificación.

- **Aprendizaje no supervisado**: en este tipo de aprendizaje no se dispone de conocimiento anterior, es decir, se dispone de un conjunto de entrada pero se desconoce la salida que produce. Esta salida es la que se determina mediante la semejanza de las entradas. Un ejemplo es la agrupación o *clustering*.

- **Aprendizaje semi-supervisado**: es una combinación de los dos tipos anteriores, se dispone de un conjunto de datos etiquetados (aprendizaje supervisado) y otro conjunto sin etiquetar (aprendizaje no supervisado).

- **Aprendizaje por refuerzo**: es un aprendizaje basado en prueba y error y que consiste en aprender a decidir, ante cualquier situación determinada, la acción más adecuada para lograr el objetivo.

En este proyecto nos centraremos en la utilización de técnicas de aprendizaje por refuerzo y más concretamente en técnicas de aprendizaje por refuerzo seguro.

2.4.2. Aprendizaje por Refuerzo

En esta sección se describe brevemente el aprendizaje por refuerzo, introduciendo un algoritmo de este tipo de aprendizaje como es *Q-Learning*.

El aprendizaje por refuerzo consiste en aprender a decidir, ante una situación determinada, qué acción es la más adecuada para lograr un objetivo.

Este tipo de aprendizaje utiliza un proceso iterativo de prueba y error basado en señales de refuerzo[34].

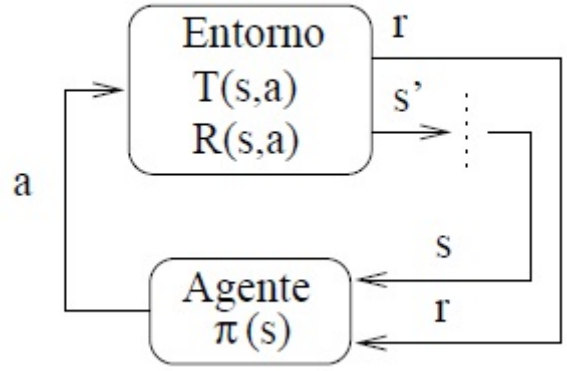


Figura 2.18: Proceso de Aprendizaje por refuerzo[34]

Los problemas de aprendizaje por refuerzo son definidos mediante *MDP* (**M**arkov **D**ecision **P**rocess). Estos procesos están formados por una tupla $< S, A, T, R >$:

- S es el conjunto de todos los posibles estados.
- A es el conjunto de todas las posibles acciones.
- T es la función de transición de estados.
- R es la función de refuerzo para cada par estado-acción.

Los *MDPs* cumplen la propiedad de Markov, es decir, el estado anterior y la última acción son suficientes para describir el estado actual y el refuerzo recibido. Esto implica que la acción a ejecutar solo depende del estado actual.

Uno de los principales algoritmos de aprendizaje por refuerzo es *Q-Learning*. Este algoritmo realiza un aprendizaje por prueba y error cuyo objetivo es aprender la política de acción que maximice el refuerzo medio esperado. Para ello se dispone de una tabla Q formada por los estados (las filas) y las acciones (las columnas), donde cada valor, $Q(s, a)$, se corresponde con lo bueno que es la acción a para el estado s. Este valor se calcula mediante la ecuación 2.1.

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')] \tag{2.1}$$

En la utilización de estas técnicas surge un problema entre la exploración y la explotación, es decir, determinar cuándo se termina de explorar el espacio de estados y de acciones, y se comienza a explotar las opciones disponibles. Esta elección afecta al comportamiento del algoritmo, ya que en el caso de que se inicie la explotación antes de tiempo, puede no obtenerse la mejor política debido a la falta de exploración del espacio de acciones.

Por otro lado, se dispone de un problema de generalización en los espacios de estados continuos o de gran tamaño. Para resolver este problema, se puede utilizar la discretización del espacio de estados, la cual puede romper fácilmente la propiedad de Markov, en caso de discretizaciones erróneas, y además produce un problema en el número de regiones.

En resumen, los algoritmos de aprendizaje por refuerzo introducen dos principales problemas, uno es el de exploración y explotación y otro es el de generalización de estados continuos. En este proyecto, se manejan entornos continuos y peligrosos por lo que una discretización del espacio de estados incorrecta puede producir daños al robot. Esto nos lleva a la utilización de los algoritmos de aprendizaje por refuerzo seguro.

2.4.3. Aprendizaje por Refuerzo Seguro

El aprendizaje por refuerzo tiene el objetivo de obtener una política que permita a un agente realizar su objetivo de manera óptima independientemente del estado del entorno. La obtención de la política mediante estos algoritmos típicos pueden ocasionar daños al agente en entornos continuos y peligrosos, por esta razón, se introducen los algoritmos de aprendizaje por refuerzo seguro.

Los algoritmos de aprendizaje por refuerzo seguro proporcionan mecanismos en la exploración del entorno que aseguran la elección de acciones seguras, para evitar daños en el agente. Una situación típica de estos algoritmos es la figura 2.19.

En la figura 2.19, se muestra un problema de navegación manejado por el algoritmo de aprendizaje por refuerzo seguro. En ella se observa como se dispone de un estado inicial (*start*) y de una meta u objetivo (*goal*). Para realizar dicha transición se muestra tres diferentes trayectorias producidas por un comportamiento base (las líneas continuas). A continuación, se aplica una pequeña variación a las trayectorias base, lo que produce trayectorias diferentes a las que se dispone (las líneas discontinuas azules), produciendo en algunos casos (las líneas discontinuas rojas) que se salgan del espacio explorado por el comportamiento base (la región dentro del cuadrado). Cuando ésto ocurre, se intenta devolver la trayectoria a la zona segura mediante la utilización del comportamiento base. En otros casos, la trayectoria no se pue-

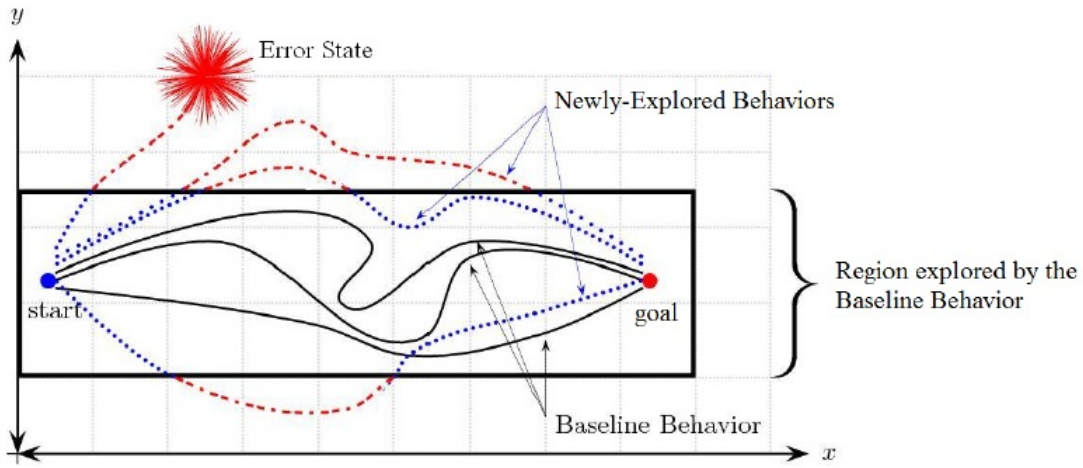


Figura 2.19: Situaciones del entorno aplicando aprendizaje por refuerzo seguro[6]

de recuperar ya que se ha encontrado un estado de error, como puede ser la caída del robot para el comportamiento de caminar.

Tras la descripción de una situación de exploración segura, se procede a la descripción de conceptos utilizados por los algoritmos de *RL* seguro.

Estados conocidos y desconocidos

Un concepto importante en la utilización de los algoritmos de aprendizaje por refuerzo seguro es la definición de estado conocido y desconocido.

Cuando estamos en un espacio de estados y de acciones continuos, los estados están compuestos por n valores reales (n-dimensional), $s = (s_1, s_2, , s_n)$ y las acciones están compuestas por m valores reales (m-dimensional), $a = (a_1, a_2, , a_m)$. A la representación del dominio, el agente incluye una memoria o base de casos para el almacenamiento de la experiencia, representados en pares estado-acción o casos. Estos casos se forman por una tupla estado-acción, que ha experimentado el agente anteriormente y un valor asociado al estado, $c =< s_i, a_i, V(s_i) >$.

Cuando se recibe un nuevo estado, s_q, se busca el vecino más cercano en la base de casos del agente, utilizando como medida la distancia euclídea (ecuación 2.2). Para la realización de esta distancia, se tiene que asegurar que las variables del estado tienen el mismo rango de valores. En caso negativo, se tiene que realizar un proceso de normalización de las variables, estableciendo valores entre 0 y 1 (ecuación 2.3).

$$d(s_q, s_i) = \sum_{j=0}^{n} \sqrt{(s_{q,j} - s_{i,j})} \tag{2.2}$$

$$s_{i,normalizado} = \frac{s_i - min(s_i)}{max(s_i) - min(s_i)} \tag{2.3}$$

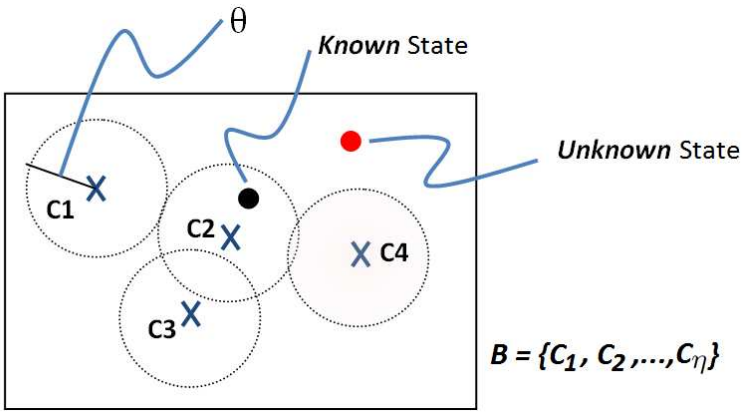


Figura 2.20: Estados conocidos y desconocidos[6]

La base de casos dispone de un parámetro θ que determina cuando un nuevo caso se añade en ella, es decir, el tamaño de la región de clasificación para cada caso en la figura 2.20 (círculo discontinuo). Cuando la distancia entre el estado del nuevo caso y el estado más cercano de la base de casos es mayor que el parámetro θ (punto rojo), el nuevo caso es declarado como estado desconocido. En caso contrario, se considera que es un estado conocido (punto negro). Para ello, se dispone de dos funciones de riesgo que determinan si un estado es conocido o desconocido, siendo una de ellas continua y otra discreta.

El factor importante de estas funciones es el valor aplicado al parámetro θ, ya que un valor alto de este parámetro puede provocar que los estados sean definidos como conocidos cuando realmente no lo son y con valor bajo, puede producir que todos los casos sean declarados como desconocidos. Por lo que para determinar este valor se utiliza la ecuación 2.4, que computa la media de las distancias (ecuación 2.2) entre los estados ejecutados en un episodio por el comportamiento base[6].

$$\theta = \frac{dist(s_1, s_2) + ... + dist(s_{n-1}, s_n)}{n - 1} \tag{2.4}$$

Comportamiento base

Cuando un agente inicia un proceso de aprendizaje basado en una exploración aleatoria, no dispone de los mecanismos suficientes para prevenir situaciones indeseables que puedan poner en riesgo al agente. Para evitar esta circunstancia, se necesita un mecanismo capaz de guiar el proceso de exploración para que el agente se mantenga lejos de situaciones dañinas.

En el aprendizaje por refuerzo seguro, este mecanismo necesario se corresponde con la inclusión de un *"teacher"*. Este *"teacher"* es el utilizado para obtener las acciones en el caso de estados desconocidos durante el proceso de exploración. Además, se utiliza como mecanismo para aprender una política, en ese caso, el *"teacher"* es tomado como comportamiento base.

Para definir correctamente un comportamiento base, tiene que cumplir tres reglas:

- Debe ser capaz de producir demostraciones seguras de la tarea a aprender para que el conocimiento pueda ser extraído.
- Debe ser capaz de soportar un proceso de exploración secuencial, aportando acciones subóptimas en estados desconocidos para reducir la probabilidad de acceder en un estado de error y retornar al sistema en el caso de estados conocidos.
- Debe estar lejos del comportamiento óptimo, ya que si el aprendizaje es óptimo no tiene sentido aplicar un proceso de aprendizaje.

Parámetro de riesgo

Para obtener una exploración segura, el movimiento a través del espacio de estados no debe ser arbitraria, sino que el espacio conocido se debe aumentar gradualmente desde estados conocidos con anterioridad.

Esta exploración se produce mediante perturbaciones en la trayectoria generada por la política base. Para generar esta perturbación, se incluye ruido aleatorio gaussiano a las acciones de la base de casos. Cuando se dispone de un estado nuevo definido como conocido, se aplica la exploración gaussiana a la acción del estado más cercano. Esta acción que se ejecuta es el resultado de aplicar una distribución gaussiana con media la acción seleccionada de la base de casos, computado mediante la ecuación 2.5.

$$\pi(s, a_i^{'}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(a_i^{'}-a_i)^2/2\sigma^2} \tag{2.5}$$

En la ecuación 2.5, se incluye un parámetro σ que determina la forma de la distribución, correspondiente con la desviación típica. Los valores de este

parámetro influyen en la probabilidad de seleccionar acciones muy diferentes a la acción actual. En el caso de que se establezca un valor alto para σ, las acciones resultantes probablemente sean muy diferentes a la original y puede llevar a estados de error. En el caso contrario, se dispone de muchas posibilidades de que la acción resultante sea muy similar a la original. Teniendo en cuenta estos aspecto, se aconseja que se comience con un valor bajo de σ y se vaya aumentando, hasta obtener el más adecuado para el dominio.

2.5. Algoritmo *PI-SRL* (Policy Improvement throught Safe Reinforcement Learning)

En esta sección se describe uno de los algoritmos utilizados para realizar el aprendizaje mediante la técnica de refuerzo seguro, *PI-SRL*. En primer lugar, se realiza una descripción general, para continuar con la descripción detallada de cada uno de los pasos.

2.5.1. *PI-SRL*: Descripción general

PI-SRL[6] es un algoritmo de aprendizaje por refuerzo seguro que realiza una exploración segura para entornos continuos y peligrosos manteniendo la integridad del agente involucrado en el proceso de aprendizaje.

La exploración segura es posible gracias a la definición y utilización de un comportamiento base, que se considera seguro y subóptimo, esto último permite que se mejore progresivamente mediante la experiencia del agente. El algoritmo *PI-SRL* está compuesto por 2 pasos:

- Modelar el comportamiento inicial a través de *CBR* (*Case-Based Reasoning*), donde se obtiene una base de casos inicial con la utilización del comportamiento base. La descripción de este paso se realiza en la sección 2.5.2.

- Mejorar el comportamiento base adquirido en el primer paso. Para ello se realiza una exploración segura del espacio estado-acción. La descripción de este paso se realiza en la sección 2.5.3.

Estos dos pasos forman el algoritmo completo, pero se puede utilizar únicamente el segundo paso en el proceso de aprendizaje. En ese caso, se dispone de una base de casos inicialmente vacía.

2.5.2. *PI-SRL*: Modelar el comportamiento inicial

El primer paso del algoritmo pretende imitar el comportamiento base utilizando técnicas de razonamiento basado en casos (*CBR*) que permite a un agente comportarse como un *"teacher"* o comportamiento base.

El uso de las técnicas *CBR* nos permite generar casos con el estado recibido a través del entorno y con la acción realizada por el comportamiento base. Esta representación es el principal objetivo de este paso, es decir, obtener una base de casos que consiga imitar el comportamiento base definido en el dominio.

La política base que se obtiene en este paso está formada por una base de casos compuesta por casos (s_j,a_j), donde s_j es el estado y a_j es la acción realizada por el comportamiento base para ese estado. Entonces, cuando se dispone de un nuevo estado, el caso que se recupera es el más semejante a dicho estado, es decir, el caso cuyo estado dispone de la distancia euclídea mínima (ecuación 2.2) respecto al nuevo estado. Dicho caso proporciona la acción a realizar para el estado recibido.

El algoritmo de este primer paso es el mostrado en la tabla 2.2.

En este paso del algoritmo, se inicia con la base de casos vacía y se va completando a medida que se obtiene experiencia con el entorno. Para completar la base de casos, en cada iteración se crea un nuevo caso formado por el estado recibido del entorno, la acción a realizar en dicho estado y la función de valor que en este paso es 0, siendo actualizado en el segundo paso del algoritmo en la sección 2.5.3.

$$\varrho^{\pi_B^\theta} = \begin{cases} 0 & if\ \text{mín}_{1\leq j\leq \eta}\, d(s_q, s_i) < \theta \\ 1 & otherwise \end{cases} \tag{2.6}$$

Para determinar si un estado es de riesgo o no, se determina mediante una función de riesgo discreta (ecuación 2.6). Si el estado es declarado como de no riesgo (estado conocido), se recupera la acción del vecino más cercano, asignándola al nuevo estado (línea 10). En caso contrario (estado desconocido), se ejecuta el comportamiento base (línea 12) y se crea un nuevo caso con el estado y la acción realizada por el comportamiento base, siendo añadido en la base de casos (línea 14). Como se empieza con la base de casos vacía, la base de casos se incrementa durante el proceso de aprendizaje, y este incremento puede producir un problema de tiempo de computación en la busqueda del estado más cercano. Para reducir este problema, se utiliza una estructura de datos que organiza los puntos en un espacio euclídeo de k dimensiones, denominados árboles KD[6].

Cuando se completa el número de episodios definidos, se comprueba si el tamaño de la base de casos excede el límite establecido (línea 18). En caso

Aproximación *CBR* para Imitación del Comportamiento[6]	
00	Dado el comportamiento base, π_T
01	Dado el umbral de densidad, θ
02	Dado el número máximo de casos, η
03	Dado un número máximo de episodios de aprendizaje a ejecutar, K
04	1. Inicializar la base de casos $B = \emptyset$
05	2. **Repetir**
06	Establecer $k = 0$
07	**Mientras** $k < K$ **hacer**
08	Calcular el caso $< s_c, a_c, 0 >$ más cercano al estado actual s_k
09	**Si** $\varrho^{\pi_B^\theta}(s_k) = 0$ **entonces** // *Mediante la ecuación 2.6*
10	Establecer $a_k = a_c$
11	**en otro caso**
12	Seleccionar a_k mediante el **comportamiento base** π_T
13	Crear un nuevo caso $c^{new} = (s_k, a_k, 0)$
14	$B := B \cup c^{new}$
15	Ejecutar a_k, y recibir el nuevo estado s_{k+1}
16	Establecer $k = k + 1$
17	**fin mientras**
18	**Si** $\|B\| > \eta$ **entonces**
19	Eliminar los $\eta - \|B\|$ casos menos frecuentemente usados de B
20	**hasta** que el criterio de parada se haga verdadero
21	3. **Devolver** ***B*** de la cual se deriva la **Política Segura Basada en Casos**, π_B^θ

Tabla 2.2: Descripción del paso 1 del algoritmo *PI-SRL*.

afirmativo, se procede a la exclusión de los casos menos recuperados (línea 19).

Al completar este paso del algoritmo, se dispone de una base de casos que garantiza una política segura y subóptima que simula el comportamiento base.

2.5.3. *PI-SRL*: Mejorar la política base aprendida

En este paso se utiliza la política segura y subóptima basada en casos obtenida en el paso anterior del algoritmo, sobre la cual se aplica una exploración segura del espacio estado-acción.

Cada caso de la base de casos disponen de una función de valor, la cual se calcula mediante el algoritmo de Monte Carlo (MC) adaptado para *CBR*, que se muestra en la tabla 2.3.

En este algoritmo mostrado en la tabla 2.3, los *returns* para cada estado

	Algoritmo *MC* adaptado a *CBR*
00	Dada la base de casos, B
01	Dado un número máximo de episodios de aprendizaje a ejecutar, K
02	1. Inicializar, **Para cada** $c^i \in B$
03	$V(s) \leftarrow$ *arbitrariamente*
04	$Returns(s) \leftarrow$ *lista vacía*
05	2. **Mientras** $k < K$
06	Generar un episodio utilizando π_B^θ
07	**Para cada** s que aparece en el episodio con $< s, a, V(s) > \in B$
08	$R \leftarrow$ retorna siguiendo la primera ocurrencia de s
09	Añadir R to *Returns(s)*
10	*V(s)* $\leftarrow$ *promedio(Returns(s))*
11	Establecer $k = k + 1$
12	3. **Devolver** B

Tabla 2.3: Descripción del algoritmo Monte Carlo para el cálculo de la función estado-valor para cada caso.

son acumulados y promediados, siguiendo la política de comportamiento obtenida en la base de casos. Además, el término 'return" se refiere al retorno esperado utilizando los siguientes a la primera ocurrencia del estado, es decir, es la suma de refuerzos aplicando un descuento a futuro desde esa ocurrencia. Por otro lado el término "Returns" se refiere a una lista compuesta por cada "return" del estado en diferentes episodios. La principal ventaja del método *MC* es la rapidez y sencillez en el cálculo de la estimación del valor de cada estado.

El segundo paso del algoritmo se muestra en la tabla 2.5. Este paso del algoritmo está formado por 4 fases para cada episodio.

1. **Inicialización del paso**: se inicializa la lista de casos realizados en ese periodo y se inicia las variables del periodo, como es el refuerzo acumulado.

2. **Generación de casos**: se corresponde con la tabla 2.4, y en ella se construye un caso para cada iteración del episodio. Para cada nuevo estado, se recupera el caso más cercano a dicho estado (línea 10) utilizando la distancia euclídea (ecuación 2.2) y se computa la función de riesgo (ecuación 2.6) para comprobar si el estado es conocido o desconocido.

   Para estados conocidos (línea 11), se recupera la acción del caso más cercano y se aplica un ruido aleatorio según una distribución gaussiana

a la acción (línea 12), formando así el nuevo caso con el estado, la acción modificada y el valor del caso más cercano (línea 14). En caso contrario (línea 15), se ejecuta el comportamiento base para obtener la acción (línea 16) y se crea un nuevo caso con ella, estableciendo el valor a 0 (línea 18).

Para concluir esta fase, es actualizado el refuerzo total del periodo con el obtenido en cada iteración (línea 20) y se añade el caso obtenido en la lista de casos del periodo (línea 21).

3. **Cálculo de la función de estado-valor para estados desconocidos**: esta fase se realiza para los casos creados en cada episodio (estados desconocidos). Para dichos casos, se calcula el valor a partir de la primera ocurrencia del estado en el episodio (línea 14) de manera similar que en el algoritmo *MC* (tabla 2.3).

4. **Actualización de los casos en la base de casos con la experiencia obtenida**: en esta fase se realiza la actualización de la base de casos, para ello se comprueba el refuerzo acumulado en el episodio y si éste es semejante (mayor o dentro del límite establecido por el parámetro Θ) al mejor obtenido en episodios anteriores (línea 17), se realiza la actualización de la base de casos.

   En esta actualización, se diferencia si el estado es conocido o desconocido. En el primero de los casos (estado conocido), se comprueba si es adecuado el reemplazo de la acción. Para esta actualización, se calcula el error de la distancia temporal (línea 22). Si este error es positivo, se actualiza la acción ya que se considera que la acción produce mejores políticas (línea 24) y se actualiza el valor utilizando la distancia temporal obtenida (línea 25). En el caso contrario (estado desconocido), se inserta en la base de casos (línea 27).

Por último, si la base de casos supera el límite establecido se procede a la eliminación de los casos que menos se han usado en el proceso de aprendizaje (línea 29).

Exploración segura del *PI-SRL* (π_T, K, B, σ)	
00	$listCasesEpisode \leftarrow \emptyset$.
01	$totalRwEpisode = 0$.
02	Establecer $k = 1$.
03	Recuperar el estado inicial, s_k.
09	**Mientras** $k < K$ **hacer**
10	Buscar el caso $< s, a, V(s) > \in B$ más cercano al estado actual s_k
11	**Si** $\varrho^{\pi^{\theta}_B}(s_k) = 0$ **entonces** // *estado conocido*
12	Seleccionar una acción a_k mediante la ecuación 2.5
13	Ejecutar la acción a_k
14	Crear un nuevo caso $c^{new} := (s, a_k, V(s))$
15	**en otro caso** // *estado desconocido*
16	Seleccionar la acción a_k mediante π_T
17	Ejecutar la acción a_k
18	Crear un nuevo caso $c^{new} := (s_k, a_k, 0)$
19	Recibir el siguiente estado s'_k y el refuerzo $r(s_k, a_k)$
20	$totalRwEpisode := totalRwEpisode + r(s_k, a_k)$
21	$listCasesEpisode := listCasesEpisode \cup c^{new}$
22	Establecer $k = k + 1$
23	Establecer $s_k \leftarrow s'_k$
24	Retornar $listCasesEpisode, totalRwEpisode$

Tabla 2.4: Descripción de la estrategia de exploración segura *PI-SRL*

Algoritmo de Mejora de la Política[6]

00 Dada la base de casos, B, y el máximo número de casos, η
01 Dado el comportamiento base, π_T
02 Dado el umbral de actualización de la base de casos, Θ
03 Dado un número máximo de episodios de aprendizaje a ejecutar, K
04 1. Establecer $maxTotalRwEpisode = 0$, el máximo de refuerzo acumulado obtenido en un episodio
05 2. **Repetir**
06 (a) ***Inicialización***:
07 **Establecer** $k = 0$, $listCasesEpisode \leftarrow \emptyset$, $totalRwEpisode = 0$
08 (b) ***Generación de Casos***:
09 $listCasesEpisode, totalRwEpisode := exploracion\_segura(\pi_T, H, B, \sigma)$ // Tabla 2.4
10 **Para cada** caso c_i en $listCasesEpisode$
11 (c) ***Calcular la función de valor-estado para los estados desconocidos***:
12 **Para cada** caso c_i en $listCasesEpisode$
13 **Si** $\varrho^{\pi^{\theta}_B}(s_i) = 1$ **entonces** // *estado desconocido*
14 $return(s_i) := \sum_{j=n}^{k} \gamma^{j-n} r(s_j, a_j)$ // *n se refiere a la primera ocurrencia de* s_i *en el episodio*
15 $V(s_i) := return(s_i)$
16 (d) ***Actualizar los casos en B utilizando la experiencia recuperada***:
17 **Si** $totalRwEpisode > (maxTotalRwEpisode - \Theta)$ **entonces**
18 $maxTotalRwEpisode := \boldsymbol{max}(maxTotalRwEpisode, totalRwEpisode)$
19 **Para cada** caso $c_i =< s_i, a_i, V(s_i) >$ en $listCasesEpisode$
20 **Si** $\varrho^{\pi^{\theta}_B}(s_i) = 0$ **entonces** // *estado conocido*
21 Calcular el caso $< s_i, a, V(s_i) > \in B$ correspondiente al estado s_i
22 Calcular $\delta = r(s_i, a_i) + \gamma V(s_{i+1}) - V(s_i)$
23 **Si** $\delta > 0$ **entonces**
24 Reemplazar el caso $< s_i, a, V(s_i) > \in B$ con el caso $< s_i, a_i, V(s_i) > \in listCasesEpisode$
25 $V(s_i) = V(s_i) + \alpha\delta$
26 **en caso contrario** // *estado desconocido*
27 $B := B \cup c_i$
28 **Si** $\|B\| > \eta$ **entonces**
29 Eliminar los $\eta - \|B\|$ menos frecuentemente usados en B
30 **hasta** que el criterio de parada se haga verdadero
31 3. **Devolver** B

Tabla 2.5: Descripción del segundo paso del algoritmo PI-SRL

2.6. Algoritmo *PR-SRL* (*Policy Reuse for Safe Reinforcement Learning*)

En esta sección se describe el otro algoritmo que aplica una exploración segura en el proceso de aprendizaje. Este algoritmo es el *PR-SRL*[7].

π-**Reuse Segura** (π_T, H, B, σ)	
00	$listCasesEpisode \leftarrow \emptyset$.
01	$totalRwEpisode = 0$.
02	Establecer $h = 1$.
03	Recuperar el estado inicial, s_h.
04	Mientras $h < H$
05	Buscar el caso $< s, a, V(s) > \in B$ más cercano al estado actual s_h
06	Calcular $\varrho^B(s_h) = 1 - \frac{1}{1+e^{\frac{k}{\theta}((\min_{1 \le j \le \eta} d(s_h, s_j) - \frac{\theta}{k}) - \theta)}}$
07	Con una probabilidad de $\varrho^B(s_h)$: $a_h = \pi_T(s_h)$, $c^{new} := (s_h, a_h, 0)$
08	Con una probabilidad de $1 - \varrho^B(s_h)$:
09	$a_h = rnd\_gaussian(\pi_B(s_h), \sigma)$ // ecuación 2.5
10	$c^{new} := (s, a_h, V(s))$
11	Ejecutar a_h y recibir el siguiente estado s'_h, y refuerzo, $r(s_h, a_h)$
12	$totalRwEpisode := totalRwEpisode + r(s_h, a_h)$
13	$listCasesEpisode := listCasesEpisode \cup c^{new}$
14	Establecer $s_h \leftarrow s'_h$
15	Retornar $listCasesEpisode, totalRwEpisode$

Tabla 2.6: Descripción de la estrategia de exploración segura π-Reuse

En la tabla 2.6 se muestra la técnica de exploración del algoritmo *PR-SRL*. En ella se utiliza una función de riesgo continua (ecuación 2.7) que permite obtener una transición suavizada entre los estados conocidos y los desconocidos. En ella se introduce un parámetro k que produce un doble efecto:

- La anchura de la función sigmoide varía (figura 2.21b). Si se establece un valor bajo, implica una función más amplia, produciendo un proceso de exploración menos agresivo del espacio de estados debido a un uso frecuente del comportamiento base.
- El desplazamiento de la función sigmoide, a la izquierda (figura 2.21b), reduciendo la probabilidad de tener en cuenta los estados desconocidos como estados conocidos.

$$\varrho^B(s) = 1 - \frac{1}{1 + e^{\frac{k}{\theta}((\text{mín}_{1 \leq j \leq \eta}\, d(s,s_j) - \frac{\theta}{k}) - \theta)}} \tag{2.7}$$

En resumen, los valores bajos de k implica un proceso de exploración del espacio de estados poco agresivo, reduciendo la probabilidad de daño en el agente pero no disponiendo de un rendimiento adecuado del algoritmo. En el caso contrario, implica que la función sigmoide (figura 2.21b) se asemeje a la función de riesgo discreta (figura 2.21a).

Esta función de riesgo es la principal diferencia entre las técnicas de exploración utilizadas por los algoritmos *PI-SRL* (tabla 2.4) y *PR-SRL* (tabla 2.6).

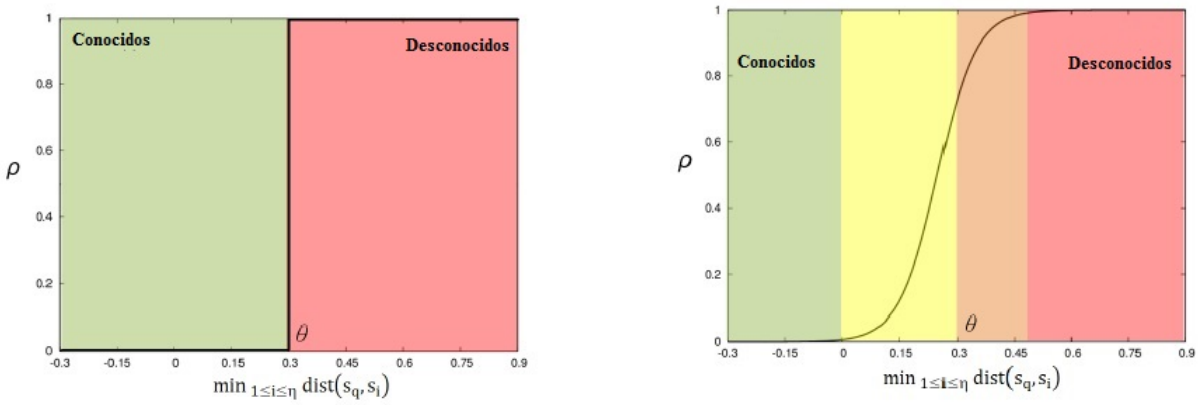


Figura 2.21: (a) Función de riesgo discreta. (b) Función de riesgo continua[7]

Por otro lado, en la tabla 2.7 se muestra la estructura del algoritmo *PR-SRL* que dispone de los mismos pasos que en el caso del segundo paso del *PI-SRL*. La única diferencia es la técnica de exploración utilizada en el paso de generación de casos (paso b), en este caso es la mostrada en la tabla 2.6 y en el caso del *PI-SRL*, la mostrada en la tabla 2.4.

Algoritmo de Mejora de la Política[7]

00 Dada la base de casos, B, y el máximo número de casos, η
01 Dado el comportamiento base, π_T
02 Dado el umbral de actualización de la base de casos, Θ y un parámetro de riesgo σ
03 Dado un número máximo de episodios de aprendizaje a ejecutar, K
04 1. Establecer $maxTotalRwEpisode = 0$, el máximo de refuerzo acumulado obtenido en un episodio
05 2. **Repetir**
06 (a) ***Inicialización***:
07 **Establecer** $k = 0$, $listCasesEpisode \leftarrow \emptyset$, $totalRwEpisode = 0$
08 (b) ***Generación de Casos***:
09 $listCasesEpisode, totalRwEpisode := \pi - reuse(\pi_T, H, B, \sigma)$
10 (c) ***Calcular la función de valor-estado para los estados desconocidos***:
11 **Para cada** caso c_i en $listCasesEpisode$
12 **Si** s_i es considerado *estado desconocido* **entonces**
13 $return(s_i) := \sum_{j=n}^{k} \gamma^{j-n} r(s_j, a_j)$
14 $V(s_i) := return(s_i)$
15 (d) ***Actualizar los casos en B utilizando la experiencia recuperada***:
16 **Si** $totalRwEpisode > (maxTotalRwEpisode - \Theta)$ **entonces**
17 $maxTotalRwEpisode := \boldsymbol{max}(maxTotalRwEpisode, totalRwEpisode)$
18 **Para cada** caso $c_i =< s_i, a_i, V(s_i) >$ en $listCasesEpisode$
19 **Si** s_i es considerado *estado conocido* **entonces**
20 Calcular el caso $< s_i, a, V(s_i) > \in B$ correspondiente al estado s_i
21 Calcular $\delta = r(s_i, a_i) + \gamma V(s_{i+1}) - V(s_i)$
22 **Si** $\delta > 0$ **entonces**
23 Reemplazar el caso $< s_i, a, V(s_i) > \in B$ con el caso $< s_i, a_i, V(s_i) > \in listCasesEpisode$
24 $V(s_i) = V(s_i) + \alpha\delta$
25 **en caso contrario** // *estado desconocido*
26 $B := B \cup c_i$
27 **Si** $\|B\| > \eta$ **entonces**
28 Eliminar los $\eta - \|B\|$ menos frecuentemente usados en B
29 **hasta** que el criterio de parada se haga verdadero
30 3. **Devolver** B

Tabla 2.7: Descripción del algoritmo *PR-SRL*

2.7. Robot Operating System (*ROS*)

ROS (**R**obot **O**perating **S**ystem) es un sistema operativo de código libre utilizado en diferentes modelos de robots. Este sistema proporciona servicios que facilitan el desarrollo de software para robots. Estos sistemas consisten en la abstracción del hardware, el control de dispositivos a bajo nivel, la implementación de habilidades que se usan habitualmente, un sistema de pasos de mensajes entre procesos y una organización estructurada por paquetes. A estos servicios, se añade el conjunto de librerías y comandos que dispone para la obtención, compilación, escritura y ejecución en múltiples ordenadores[39].

A continuación se muestran los conceptos relacionados al sistema de ficheros y de computación definidos por *ROS*.

2.7.1. Sistema de ficheros

Los ficheros en *ROS* disponen de una estructura fija compuesta por los conceptos:

- **Paquete**: es la principal unidad para organizar el software. Puede contener los procesos ejecutables de *ROS* (nodos), librería, estructuras de datos o como mecanismo de organización.

- **Manifests**: contiene los metadatos sobre el paquete, incluyendo información de licencia y la dependencia con otros paquetes.

- **Stacks**: es una colección de paquetes.

- **Stack Manifests**: contiene los metadatos sobre la *stack*, incluyendo información sobre su organización y la dependencia con otras *stack*.

- **Tipo de mensajes**: la descripción de los mensajes, que es una estructura de datos utilizada para la comunicación en *ROS*. Estos mensajes se localizan en la carpeta "msg" de la estructura de paquetes.

- **Tipo de servicios**: la descripción de los servicios, que es una estructura utilizada para la comunicación en *ROS*. Estos servicios se localizan en la carpeta "srv" de la estructura de paquetes.

Por último, se muestra un ejemplo de la organización de un software desarrollado en *ROS* donde se muestra los conceptos descritos anteriormente (figura 2.22).

Figura 2.22: Sistema de archivos en *ROS*

2.7.2. Sistema de computación

El sistema de computación gráfico es una red *peer-to-peer* de procesos débilmente acoplados a la infraestructura de comunicación de *ROS*. *ROS* utiliza diferentes estilos de comunicación, como son la síncrona mediante *RPC* (servicios), el intercambio de datos asíncrono (*topics*) o almacenando datos (*Parameter Server*). Los principales conceptos utilizados en estos sistemas de comunicación son:

- **Nodos**: son procesos que realizan la computación. Un sistema de control de un robot comprende habitualmente a múltiples nodos. Los nodos de *ROS* son escritos usando librerías de clientes *ROS*, como son *roscpp*, para su desarrollo en *C++*, o *rospy*, para su desarrollo en *python*.
- **Master**: contiene el registro de nombres y realiza la búsqueda del resto de elementos del sistema de computación. Sin el *Master*, el resto de nodos no pueden intercambiar mensajes ni invocar servicios, ya que no se encontrarían.
- **Parameter Server**: permiten establecer datos al proceso central (*Master*).
- **Mensajes**: es la estructura de datos utilizada para el intercambio de mensajes entre nodos.

- ***Topics***: es el nombre que identifica el contenido del mensaje en el sistema de publicador/subscriptor. Este nombre es único y puede ser utilizado por varios publicadores y subscriptores.

- **Servicios**: es la representación del modelo petición/respuesta para la comunicación entre nodos. Los nodos ofertan servicios utilizando un nombre y los clientes ejecutan el servicio mandando la petición y esperan a la respuesta del servicio.

Por último, se ilustra en un ejemplo los conceptos comentados anteriormente (figura 2.23).

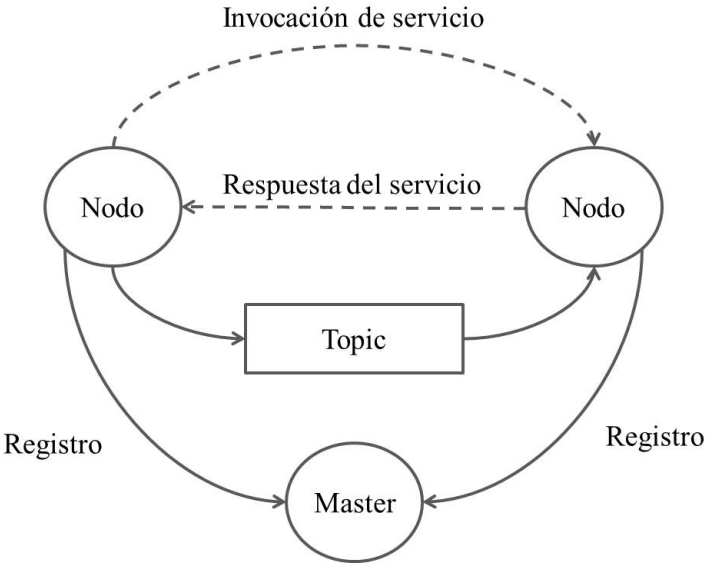


Figura 2.23: Sistema de computación en *ROS*

Esta plataforma se utiliza en el proyecto para la compilación y ejecución del software. Además, se utiliza el sistema de comunicaciones que proporciona tanto el intercambio de mensajes como la ejecución de servicios, aprovechando el resto de facilidades que aporta esta plataforma.

Capítulo 3

Arquitectura para aprender el comportamiento de caminar de un robot *NAO*

En este capítulo se describe la arquitectura desarrollada para aplicar el algoritmo *PI-SRL* y *PR-SRL* al comportamiento de caminar del robot *NAO*.

En primer lugar se muestra la estructura general de la arquitectura, que muestra los módulos desarrollados para realizar el proceso de aprendizaje. A continuación, se describe el comportamiento de caminar en el robot *NAO* para problemas de *RL* (*R*einforcement *L*earning), detallando los diferentes niveles de abstracción probados para este comportamiento. Por último, se mostrará el diseño de la arquitectura, mediante el diagrama de clases de los diferentes módulos y la estructura de nodos utilizada en *ROS*.

3.1. Estructura general

En esta sección se muestra la estructura general de la arquitectura desarrollada para representar el comportamiento de caminar del robot *NAO* mediante algoritmo de *RL* seguro.

En la figura 3.1, se observan dos módulos, "Safe Reinforcement Learning" y "Humanoid Step".

El módulo "Safe Reinforcement Learning" contiene la implementación del dominio y de los algoritmos. Este módulo es el encargado de seleccionar las acciones que realiza el robot mediante la aplicación de algoritmos de aprendizaje.

Por otro lado, el módulo "Humanoid Step" es el encargado de conectar con el robot *NAO* para realizar las acciones y obtener la información necesaria

en el proceso de aprendizaje.

Por último, estos módulos se incluyen dentro de *ROS* para ser compilados y ejecutados en dicha plataforma. Además, *ROS* proporciona los mecanismos de comunicación (sección 2.7), como son la definición y utilización de mensajes y servicios, para la comunicación entre los dos módulos. Por otro lado, se utilizan los comandos de alto nivel para la comunicación con el robot *NAO*, cargando la librería del robot *NAO* (*NAOqi*) en el código *ROS*.

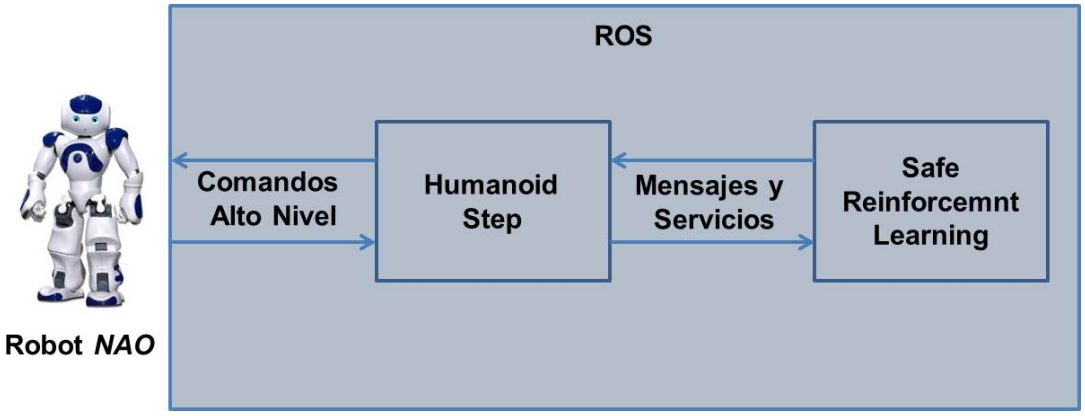


Figura 3.1: Estructura general de la arquitectura

3.2. Descripción del comportamiento de caminar como un problema de Aprendizaje por Refuerzo

En este apartado se define la representación del comportamiento de caminar como un problema de *RL*. Además, se incluye la adaptación de los algoritmos *PI-SRL* y *PR-SRL* para la representación utilizada.

El comportamiento de caminar se puede tratar desde dos niveles de abstracción distintos:

1. Abstracción de bajo nivel: el comportamiento de caminar se descompone en una sucesión de ángulos de las articulaciones del robot, compuestos por 26 variables, mostradas en la figura 2.10.

2. Abstracción de alto nivel: el comportamiento de caminar se descompone en una secuencia de pasos (figura 2.13).

El nivel de abstracción en este trabajo es el alto, ya que para el dominio de bajo nivel no se dispone de los mecanismos suficientes para obtención de experiencia válida y de mecanismos de evaluación, como puede ser la distancia recorrida por cada paso o acción.

3.2.1. Abstracción de alto nivel

Esta abstracción de alto nivel representa el comportamiento de caminar como una sucesión de pasos hasta que el robot alcance la distancia fijada, lo que determina la finalización de los episodios.

Un episodio de este dominio está compuesto por dos fases, ya que al evaluar cada acción del proceso de aprendizaje se introduce un sesgo en la medida de evaluación del episodio, lo que impide evaluar el episodio en una sola fase.

1. **Fase de aprendizaje**: se realiza el proceso de aprendizaje del algoritmo. En esta fase, el número de pasos del episodio está determinado por los pasos realizados por el robot para completar la distancia fijada, o hasta su caída en caso de error.

2. **Fase de evaluación**: al finalizar la fase de entrenamiento, se ejecutan secuencialmente todos los pasos obtenidos y de este modo, se obtiene la velocidad del episodio.

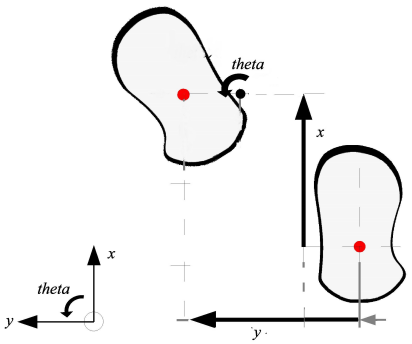


Figura 3.2: Representación del estado

Para representar este dominio, se dispone de un espacio de estados y acciones continuos. El espacio de estados está compuesto por una tupla de 3 componentes, $< x, y, \theta >$, siendo x e y, la separación longitudinal y transversal en metros con respecto al pie contrario, y θ, la rotación a través del eje vertical (Z) en radianes (figura 3.2). El espacio de acciones se compone

de una tupla de 4 componentes $< \Delta x, \Delta y, \Delta\theta, t >$, que representa con el desplazamiento a realizar por un pie, respecto al contrario en cada una de las variables de estado, y el tiempo de realización del desplazamiento, t.

Por otro lado, el refuerzo aplicado a las acciones es el desplazamiento del robot en un paso ($x_{step} - x_i$, en la figura 3.3), es decir, el algoritmo intentaría realizar pasos más largos. Adicionalmente, el refuerzo total del episodio se computa mediante la ecuación 3.1, que se corresponde con la velocidad obtenida en la fase de evaluación del episodio.

$$totalReward = \frac{x_f - x_i}{t_f - t_i} \tag{3.1}$$

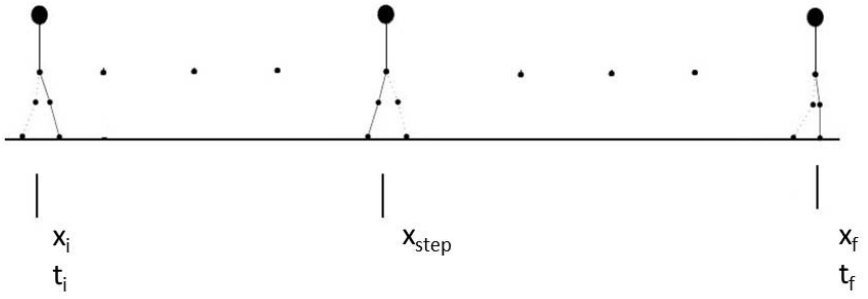


Figura 3.3: Representación de la distancia recorrida

En la tabla 3.1 se muestra el comportamiento base definido para los algoritmos *PI-SRL* y *PR-SRL*. El comportamiento base se configura mediante dos parámetros, la distancia que recorre en este comportamiento x y la configuración de los pasos $config$, proporcionando como salida una acción o paso, cada vez que es requerido. El proceso se compone de 2 fases:

1. **Obtener pasos**: en esta fase, se obtienen los pasos que el comportamiento base retorna, en caso que sea necesario. Si el estado anterior al definido como desconocido ha sido definido como conocido (línea 04) por el algoritmo, se utiliza el comando **walkTo** con los parámetros definidos de distancia x (línea 00) y de configuración de pasos $config$ (línea 01) para obtener una lista de pasos, que es almacenada en este comportamiento. En caso contrario, se actualiza el índice de posición (línea 07) para utilizar el siguiente paso, ya que la secuencia de pasos almacenada es válida.

2. **Seleccionar de pasos**: en esta fase se selecciona el paso que retorna el comportamiento base. El paso seleccionado se corresponde con el

paso apuntado por el índice de pasos, *posicion_paso*, actualizando este contador (línea 09).

Comportamiento base para el dominio de caminar	
00	Definir una distancia x
01	Definir la configuración de los pasos $config$ //Máxima o Defecto
02	1. Establecer la lista de pasos, $pasos = \emptyset$, y la posición de paso, $posicion\_paso = 0$
03	2. **Obtener pasos**
04	**Si** $estado\_anterior\_conocido = true$ **entonces**
05	$posicion\_paso = 0$
06	Establecer *steps* usando **walkTo**$(x, 0, 0, config)$
07	**en otro caso** $posicion\_paso = posicion\_paso + 1$
08	3. **Seleccionar de acción**
09	Establecer $step = steps[posicion\_paso]$
10	5. **Retornar** step

Tabla 3.1: Descripción del comportamiento base para el problema de caminar

3.2.2. Adaptación de los algoritmos *PI-SRL* y *PR-SRL*

En esta sección se muestra las fases de exploración de los algoritmos *PI-SRL* y *PR-SRL*, adaptados a la abstracción de alto nivel.

En las tablas 3.2 y 3.3 se muestra la adaptación de la exploración utilizada en la segunda fase del algoritmo *PI-SRL* (tabla 2.4) y de la pólitica π-reuse segura (tabla 2.6) del algoritmo *PR-SRL*, respectivamente. Esta adaptación se refleja en seis aspectos:

1. El criterio de parada del episodio (línea 05). Se establece la distancia que tiene que recorrer el robot en un episodio como criterio de finalización, produciendo episodios con diferentes números de pasos. Este número de pasos por episodio tenderá a reducirse en la obtención de un comportamiento más rápido.

2. Inclusión de un mecanismo de normalización para el estado (línea 06), debido a los diferentes rangos de valores de los atributos de los estados $< x, y, \theta >$.

3. La inclusión del comportamiento base (líneas 09-15 en la tabla 3.2 y líneas 10-16 en la tabla 3.3). Este comportamiento base se describe en la tabla 3.1.

4. Modificación de la componente Δy de la acción (línea 21 y 22, respectivamente), que se realiza en el caso de que el pie que debe moverse sea el izquierdo, ya que la posición respecto al pie contrario es la inversa.

5. Actualización del pie que realiza el movimiento (línea 25 y 26, respectivamente), se alterna el pie para la siguiente acción sea realizado con el pie contrario.

6. Asignación del refuerzo total del episodio (línea 29 y 30), se asigna el refuerzo total del episodio según el valor asignado a la fase de evaluación.

Exploración segura del *PI-SRL* adaptada al comportamiento de caminar

00 Dada la distancia del episodio, *distanciaEpisodio*
01 *listaCasosEpisodio* $\leftarrow \emptyset$, *pasos* $\leftarrow \emptyset$.
02 *totalRwEpisodio* = *0*, *estado_anterior_conocido* = *true*.
03 Establecer *distancia* = 0, *posicion_paso* = 0, *pierna* = *Derecha*.
04 Establecer el estado inicial, s_h.
05 Mientras *distancia* < *distanciaEpisode*
06 Establecer s_n normalizando s_h // ecuación 2.3
07 Buscar el caso $< s, a, V(s) > \in B$ más cercano al estado actual s_n
08 **Si** $\varrho^{\pi_B^{\theta}}(s_k) = 1$ **entonces**
09 **Si** *estado_anterior_conocido* **entonces**
10 *posicion_paso* = 0
11 Establecer *steps* usando **walkTo**(*dist*, 0, 0, $config$)
12 **en otro caso** *posicion_paso*+ = 1
13 $a_h = pasos[posicion\_paso]$
14 $c^{new} := (s_n, a_h, 0)$
15 $estado\_anterior\_conocido = false$
16 **en otro caso** //Estado conocido
17 $a_h = rnd\_gaussian(\pi_B(s_n), \sigma)$, $c^{new} := (s, a_h, V(s))$
18 $pasos \leftarrow \emptyset$
19 $estado\_anterior\_conocido = true$
20 **Si** $pierna == Izquierda$ **entonces**
21 Establecer $a_h(\Delta y) = -a_h(\Delta y)$
22 Ejecutar a_h y recibir el siguiente estado s'_h, y refuerzo, $r(s_h, a_h)$
23 $listaCasosEpisodio := listaCasosEpisodio \cup c^{new}$
24 $distance := distance + r(s_h, a_h)$
25 **Si** $pierna == Derecha$ **entonces**
26 $pierna = Izquierda$
27 **en otro caso** $pierna = Derecha$
28 Establecer $s_h \leftarrow s'_h$
29 *totalRwEpisodio* = *velocidad_episodio*

Tabla 3.2: Exploración segura del *PI-SRL* adaptada al comportamiento de caminar

π-reuse segura adaptada al comportamiento de caminar

00 Dada la distancia del episodio, *distanciaEpisodio*
01 $listaCasosEpisodio \leftarrow \emptyset$, $pasos \leftarrow \emptyset$.
02 $totalRwEpisodio = 0$, $estado\_anterior\_conocido = true$.
03 Establecer $distancia = 0$, $posicion\_paso = 0$, $pierna = Derecha$.
04 Establecer el estado inicial, s_h.
05 Mientras $distancia < distanciaEpisode$
06 Establecer s_n normalizando s_h // ecuación 2.3
07 Buscar el caso $< s, a, V(s) > \in B$ más cercano al estado actual s_n
08 Calcular $\varrho^B(s_h) = 1 - \frac{1}{1+e^{\frac{k}{\theta}((\min_{1 \leq j \leq \eta} d(s_h, s_j) - \frac{\theta}{k}) - \theta)}}$
09 Con una probabilidad de $\varrho^B(s_h)$:
10 **Si** $estado\_anterior\_conocido$ **entonces**
11 $posicion\_paso = 0$
12 Establecer *steps* usando **walkTo**($dist$, 0, 0, $config$)
13 **en otro caso** $posicion\_paso+ = 1$
14 $a_h = pasos[posicion\_paso]$
15 $c^{new} := (s_n, a_h, 0)$
16 $estado\_anterior\_conocido = false$
17 Con una probabilidad de $1 - \varrho^B(s_h)$:
18 $a_h = rnd\_gaussian(\pi_B(s_n), \sigma)$, $c^{new} := (s, a_h, V(s))$
19 $pasos \leftarrow \emptyset$
20 $estado\_anterior\_conocido = true$
21 **Si** $pierna == Izquierda$ **entonces**
22 Establecer $a_h(\Delta y) = -a_h(\Delta y)$
23 Ejecutar a_h y recibir el siguiente estado s'_h, y refuerzo, $r(s_h, a_h)$
24 $listaCasosEpisodio := listaCasosEpisodio \cup c^{new}$
25 $distance := distance + r(s_h, a_h)$
26 **Si** $pierna == Derecha$ **entonces**
27 $pierna = Izquierda$
28 **en otro caso** $pierna = Derecha$
29 Establecer $s_h \leftarrow s'_h$
30 $totalRwEpisodio = velocidad\_episodio$

Tabla 3.3: Política π-reuse segura adaptada al comportamiento de caminar

3.3. Diseño detallado de la arquitectura

En esta sección se detalla la arquitectura, en relación a la organización de paquetes y la dependencia con paquetes proporcionados por *ROS* (sección 3.3.1). Además, se muestra la estructura interna de cada uno de los paquetes desarrollados (sección 3.3.2). Por último, se muestra la estructura de comunicación entre los nodos del proceso de aprendizaje (sección 3.3.3).

3.3.1. Diagrama de paquetes

En este apartado se muestran los paquetes que conforman la arquitectura y las dependencias con el resto de paquetes de *ROS*. Estos paquetes se corresponden con los módulos incluidos en la figura 3.1, *"Reinforcement Learning"* y *"humanoid_step"*.

La estructura del paquete *"Reinforcement Learning"* está basada en el paquete de "reinforcement_learning" disponible en *ROS*, desarrollado por Todd Hester[35].

En la figura 3.4 se observa la estructura de los paquetes *"Reinforcement Learning"* y *"humanoid_step"*, y la dependencia entre ellos.

El paquete *"Reinforcement Learning"* se compone de 5 paquetes.

1. **rl_agent**: contiene la definición de algoritmos de *RL*.
2. **rl_common**: define estructuras de datos utilizados por el resto de paquetes.
3. **rl_env**: contiene la definición de los entorno que representan los problemas a resolver.
4. **rl_experiment**: contiene el código ejecutable de los experimentos eliminando el paso de mensajes entre los agentes y entornos.
5. **rl_msgs**: contiene los mensajes utilizados en la comunicación entre los agentes y entornos.

Por otro lado, el paquete *"humanoid_step"* está compuesto por 2 paquetes.

1. **humanoid_step_launch**: contiene el script que ejecuta el nodo que conecta con el robot *NAO*.
2. **humanoid_step_msgs**: contiene la definición de los mensajes y servicios utilizados para la comunicación con el módulo *"Reinforcement Learning"*.

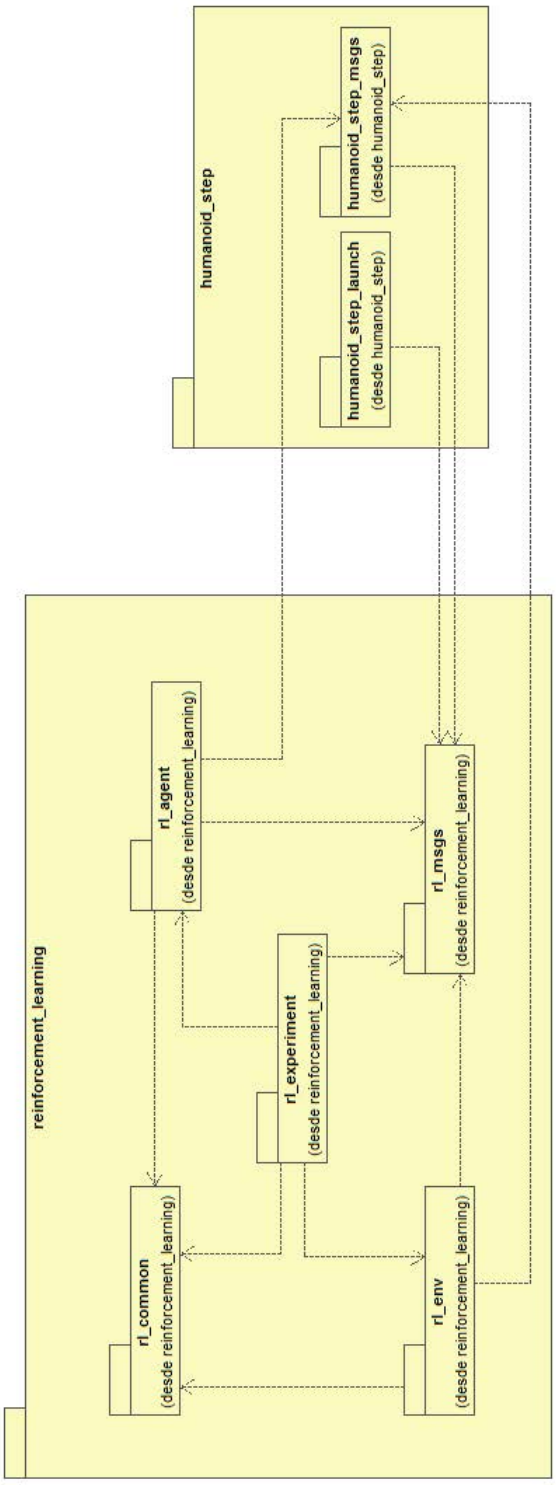


Figura 3.4: Diagrama de paquetes - Dependencias entre "Reinforcement Learning" y "humanoid_step"

En la figura 3.5 se muestra la dependencia de los paquetes desarrollado en la arquitectura con el resto de paquetes definidos en *ROS*.

Los paquetes de *ROS* requeridos por la arquitectura son:

- **nao_msgs**: contiene la definición de los mensajes y servicios para el robot *NAO*.
- **actionlib**: es la librería para la utilización de servicios.
- **geometry_msgs**: proporciona primitivas comunes para representar datos geométricos.
- **nao_driver**: proporciona el driver necesario para acceder a los comandos de alto nivel del robot *NAO*.

Los mensajes y servicios utilizados de cada uno de los paquetes se describen con detalle en el anexo B.

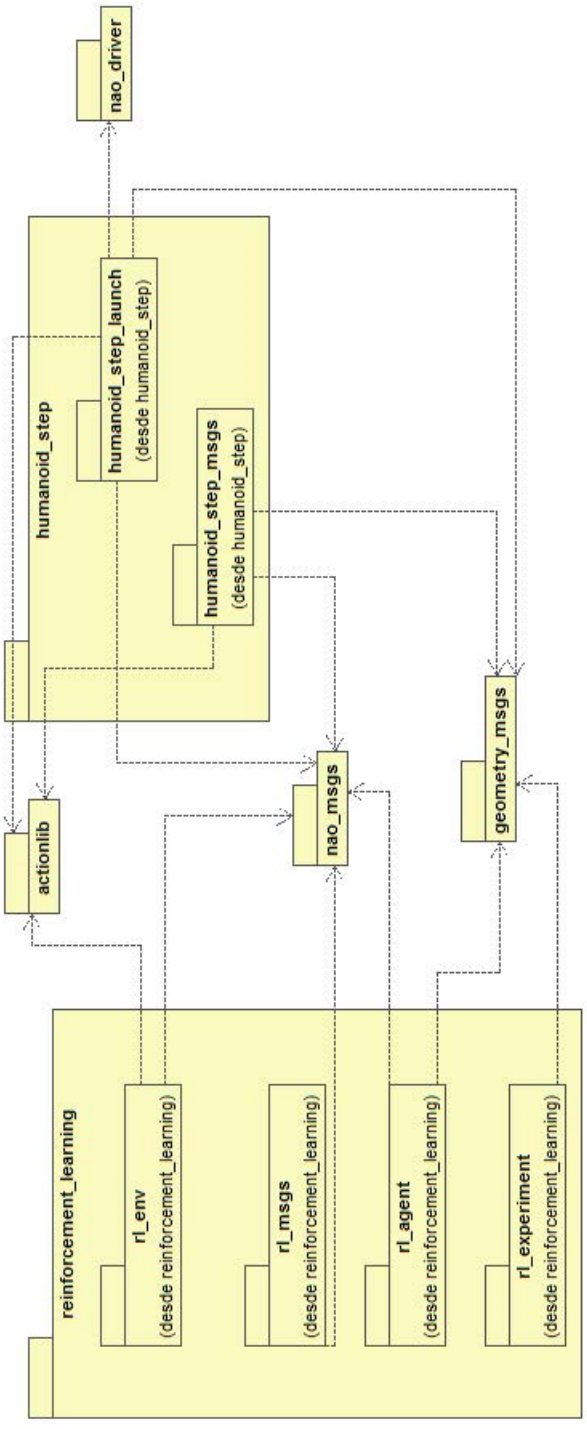


Figura 3.5: Diagrama de paquetes - Dependencias entre "Reinforcement Learning" y "humanoid_step", y los paquetes de *ROS*

3.3.2. Diagrama de clases

En esta subsección se describe la estructura de clases de los paquetes que forman parte de la arquitectura desarrollada.

En la figura 3.6 se muestran los paquetes que pertenecen a "Reinforcement Learning" que contienen clases: "rl_agent", "rl_env" y "rl_common". El resto de paquetes no son especificados en el diagrama puesto que en el caso del paquete "rl_experiment", dispone únicamente de un archivo ejecutable que utiliza los agentes y entornos definidos, y en el caso del paquete "rl_msgs", se incluye únicamente la definición de los mensajes y servicios (anexo B).

En el paquete "rl_agent" se incluye la definición de los agentes o algoritmos. Estas clases implementan la interfaz "Agent" que aporta una estructura única para todos los agentes, facilitando de este modo la definición de nuevos agentes. Entre los algoritmos disponibles se encuentra el *PI-SRL* ("PISRL1", primer paso y "PISRL2", segundo paso), el *PR-SRL*, *Sarsa* o *Q-Learning*, estos dos últimos están incluidos en el paquete de *ROS* utilizado.

El paquete "rl_env" incluye la definición de los entornos o problemas a resolver. Estas clases implementan la interfaz "Environment", ofreciendo una estructura propia para todos los entornos. En este caso, se dispone de un entorno que representa el comportamiento de caminar utilizado para el robot *NAO*.

Tanto el paquete "rl_agent" y "rl_env", disponen de un archivo ejecutable para lanzar sus nodos por separado.

El paquete "rl_common" dispone de las clases comunes al resto de paquetes. Entre ellas, se define la estructura de los agentes (interfaz "Agent") y de los entornos (interfaz "Environment"). Además, se incluyen estructuras de datos utilizadas por los agentes *PI-SRL* y *PR-SRL*, como es la base de casos (clase "BaseCasos"). También se incluye una definición propia de los árboles KD (clase "myKDTreeCasos"), utilizando la librería para *C++*[38].

En el caso de los paquetes "humanoid_step_launch" y "humanoid_step_msgs" no son especificados debido a que, el primero de ellos dispone únicamente de un archivo ejecutable y en caso del segundo, dispone de la definición de los mensajes y servicios utilizados en la comunicación.

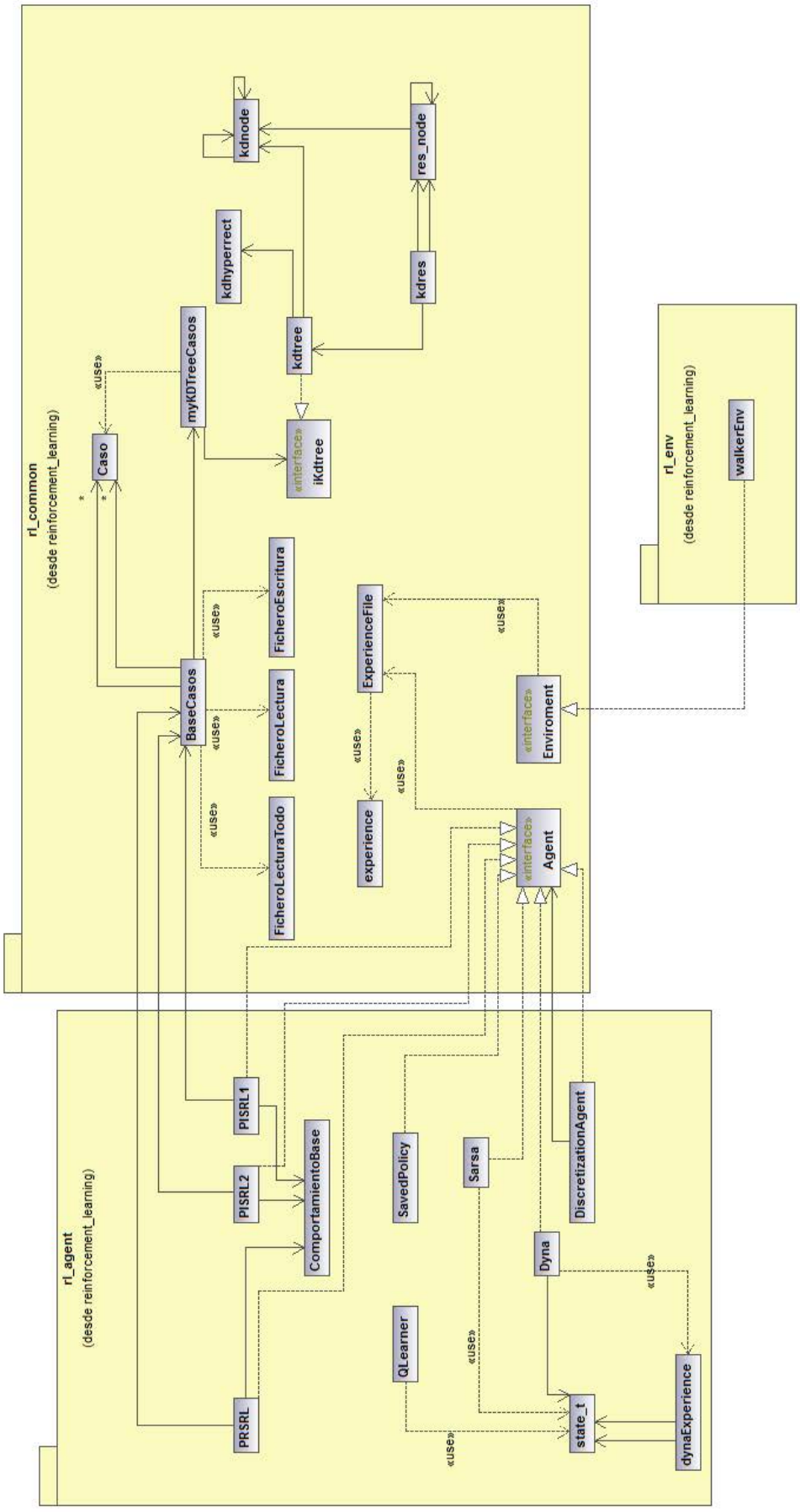


Figura 3.6: Diagrama de clases del paquete "Reinforcement Learning"

3.3.3. Estructura de comunicación

En las figuras 3.7 y 3.8 se muestra la estructura de comunicación, en ella se muestra los nodos involucrados (los círculos) y la conexión entre ellos mediante mensajes (rectángulos con el contorno continuo) o servicios (rectángulos con línea discontinua). Además, se muestra el *topic* con el que se registra en el sistema *ROS*.

Se dispone de dos estructuras de nodos. La primera de ellas consta de 2 nodos (figura 3.7), y la segunda consta de 3 nodos (figura 3.8).

La primera estructura (figura 3.7) dispone de un nodo que representa al agente y al entorno, denominado como "/rl_experiment", y un segundo nodo que se encarga de conectar con el robot *NAO*, nombrado como "/steps".

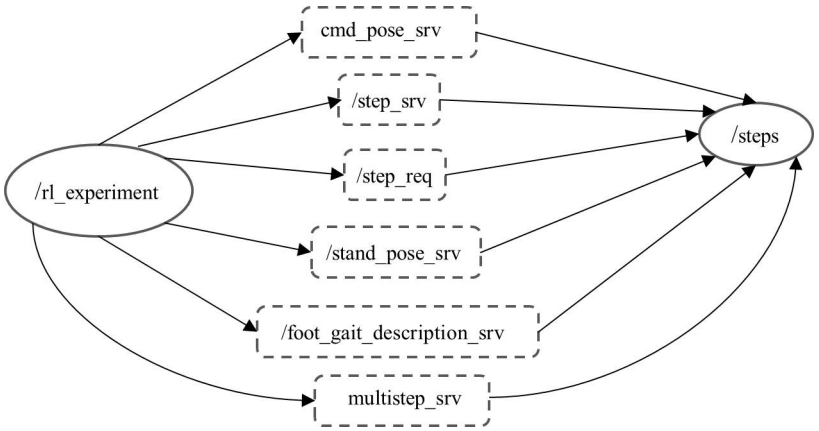


Figura 3.7: Estructura de comunicación con 2 nodos

La segunda estructura (figura 3.8) consta de un nodo que representa al agente ("/rl_agent"), otro nodo que representa al entorno ("/rl_env") y el nodo que conecta con el *NAO* ("/steps"), siendo el mismo nodo que en la primera estructura.

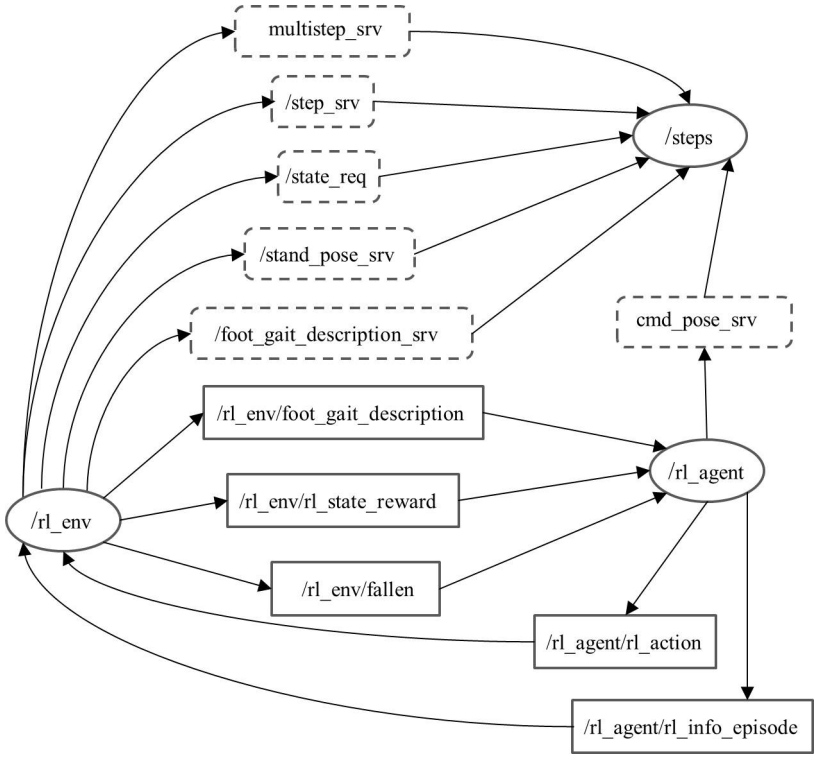


Figura 3.8: Estructura de comunicación con 3 nodos

En esta segunda estructura se observa que la comunicación entre el agente ("/rl_agent") y el entorno ("/rl_env") se realiza mediante intercambio de mensajes y en el caso de la comunicación con el conector del *NAO* ("/steps") se realiza mediante ejecución de servicios. Esta circunstancia se produce por la necesidad de realizar comunicaciones síncronas para la recuperación de información.

La descripción detallada de las comunicaciones de estas dos estructuras se muestra en el anexo C.

Capítulo 4

Experimentos

En este capítulo se muestran los resultados obtenidos en la utilización de los algoritmos *PI-SRL* y *PR-SRL* para el comportamiento de caminar en el robot *NAO* mediante la utilización de simulador y la validación en el robot real.

4.1. Entorno de los experimentos

Los experimentos se han realizado en el simulador proporcionado por la empresa *Aldebaran Robotics* con el robot *NAO*, llamado *NAOsim* en su versión 1.12.5. En el entorno utilizado en el simulador, se dispone de una habitación vacía en la cual se sitúa el robot *NAO*, el modelo utilizado es el *NAO V33 H25*, como se ilustra en la figura 4.1. En este entorno se permite el libre movimiento del robot sin tener problemas de obstáculos.

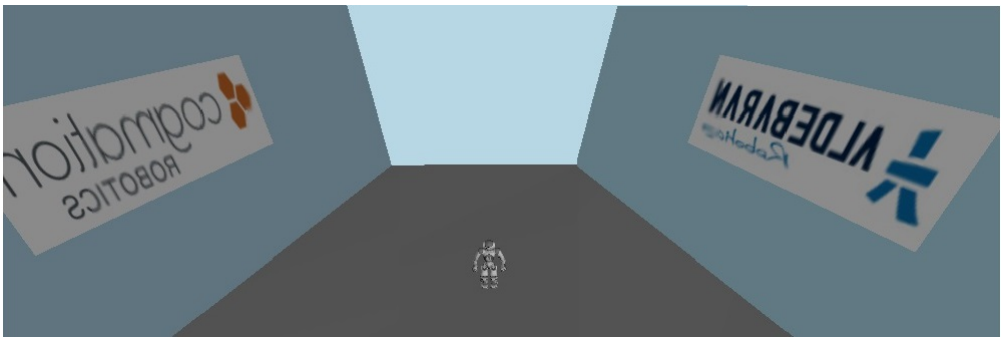


Figura 4.1: Entorno de los experimentos

Además el robot dispone de una postura fija para el comienzo de cada episodio (figura 4.2).

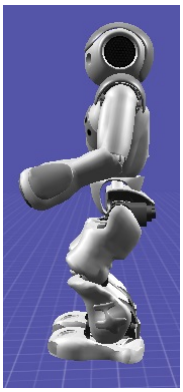


Figura 4.2: Posición de inicio de episodio

4.2. Resultados de los experimentos utilizando simulación

En esta sección se muestran los resultados obtenidos aplicando los algoritmos, el segundo paso del *PI-SRL* y el *PR-SRL*, para mejorar el comportamiento por defecto (configuración de paso por defecto) y rápido (configuración de paso máxima) predefinido en el robot *NAO*.

En estos experimentos, se ha definido una distancia de 0.5 metros para cada episodio. Esta distancia se ha establecido por las limitaciones del simulador, ya que recorriendo una distancia superior a 2 metros se observa un balanceo hacia atrás en el robot que provoca la caída del robot. Este efecto no se produce en el robot real, como se muestra en la sección 4.4.

Por otro lado, los parámetros de la función de actualización de Q-Learning son $\gamma = 0{,}9$ y $\alpha = 0{,}25$. Los valores de los parámetros de los algoritmos *PI-SRL* y *PR-SRL* son $\theta = 2{,}675 \times 10^{-1}$ (calculada mediante la ecuación 2.4), $\Theta = 5 \times 10^{-2}$ y en el caso de *PR-SRL*, el parámetro $k = 6$. En el caso del parámetro de riesgo, se prueba con dos valores, $\sigma = 9 \times 10^{-4}$ y $\sigma = 9 \times 10^{-3}$. Los parámetros del comportamiento base son de 0.1 metros de distancia y la configuración de los pasos se varía con la defecto y la máxima (tabla 2.1).

Las gráficas que se muestran se han realizado sobre un proceso de aprendizaje de 102 episodios y realizando medias sobre 5 diferentes ejecuciones para cada una de ellas.

En las siguientes gráficas se muestran por un lado, el comportamiento base (configuración mostrada anteriormente)y por otro lado, el comportamiento predefinido, donde el robot recorre la distancia fijada para un episodio (0.5 metros, en este caso).

4.2.1. Algoritmo *PI-SRL*

En esta sección se muestran los resultados obtenidos en los experimentos realizados con la segunda fase del algoritmo *PI-SRL*, la cual se inicia con la base de casos vacía.

Comportamiento de caminar con configuración por defecto

Las gráficas mostradas en esta sección se corresponden a la utilización de la configuración por defecto como comportamiento base del algoritmo *PI-SRL*.

En la gráfica 4.3 se muestra la evolución de la base de casos para dos configuraciones del parámetro de riesgo (σ). En ambas configuraciones se observa como el número de casos insertados se incrementan en los primeros episodios, y alrededor del episodio 30, el tamaño de la base de casos se estabiliza. Además, se observa un mayor tamaño para la configuración de riesgo mayor (línea verde), debido a una variación más pronunciada de las acciones con el valor alto del parámetro de riesgo ($\sigma = 9 \times 10^{-3}$), que requiere la inserción de más casos a la base de casos por llevar a cabo una exploración más agresiva del espacio.

En las gráficas 4.4a y 4.4b se muestra la comparación de los pasos realizados por la política de la base de casos (línea verde) y por el comportamiento base (línea roja) para las dos configuraciones del parámetro de riesgo. En los primeros episodios, los pasos son realizados en su totalidad por el comportamiento base, ya que se inicia con la base de casos vacía, pero en torno al episodio 10, la ejecución del comportamiento base es sustituido mayormente por la política de la base de casos.

Por otro lado, se observa como se produce una reducción en el número de pasos totales de los episodios, comenzando con un total de 25 pasos para recorrer la distancia de 0.5 metros y con el paso de los episodios, se reduce en torno a 15 pasos, en el caso de la configuración alta del parámetro de riesgo ($\sigma = 9 \times 10^{-3}$) y alrededor de 16 pasos para la configuración baja ($\sigma = 9 \times 10^{-4}$).

En la gráfica 4.5 se muestra el refuerzo acumulado (representado por la velocidad) por episodio para el comportamiento predefinido (línea morada) y para el comportamiento base (línea azul), utilizando la configuración de

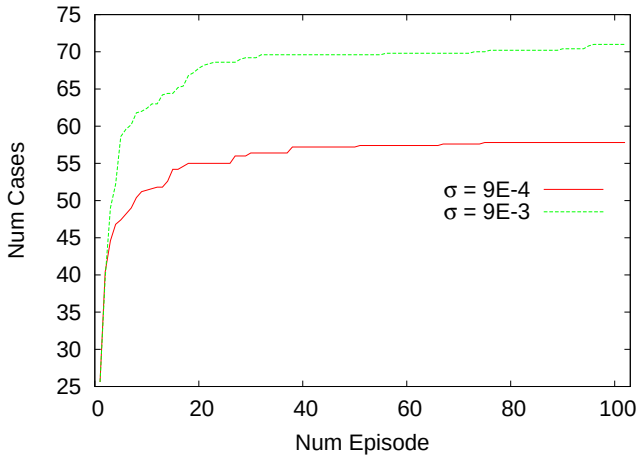
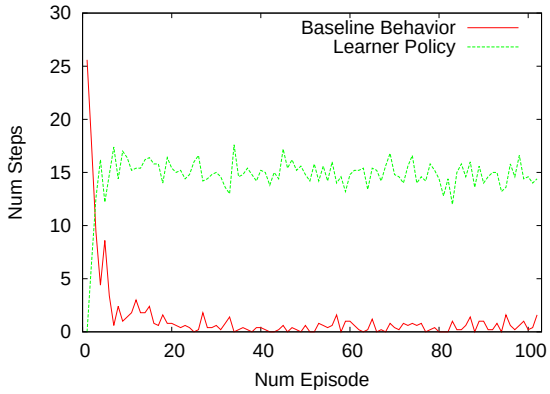
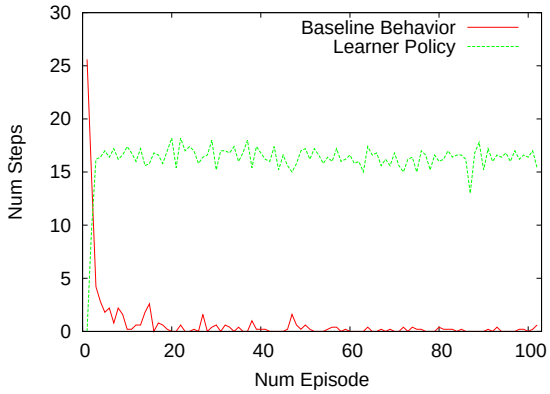


Figura 4.3: *PI-SRL* - Tamaño de la base de casos por episodio para diferentes configuraciones del riesgo (σ) usando la configuración por defecto del comportamiento base

paso por defecto. Para este comportamiento se utilizan dos configuraciones del parámetro de riesgo, con un valor alto (línea verde) y otro bajo (línea roja). En este caso, se observa como la configuración alta del parámetro de riesgo ($\sigma = 9 \times 10^{-3}$) produce mejores episodios que la configuración baja ($\sigma = 9 \times 10^{-4}$), mejorando ambos casos el comportamiento base del que parten. En el caso del comportamiento predefinido, el valor alto mejora a dicho comportamiento en la mayoría de los episodios del proceso de aprendizaje y con el valor bajo de riesgo no se consigue mejorar el comportamiento predefinido, exceptuando un número reducido de episodios.



(a)



(b)

Figura 4.4: *PI-SRL* - Número de pasos por episodio realizados por el comportamiento base y la política de la base de casos usando diferentes configuraciones del parámetro de riesgo: (a) $\sigma = 9 \times 10^{-3}$ (b) $\sigma = 9 \times 10^{-4}$

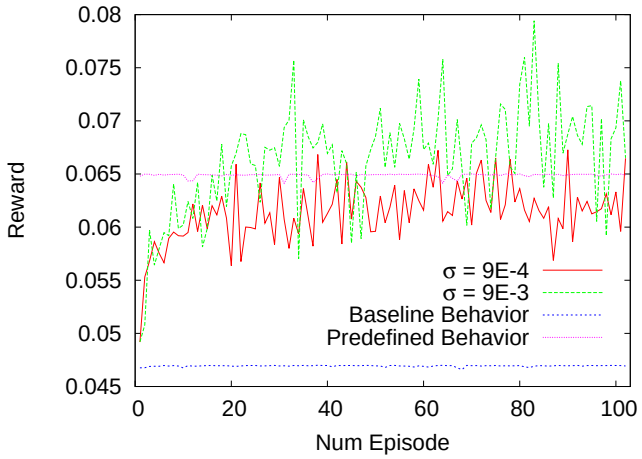


Figura 4.5: *PI-SRL* - Refuerzo acumulado por episodio para diferentes configuraciones del parámetro de riesgo (σ) usando la configuración por defecto del comportamiento

Comportamiento de caminar con configuración máxima

Las gráficas mostradas en esta sección se corresponden a la utilización de la configuración máxima como comportamiento base.

En la gráfica 4.6 se observa como el tamaño de la base de casos aumenta significativamente durante los primeros 10 episodios aproximadamente y a partir de ese episodio el crecimiento de la base de casos es menos pronunciada. Otro aspecto que se observa, es que la configuración con mayor valor del parámetro de riesgo (línea verde) produce una base de casos mayor que con el valor bajo (línea roja).

La gráfica 4.7 muestra los pasos ejecutados por el comportamiento base (línea roja) y por la política de la base de casos (línea verde) para diferentes configuraciones del parámetro de riesgo. Las curvas del proceso de esta configuración es similar a la que se produce con el uso del comportamiento de caminar por defecto. La principal diferencia es el número de pasos que se realizan por episodio, en este caso se inicia con 16 pasos y se reduce a 9 pasos por episodio para cada una de las configuraciones del parámetro de riesgo.

En la gráfica 4.8 se muestra el refuerzo acumulado por episodio del comportamiento máximo de caminar utilizando diferentes configuraciones del parámetro de riesgo. En este caso, ambas configuraciones describen un comportamiento similar en el proceso de aprendizaje y además, ambos mejoran

significativamente tanto el comportamiento base como el predefinido del robot.

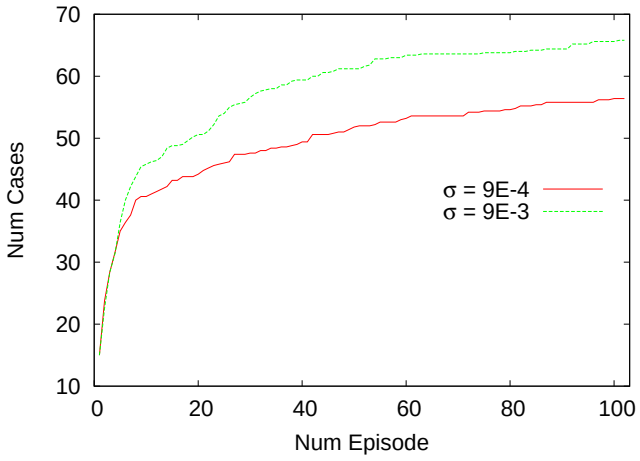
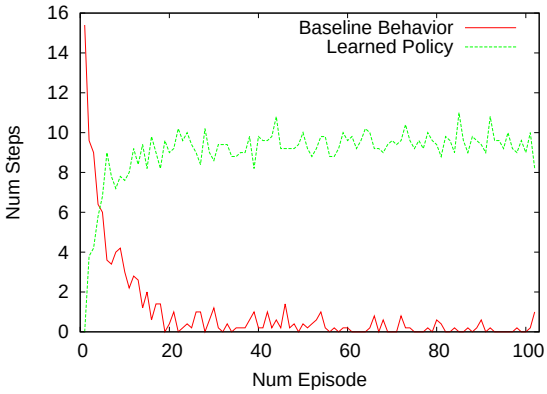
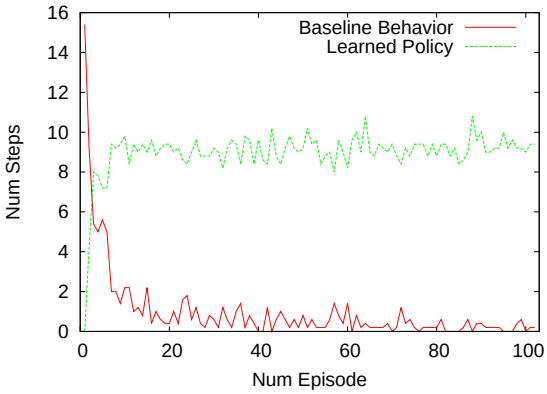


Figura 4.6: *PI-SRL* - Tamaño de la base de casos por episodio para diferentes configuraciones del riesgo (σ) usando la configuración máxima del comportamiento base



(a)



(b)

Figura 4.7: *PI-SRL* - Número de pasos por episodio realizados por el comportamiento base y la política de la base de casos usando diferentes configuraciones del parámetro de riesgo: (a) $\sigma = 9 \times 10^{-3}$ (b) $\sigma = 9 \times 10^{-4}$

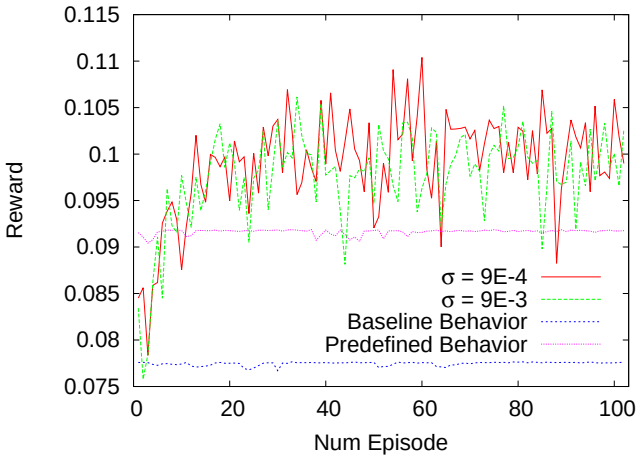


Figura 4.8: *PI-SRL* - Refuerzo acumulado por episodio para diferentes configuraciones del parámetro de riesgo (σ) usando la configuración máxima del comportamiento base

4.2.2. Algoritmo *PR-SRL*

En esta sección se muestran los resultados obtenidos en los experimentos realizados con el algoritmo *PR-SRL* para las dos diferentes configuraciones del comportamiento de caminar del robot *NAO*, por defecto o máxima.

Comportamiento de caminar con configuración por defecto

Las gráficas mostradas en esta sección se corresponden a la utilización de la configuración por defecto como comportamiento base del algoritmo *PR-SRL*.

En la gráfica 4.9 se observa la evolución de la base de casos para las diferentes configuraciones del parámetro de riesgo (σ). En ella se observa como el tamaño de la base de casos aumenta con el ejecución de los episodios. Este tamaño de la base de casos es mayor para el valor de riesgo alto ($\sigma = 9 \times 10^{-3}$), es debido a que aplica una mayor variación en las acciones y eso lleva a nuevos estados que son muy diferentes a los disponibles en la base de casos.

En la gráfica 4.10 se muestra la utilización del comportamiento base (línea roja) y de la política aprendida en la base de casos (línea verde), incluyendo para dos configuraciones del parámetro de riesgo (σ). Al principio del proceso

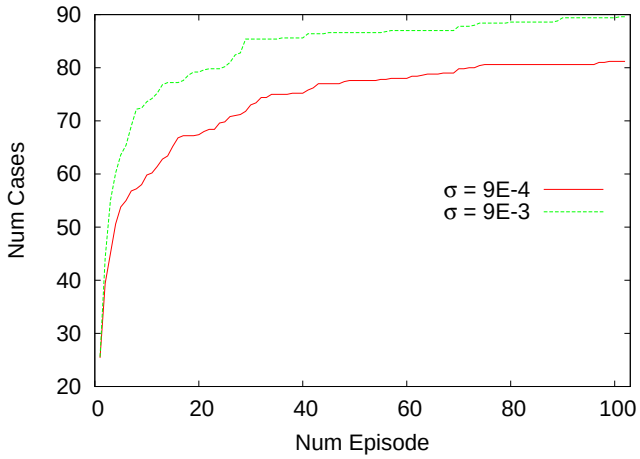


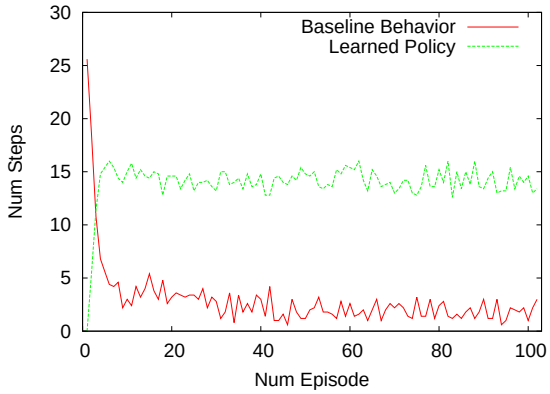
Figura 4.9: *PR-SRL* - Tamaño de la base de casos por episodio para diferentes configuraciones del riesgo (σ) usando la configuración por defecto del comportamiento base

de aprendizaje (base de casos vacía), todos los pasos son ejecutados por el comportamiento base. Sin embargo, con el avance del proceso el comportamiento base es reemplazado por la política de la base de casos. Comparando ambas configuraciones, con el valor alto de riesgo ($\sigma = 9 \times 10^{-3}$) se ejecuta más veces el comportamiento base que con el valor bajo ($\sigma = 9 \times 10^{-4}$), ya que se realiza una exploración del espacio de estados y de acciones más agresiva.

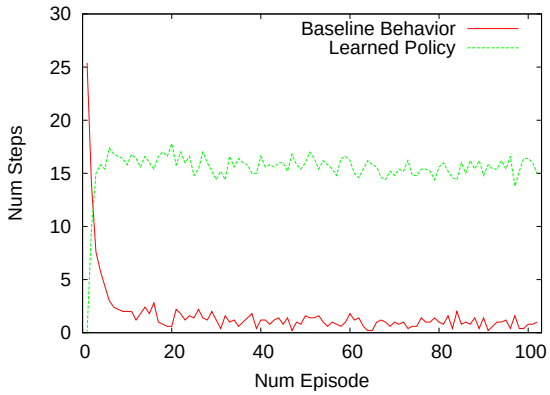
Por otro lado, se aprecia como el número total de pasos del episodio se reduce, ya que al principio del episodio se encuentra en torno a 25 pasos por episodio para recorrer 0.5 metros y con la evolución del proceso, se utilizan alrededor de 16 pasos para recorrer la misma distancia. Esta circunstancia se debe a que las acciones aprendidas tienden a alargar los pasos, lo que conlleva una reducción en el número de pasos del episodio.

En la gráfica 4.11 se muestra el refuerzo acumulado para las dos configuraciones de riesgo diferentes. En este caso, ambas configuraciones mejoran el comportamiento base utilizado en el proceso de aprendizaje. Por otro lado, la configuración alta del parámetro de riesgo ($\sigma = 9 \times 10^{-3}$) obtiene mejores resultados que la configuración baja del parámetro de riesgo ($\sigma = 9 \times 10^{-4}$), esto se observa comparando con el comportamiento predefinido. Éste es mejorado por la valor alto de riesgo alto en la mayoría de los episodios y en un número reducido de episodios en el caso del valor bajo de riesgo, no obte-

niendo un comportamiento mejor con esta última configuración.



(a)



(b)

Figura 4.10: *PR-SRL* - Número de pasos por episodio realizados por el comportamiento base y la política de la base de casos usando diferentes configuraciones del parámetro de riesgo: (a) $\sigma = 9 \times 10^{-3}$ (b) $\sigma = 9 \times 10^{-4}$

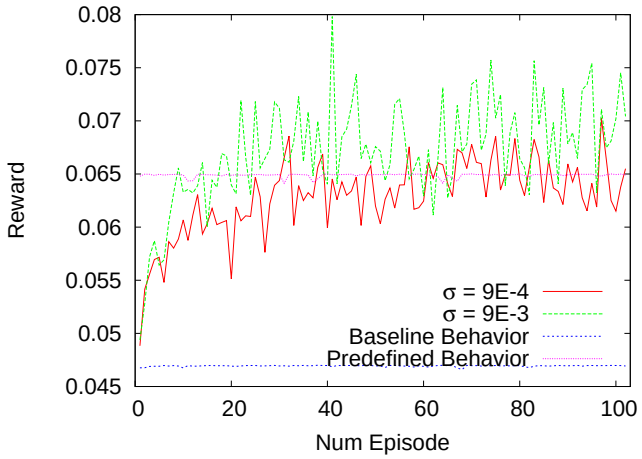


Figura 4.11: *PR-SRL* - Refuerzo acumulado por episodio para diferentes configuraciones del parámetro de riesgo (σ) usando la configuración por defecto del comportamiento base

Comportamiento de caminar con configuración máxima

Las gráficas mostradas en esta sección se corresponden a la utilización de la configuración máxima como comportamiento base del algoritmo *PR-SRL*.

La gráfica 4.12 muestra el tamaño de la base de casos para diferentes configuraciones del parámetro de riesgo. Como en el caso del comportamiento por defecto, la configuración alta del parámetro de riesgo ($\sigma = 9 \times 10^{-3}$) produce una base de casos de mayor tamaño que la configuración baja ($\sigma = 9 \times 10^{-4}$). La principal diferencia entre esta gráfica y la gráfica 4.9 es el tamaño de la base de casos, ya que en el primero de los casos se dispone de un tamaño máximo en torno a 65 casos y en el segundo, se disponde de un tamaño máximo alrededor de 90 casos. Esta diferencia en el tamaño de la base de casos se debe a que con la configuración máxima de los pasos, la exploración del espacio de estados y acciones es mucho menor que con la configuración por defecto.

La gráfica 4.13 muestra el número de pasos ejecutados por el comportamiento base (línea roja) y la política aprendida en la base de casos (línea verde). Estas gráficas se comportan del mismo modo que en el caso de la gráfica 4.10. La principal diferencia es el número de pasos que se realiza en cada episodio, ya que en este caso se inicia el proceso con episodios de 15

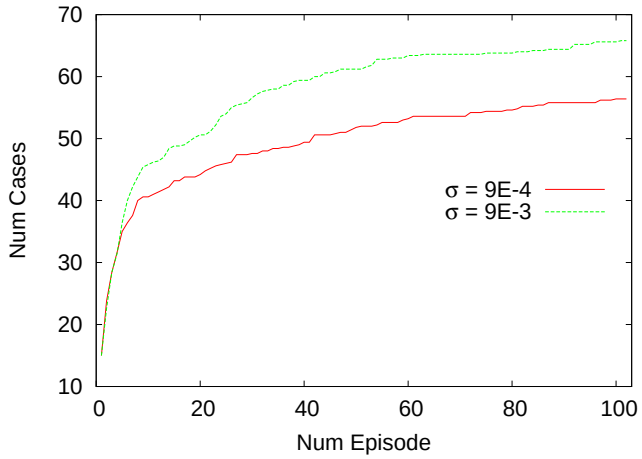
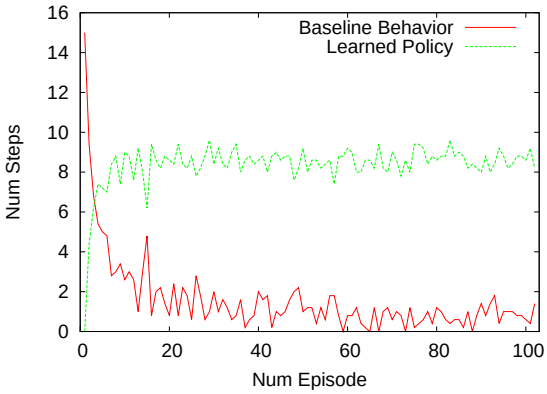


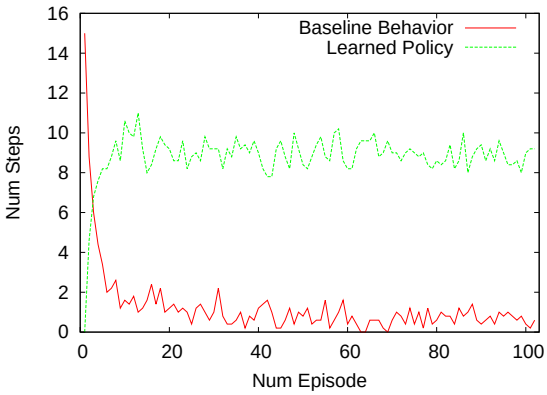
Figura 4.12: *PR-SRL* - Tamaño de la base de casos por episodio para diferentes configuraciones del riesgo (σ) usando la configuración máxima del comportamiento base

pasos y se reduce en torno a 8 pasos en el caso de la configuración alta del parámetro de riesgo ($\sigma = 9 \times 10^{-3}$) y alrededor de 9 para la configuración baja ($\sigma = 9 \times 10^{-4}$).

En la gráfica 4.14 se muestra la velocidad obtenida por episodio para las configuraciones del parámetro de riesgo. En este caso, ambas producen mejores resultados que el comportamiento base y el predefinido del robot. Comparando las dos configuraciones del parámetro de riesgo las gráficas son similares, pero se aprecia una mayor variabilidad en los periodos con la configuración baja del parámetro de riesgo ($\sigma = 9 \times 10^{-4}$).



(a)



(b)

Figura 4.13: *PR-SRL* - Número de pasos por episodio realizados por el comportamiento base y la política de la base de casos usando diferentes configuraciones del parámetro de riesgo: (a) $\sigma = 9 \times 10^{-3}$ (b) $\sigma = 9 \times 10^{-4}$

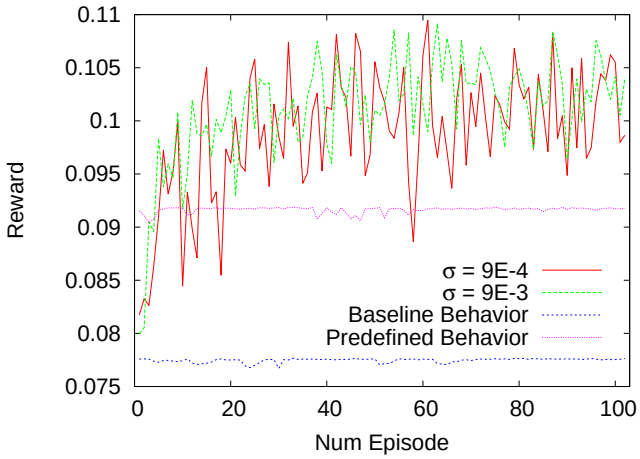


Figura 4.14: *PR-SRL* - Refuerzo acumulado por episodio para diferentes configuraciones del parámetro de riesgo (σ) usando la configuración máxima del comportamiento base

4.3. Comparativa entre *PI-SRL* y *PR-SRL*

En esta sección se compara los resultados obtenidos de los dos algoritmos utilizados, *PI-SRL* y *PR-SRL*.

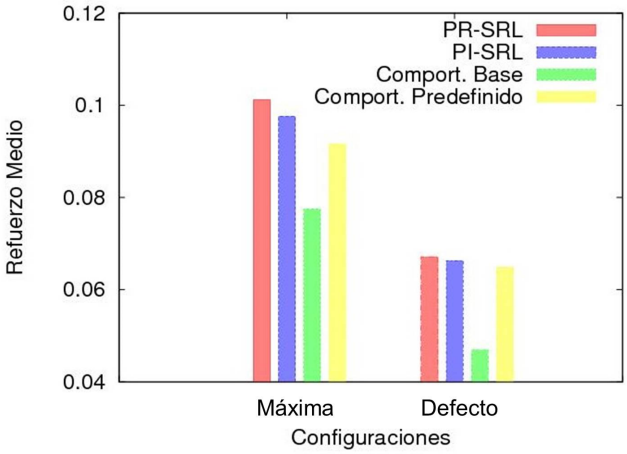
En la figura 4.15 se muestra la comparativa del refuerzo medio obtenido por los algoritmos *PI-SRL* (barra roja) y *PR-SRL* (barra azul), por el comportamiento base (barra verde) y el predefinido (barra amarilla), para diferentes configuraciones del parámetro de riesgo. En ella se observa como el refuerzo medio (la velocidad media obtenida en una distancia de 0.5 metros) obtenido por los algoritmos mejora significativamente al comportamiento base utilizado. Estas mejoras están entorno de 2,5 cm/seg para cada una de las configuraciones de pasos utilizada.

En el caso del comportamiento predefinido, la mejora depende de la configuración de pasos y del valor del parámetro de riesgo. En el caso de un valor alto de la función de riesgo la política aprendida por ambos algoritmos es mejor que el comportamiento predefinido del robot, mejorando en mayor proporción para la configuración máxima. Esta mejora se debe a la obtención de pasos que producen un desplazamiento mayor en el robot y así se reduce el número de pasos para recorrer la distancia fijada. En el caso de un valor bajo de riesgo, se aprecia dos situaciones, por un lado se mejora cuando se

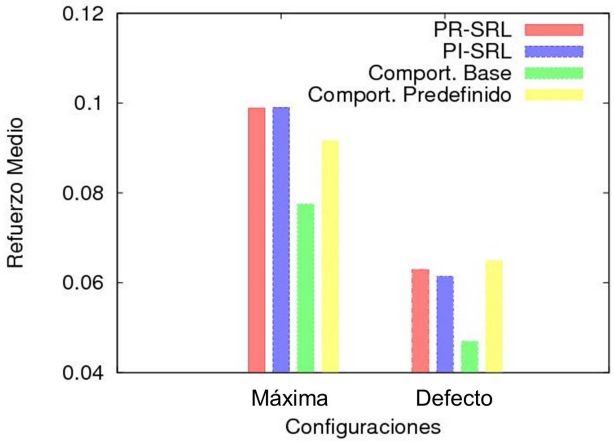
utiliza la configuración máxima y por otro lado con la configuración por defecto no se consigue mejorar el comportamiento predefinido del robot, lo que se observaba en el apartado anterior.

Por otro lado, comparando los diferentes algoritmos se aprecia como el algoritmo *PR-SRL* obtiene un refuerzo medio mayor o igual que el *PI-SRL*, produciendo un comportamiento general mejor. Por otra parte, para obtener ese mejor rendimiento el algoritmo *PR-SRL* necesita una base de casos de mayor tamaño, debido a la utilización de la función de riesgo continua en la exploración del espacio, la cual produce una exploración más permisiva.

En resumen, el algoritmo *PR-SRL* obtiene un mejor comportamiento en este dominio que el algoritmo *PI-SRL*, pero ésto conlleva la utilización de una base de casos mayor debido a una exploración más conservadora producida por la función de riesgo continua. Por otro lado, el valor alto del parámetro de riesgo ($\sigma = 9 \times 10^{-3}$) produce mejores políticas siempre que no se disponga de límites superiores, como en el caso del comportamiento máximo.



(a)



(b)

Figura 4.15: Comparativa de los algoritmos *PI-SRL*, *PR-SRL*, comportamiento base y predefinido utilizando la media del refuerzo acumulado usando diferentes configuraciones del parámetro de riesgo: (a) $\sigma = 9 \times 10^{-3}$ (b) $\sigma = 9 \times 10^{-4}$

4.4. Validación del comportamiento de caminar en el robot *NAO* real

En esta sección se evalúa en el robot *NAO* real el funcionamiento de los comportamientos aprendidos en el simulador.

En esta validación se utilizan las bases de casos obtenidas en el simulador por el algoritmo *PR-SRL* con el parámetro de riesgo $\sigma = 9 \times 10^{-3}$. En este caso, se ha probado con periodos de dos distancias, como son 0.5 metros (la distancia utilizada en el proceso de aprendizaje en el simulador) y 2 metros. Para cada distancia se han realizado 5 ejecuciones realizando la media entre ellas para generar las gráficas.

En la figura 4.16 se muestra el refuerzo obtenido con la política aprendida y el comportamiento predefinido para diferentes configuraciones de paso para una distancia de 0.5 metros. En ella se observa que la política aprendida mejora al comportamiento predefinido en el robot *NAO* independientemente de la configuración de paso utilizada. Si se comparan estos resultados con los obtenidos en el simulador, con la configuración por defecto se consigue mejorar en este caso mientras que en el simulador no era posible. Ésto se debe principalmente a la diferencia que existe entre los estados del simulador y del robot real (figura 4.18), lo que afecta a la política aprendida.

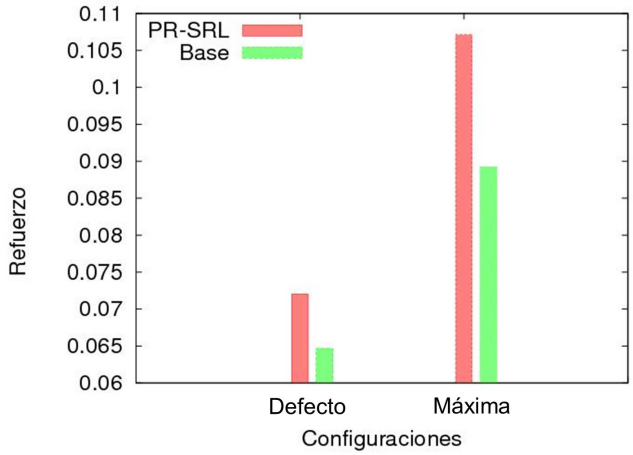


Figura 4.16: Comparativa utilizando el robot real entre el comportamiento base y la política aprendida con el algoritmo *PR-SRL* para una distancia de 0.5 metros

En la figura 4.17 se muestra el refuerzo obtenido con la política aprendida

y el comportamiento predefinido para diferentes configuraciones de paso para una distancia de 2 metros. En este caso se observa que, para el comportamiento por defecto la política aprendida mejora a la predefinida pero en el caso de la máxima, la política no es capaz de mejorar dicho comportamiento.

Esta circunstancia del caso máximo se debe a la distancia del comportamiento base utilizada (0.1 metros) en el proceso de aprendizaje. Con esta configuración se producen pasos de 0.049 metros mientras que en el caso de una distancia de 2 metros (la mostrada en la gráfica) se producen pasos de 0.077 metros. Aunque en el proceso de aprendizaje se mejoren los pasos (como se muestra en la sección 4.2.2) no es suficiente para mejorar el comportamiento máximo predefinido para una distancia de 2 metros. En el caso de la configuración por defecto, no ocurre puesto que la diferencia entre los pasos utilizados en el comportamiento base (0.025 metros) y los del comportamiento predefinido (0.040 metros) es menor que en el caso máximo, siendo subsanado por el proceso de aprendizaje. Además, el resultado también se ve afectado por la diferenciación de los estados entre simulación y robot real.

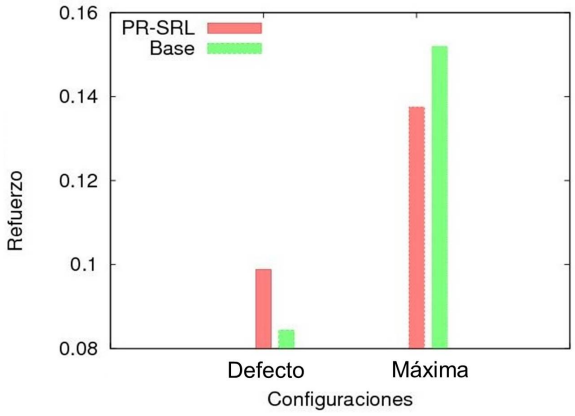


Figura 4.17: Comparativa utilizando el robot real entre el comportamiento base y la política aprendida con el algoritmo *PR-SRL* para una distancia de 2 metros

Por otro lado, en la figura 4.18 se muestra la distancia entre los estados del simulador y del robot real con los estados de la base de casos generada en simulación. En ella se observa como la distancia entre los estados del robot real y los de la base de casos es mucho mayor que en el caso de los estados del entorno simulado. Ésto se debe a que los estados que se obtienen en el entorno simulado son diferentes a los entorno real. Esta situación afec-

ta al funcionamiento de la política aprendida en el robot real, produciendo diferencias de la política realizada en el simulador.

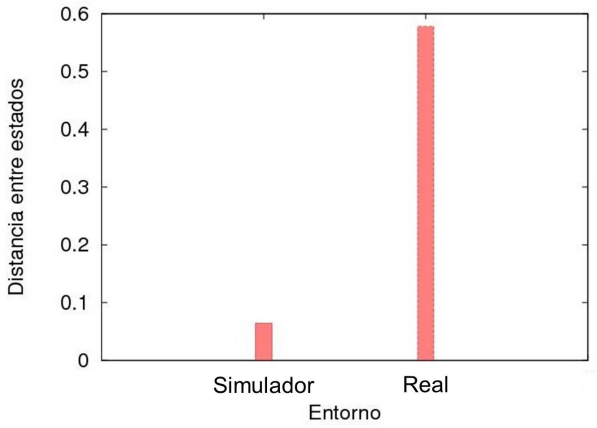


Figura 4.18: Comparativa entre el entorno simulado y el real en relación a la distancia entre los estados recibidos y el más cercano de la base de casos

Capítulo 5

Conclusiones

En esta sección se exponen las conclusiones de los resultados obtenidos utilizando técnicas de *RL* para el comportamiento de caminar del robot *NAO*. Además, se incluyen otras líneas de investigación en relación con este proyecto.

5.1. Conclusiones del proyecto

En este proyecto se ha conseguido mejorar el comportamiento predefinido de caminar del robot *NAO* utilizando técnicas de aprendizaje por refuerzo seguro, más concretamente con los algoritmos *PI-SRL* y *PR-SRL*.

Los resultados obtenidos por ambos algoritmos son similares pero ligeramente mejores para el algoritmo *PR-SRL*. Pero esa mejora conlleva un mayor tamaño de la base de casos, al realizarse una exploración más permisiva con los nuevos estados.

Además, el valor del parámetro de riesgo condiciona el funcionamiento de los algoritmos produciendo diferentes comportamientos. En este dominio, se ha comprobado que un mayor valor de dicho parámetro produce un comportamiento mejor.

En relación a los entornos utilizados para el aprendizaje y validación del comportamiento, se ha podido comprobar que las bases de casos generadas en el proceso de simulación no son adecuadas para el uso en el robot real, ya que los estados difieren demasiado lo que produce variaciones en relación a la política aprendida en el proceso de simulación.

En resumen, el algoritmo *PR-SRL* es el algoritmo más adecuado para este dominio, produciendo caminatas con mayor velocidad en la mayoría de las configuraciones. Además, la utilización de un parámetro de riesgo mayor produce mejores comportamientos. Por último, se debe diferenciar las

políticas aprendidas mediante simulación y robot real, ya que los estados difieren demasiado lo que provoca que no se refleje realmente la política aprendida, aunque en este caso se han obtenido resultados que mejoran los comportamientos predefinidos del robot real.

5.2. Líneas futuras

En esta sección se muestran diferentes líneas de investigación relativas a la utilización de algoritmos de aprendizaje automático para comportamientos de cualquier robot:

- Aplicar otros algoritmos de aprendizaje automático a la representación del comportamiento de caminar desarrollado en este proyecto, como puede ser *Q-Learning*.
- Comprobar el comportamiento de los algoritmos para un caminar omnidireccional. Este comportamiento completaría a este proyecto, ya que en este caso solo se tiene en cuenta la posición y desplazamiento frontal.
- Mejorar el comportamiento de caminar evitando desviaciones en la dirección del movimiento mediante la utilización de los algoritmos de aprendizaje por refuerzo seguro.
- Definir un comportamiento de caminar utilizando la abstracción baja en los algoritmos *PI-SRL* y *PR-SRL*.
- Definición de otros comportamientos para su utilización con algoritmos de aprendizaje por refuerzo seguro, como pueden ser recorrer una habitación con obstáculos, o coger un objeto, entre otros.

Apéndice A

Gestión del proyecto

En este apéndice se detalla la planificación y las fases del proyecto, los componentes hardware y software utilizados en el desarrollo del proyecto y por último, se detalla el presupuesto.

A.1. Fases del proyecto

En esta sección se muestra la planificación del proyecto utilizando el diagrama de Gantt. El proyecto se ha dividido en 5 fases:

1. **Análisis**: se corresponde con el estudio de los elementos relacionados con el proyecto, como son el software *ROS*, el robot *NAO* y su software, y el comportamiento de caminar en robots humanoides, concretando en el robot *NAO*.

2. **Diseño de la arquitectura**: se corresponde con el diseño de la estructura del software a implementar y del dominio utilizado para la representación del comportamiento de caminar.

3. **Implementación**: se corresponde con la realización del código en *C++* y *python* utilizando el diseño realizado en la fase anterior.

4. **Experimentación**: se lleva a cabo las pruebas del software implementado.

5. **Documentación del proyecto**: realización de la documentación y presentación de este proyecto.

En la figura A.1 se muestra la descripción de las tareas mostrando la duración de cada una de ellas y en la figura A.2 se muestra el diagrama de Gantt asociado a las fases mostradas.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesor
1	**Proyecto Fin de Carrera**	**199,38 días**	**08/10/12**	**17/07/13**	
2	**Analisis**	**53,38 días**	**08/10/12**	**20/12/12**	
3	ROS	65 horas	08/10/12	13/11/12	
4	NAO	35 horas	13/11/12	30/11/12	3
5	Caminar Robot NAO	50 horas	30/11/12	20/12/12	4
6	**Diseño del framework**	**48,5 días**	**20/12/12**	**25/02/13**	
7	Diseño del Dominio	20 horas	20/12/12	10/01/13	5
8	Diseño Framework	120 horas	10/01/13	25/02/13	7
9	**Implementación**	**39,5 días**	**25/02/13**	**19/04/13**	
10	Implementación	100 horas	25/02/13	19/04/13	8
11	**Experimentación**	**61,5 días**	**19/04/13**	**12/07/13**	
12	Ejecución de pruebas	120 horas	19/04/13	22/05/13	10
13	**Documentación del proyecto**	**41,5 días**	**22/05/13**	**17/07/13**	
14	Documentación del sistema	60 horas	22/05/13	12/07/13	12
15	Presentación del proyecto	10 horas	12/07/13	17/07/13	14

Figura A.1: Planificación - Descripción de tareas

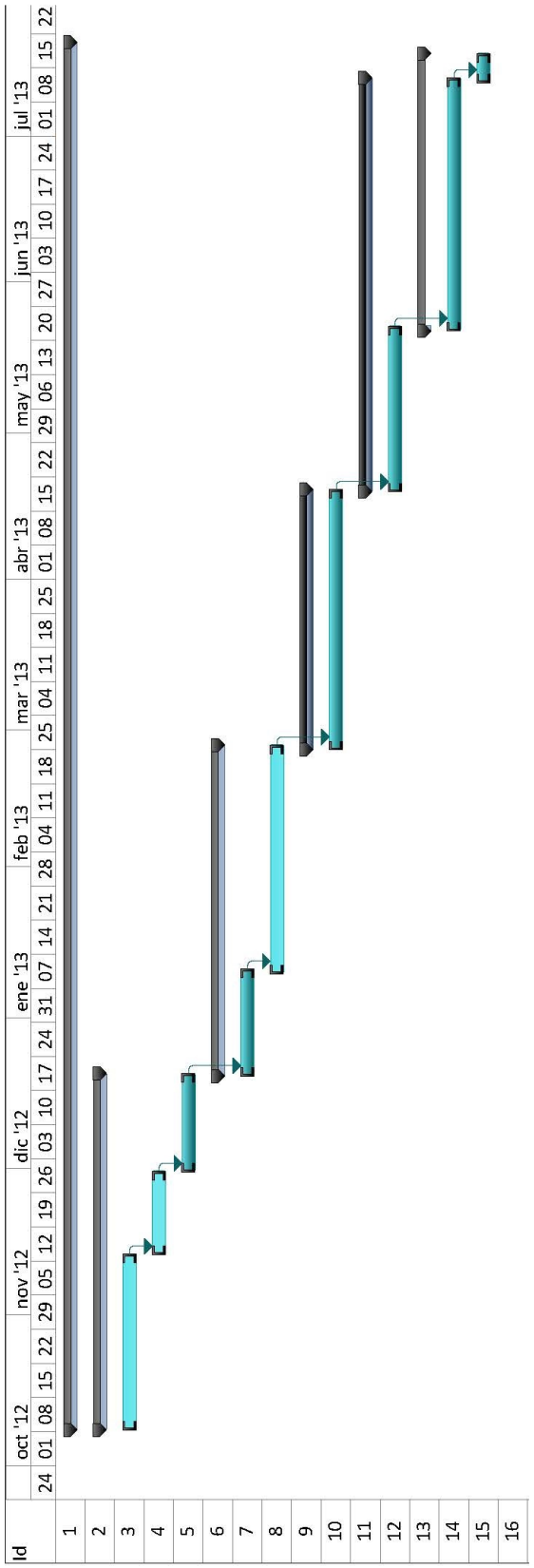


Figura A.2: Planificación - Diagrama de Gantt

A.2. Medios utilizados

En esta sección se describen los componentes hardware y software durante el desarrollo de este proyecto.

En primer lugar, se detalla los componentes hardware utilizados para el desarrollo del proyecto.

- **Notebook HP Pavilion dv6-1115es**, Intel Core 2 Duo T6400 @ 2GHz.
- **Robot *NAO* v33 H25**
- **Router Comtrend**

Por otro lado, se muestra los componentes software utilizados en el proyecto.

- **Windows 7 Profesional**
- **Ubuntu 11.10 Natty**
- **Oracle VM VirtualBox**
- **Microsoft Project 2010**
- **Altova UModel 2013**
- **Simulador *NAOsim***
- ***NAOqi***
- **ROS**
- **gnuplot**
- **Latex y editor Texmaker**

En el caso de los software, se han utilizado bajo licencia universitaria en el caso de *Windows*, *Microsoft Project*, *Altova UModel* o simulador *NAOsim*. En el caso del resto de software, son de libre distribución.

A.3. Presupuesto

En esta sección se muestra el presupuesto del proyecto, para su confección se ha utilizado la plantilla disponible para los proyectos de fin de carrera.

En la tabla A.4 se muestra el presupuesto total del proyecto. El presupuesto está compuesto por el gasto de personal (A.1), de equipos (A.2), de costes de funcionamiento (A.3) y de gastos indirectos, que se corresponde con el 20 % de los demás gastos.

Categoría	**Dedicación (hombre mes)**[a]	**Coste (hombre mes)**	**Coste(€)**
Jefe de Proyecto	0,92	2.229,48	2.051,12
Analista	1,14	1.843,11	2.101,15
Diseñador	1,07	1.727,28	1.848,19
Programador	1,68	2.229,48	1.971,61
		Total	**7.972,07**

[a]1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)

Tabla A.1: Presupuesto - Costes en personal

El cálculo de la amortización de los equipos se realiza mediante la fórmula A.1.

$$\frac{A}{B} \times C \times D \tag{A.1}$$

Siendo,

- A: número de meses desde la fecha de facturación en que el equipo es utilizado.
- B: periodo de depreciación (60 meses).
- C: coste del equipo (sin IVA).
- D: Tanto por cierto de uso que se dedica al proyecto.

Descripción	**Coste(€)**	**Uso dedicado**	**Dedicación (meses)**	**Periodo de depreciación**	**Coste imputable**
Notebook hp pavilion dv6-1115es	560,00	100	9	60	84,00
Robot *NAO*	12.266,12	100	1	60	204,44
Router Comtrend	30,00	100	3	60	1,50
				Total	**289,94**

Tabla A.2: Presupuesto - Costes en equipos

Descripción	**Empresa**	**Coste computable(€)**
Material Fungible	Papelería	15,00
	Total	**15,00**

Tabla A.3: Presupuesto - Otros costes directos

Concepto Costes	**Presupuesto Costes Totales**
Personal	7.972,00
Amortización	290,00
Costes de funcionamiento	15,00
Costes indirectos	1.655,00
Total	**9.932,00**

Tabla A.4: Presupuesto Total

Apéndice B

Mensajes y servicios

En este anexo se detalla la estructura de los mensajes y servicios que son utilizados para la comunicación entre los diferentes nodos de la ejecución.

B.1. Mensajes

En este apartado, se definen los mensajes que se utilizan para la comunicación entre los nodos. Estos mensajes pueden ser utilizados por un servicio o enviados directamente entre nodos.

La descripción de los mensajes se realiza según el paquete perteneciente e incluyendo únicamente los involucrados en el desarrollo de la arquitectura.

B.1.1. Paquete *geometry_msgs*

Este paquete dispone de los mensajes para expresar primitivas geométricas como pueden ser puntos, vectores o posiciones[37].

- **Mensaje Pose2D**
  - **Descripción**: indica la posición y la orientación expresada en un sistema de referencia bidimensional.
  - **Contenido**:
    - **Header header**: la cabecera que contiene campos de metadatos comunes, como son el timestamp y un identificador de paquete.
    - **float64 x**: la posición en torno al eje x.
    - **float64 y**: la posición en torno al eje y.
    - **float64 theta**: la orientación de la posición.

B.1.2. Paquete *nao_msgs*

Este paquete contiene los mensajes relativos al robot NAO[36].

- **Mensaje TorsoOdometry**
  - **Descripción**: indica la estimación realizada por la odometría del torso del NAO.
  - **Contenido**:
    - **Header header**: la cabecera que contiene campos de metadatos comunes, como son el timestamp y un identificador de paquete.
    - **float32 x**: la posición del cuerpo en relación al eje x en el marco de la odometría.
    - **float32 y**: la posición del cuerpo en relación al eje y en el marco de la odometría.
    - **float32 z**: la posición del cuerpo en relación al eje z en el marco de la odometría.
    - **float32 wx**: la posición del cuerpo entorno al eje x.
    - **float32 wy**: la posición del cuerpo entorno al eje y.
    - **float32 wz**: la posición del cuerpo entorno al eje z.

B.1.3. Paquete *humanoid_step_msgs*

Este paquete contiene mensajes que permite la comunicación entre el entorno y el propio robot.

- **Mensaje StepMultiTarget**
  - **Descripción**: contiene una lista de pasos.
  - **Contenido**:
    - **Header header**: la cabecera que contiene campos de metadatos comunes, como son el timestamp y un identificador de paquete.
    - **humanoid_step_msgs/StepTarget[ ] footsteps**: una lista de pasos.
- **Mensaje FootGaitDescription**
  - **Descripción**: contiene los valores mínimos y máximos permitidos para cada una de las variables que conforman la acción.

- **Contenido**:
  - **Header header**: la cabecera que contiene campos de metadatos comunes, como son el timestamp y un identificador de paquete.
  - **float64 maxStepX**: el valor máximo permitido para el paso en el eje x.
  - **float64 minStepX**: el valor mínimo permitido para el paso en el eje x.
  - **float64 maxStepY**: el valor máximo permitido para el paso en el eje y.
  - **float64 minStepY**: el valor mínimo permitido para el paso en el eje y.
  - **float64 maxStepTheta**: el valor máximo permitido para el paso respecto a la diferencia angular de los pies.
  - **float64 minStepTheta**: el valor mínimo permitido para el paso respecto a la diferencia angular de los pies.
  - **float64 maxStepPeriod**: el valor máximo permitido para el tiempo de realización del paso.
  - **float64 minStepPeriod**: el valor mínimo permitido para el tiempo de realización del paso.

- **Mensaje StepTarget**

  - **Descripción**: indica el movimiento de una pierna a una determinada posición, respecto a la pierna contraria del movimiento.
  - **Contenido**:
    - **Header header**: la cabecera que contiene campos de metadatos comunes, como son el timestamp y un identificador de paquete.
    - **geometry_msgs/Pose2D pose**: la posición donde se tiene que situar la pierna indicada.
    - **uint8 leg**: la pierna que se desplaza a la posición indicada
    - **uint8 right=0**: constante para indicar el valor de referencia para la pierna derecha.
    - **uint8 left=1**: constante para indicar el valor de referencia para la pierna izquierda.

B.1.4. Paquete *rl_msgs*

Este paquete contiene los mensajes propios para la comunicación entre los nodos propios del software desarrollado.

- **Mensaje RLStateReward**
  - **Descripción**: contiene la información sobre el estado actual del entorno y el refuerzo producido con la última acción realizada.
  - **Contenido**:
    - **Header header**: la cabecera que contiene campos de metadatos comunes, como son el timestamp y un identificador de paquete.
    - **float64[ ] state**: el estado actual del entorno, expresado para un espacio de estado continuo.
    - **float64 reward**: el refuerzo obtenido en la ejecución de la acción anterior.
    - **bool terminal**: informa si el estado se trata de un estado terminal.
- **Mensaje RLExperimentInfo**
  - **Descripción**: contiene la información producida al finalizar un episodio.
  - **Contenido**:
    - **Header header**: la cabecera que contiene campos de metadatos comunes, como son el timestamp y un identificador de paquete.
    - **int32 episode_number**: el número del episodio finalizado.
    - **float64 episode_reward**: el refuerzo acumulado en el episodio.
    - **int32 anyadidos**: el número de elementos añadidos a la base de casos.
    - **int32 number_actions**: el número de acciones realizadas en el episodio.

B.2. Servicios

En esta sección se describen los servicios utilizados para la comunicación con el módulo conectado con el robot. Estos servicios pertenecen al paquete *humanoid_step_msgs*.

- **Servicio FootGaitDescriptionService**
  - **Descripción**: representa la petición para recuperar la configuración de los pasos.
  - **Petición**:
    - No incluye ningún mensaje.
  - **Respuesta**:
    - **humanoid_step_msgs/FootGaitDescription description**: la descripción de los pasos máximos y mínimos permitidos.
- **Servicio StateReq**
  - **Descripción**: representa la petición para recuperar el estado del robot.
  - **Petición**:
    - No incluye ningún mensaje.
  - **Respuesta**:
    - **nao_msgs/TorsoOdometry position**: la posición de los pies en un determinado instante.
- **Servicio MultiStepService**
  - **Descripción**: transmite un conjunto de pasos para ser ejecutados en una sola acción de manera secuencial.
  - **Petición**:
    - **humanoid_step_msgs/MultiStepTarget steps**: la secuencia de pasos a ejecutar en el robot.
  - **Respuesta**:
    - **float64 speed**: la velocidad obtenida en la ejecución de la secuencia de pasos.
    - **float64 distance**: la distancia recorrida en la ejecución de la secuencia de pasos.

    - **float64 time**: el tiempo dedicado en realizar el movimiento.

- **Servicio PoseService**
  - **Descripción**: transmite una posición y configuración de pasos a realizar por el robot.
  - **Petición**:
    - **geometry_msgs/Pose2D pose**: la posición de destino del robot, respecto a la actual.
    - **bool config**: establece la configuración utilizada para realizar el proceso de caminar. Si el valor es *true*, se realiza el comportamiento de caminar con los valores máximos de paso, en caso contrario con los valores por defecto.
  - **Respuesta**:
    - **humanoid_step_msgs/StepMultiTarget steps**: la lista de pasos realizados para completar el movimiento indicado.
- **Servicio StandPoseService**
  - **Descripción**: representa una petición para que el robot se ponga en la posición de inicio de episodio (figura 4.2).
  - **Petición**:
    - No incluye ningún mensaje.
  - **Respuesta**:
    - No incluye ningún mensaje.
- **Servicio StepService**
  - **Descripción**: representa la ejecución de un paso, recuperando el estado inicial y final de los pies, la distancia recorrida y la caída del robot.
  - **Petición**:
    - **humanoid_step_msgs/StepTarget step**: el paso a ejecutar en el robot.
  - **Respuesta**:
    - **nao_msgs/TorsoOdometry initialLeg**: la posición de los pies antes de realizar el paso.
    - **nao_msgs/TorsoOdometry initialLeg**: la posición de los pies tras la realización del paso.

- **nao_msgs/TorsoOdometry distance**: la distancia recorrida por la realización del paso.
- **bool fallen**: indica la caída del robot.

Apéndice C

Comunicación detallada de los nodos

En este apéndice se detalla el proceso de comunicación de los nodos para las estructuras mostradas en la sección 3.3.3

En las figuras C.1 y C.2 se muestra detallada el proceso de comunicación de los nodos mediante diagramas de secuencia correspondientes a un episodio del proceso de aprendizaje. En ellos, se representan los nodos (la línea de vida), los mensajes identificados por el *topic* (línea con la flecha abierta), los servicios con el *topic* (línea con la flecha cerrada) y su correspondiente retorno (línea discontinua).

En la tabla C.1 se muestra el tipo de mensaje o servicio correspondientes con los *topics* utilizados en la comunicación. La descripción detallada de cada uno de los tipos de mensajes y servicios se encuentra en el anexo B.

En la figura C.1 se observa la ejecución de servicios que se realiza entre el nodo que conecta con el robot ("/steps") y el nodo que representa al agente y al entorno ("/rl_experiment"). En esta comunicación se observa 3 fases:

1. **Inicialización del episodio**: en esta fase se obtiene una descipción de los límites permitidos para las acciones (llamada 1) y el estado inicial del robot (llamada 2).

2. **Proceso de aprendizaje**: se realiza un proceso iterativo hasta que se cumple la condición de parada, como es recorrer una distancia fijada. Si el algoritmo detecta un estado desconocido, ejecuta el comportamiento base (llamada 3). Para completar este proceso, se manda el paso seleccionado para ser ejecutado en el robot (llamada 4) y se recupera el siguiente estado (el retorno de la llamada 4).

3. **Finalización del episodio**: una vez que el proceso de aprendizaje ha

terminado, se ejecuta todos los pasos seleccionados de manera secuencial (llamada 6), estableciendo con anterioridad la posición inicial del robot (llamada 5).

Topic	Tipo	Mensaje/ Servicio
/rl_agent/rl_action	humanoid_step_msgs::StepTarget	Mensaje
/rl_agent/rl_info_episode	rl_msgs::RLExperimentInfo	Mensaje
/rl_env/fallen	std_msgs::Empty	Mensaje
/rl_env/foot_gait_description	humanoid_step_msgs::FootGaitDescription	Mensaje
/rl_env/rl_state_reward	rl_msgs::RLStateReward	Mensaje
/cmd_pose_srv	humanoid_step_msgs::PoseService	Servicio
/foot_gait_description_srv	humanoid_step_msgs::FootGaitDescriptionService	Servicio
/multistep_srv	humanoid_step_msgs::MultiStepService	Servicio
/stand_pose_srv	humanoid_step_msgs::StandPoseService	Servicio
/state_req	humanoid_step_msgs::StateReqService	Servicio
/step_srv	humanoid_step_msgs::StepService	Servicio

Tabla C.1: *Topics* de la comunicación

Por otro lado, en la figura C.2 se muestra la comunicación entre los nodos, cuando el agente y el entorno se representan por separado. Este proceso de comunicación dispone de las mismas fases que en el caso de 2 nodos. La principal diferencia es la inclusión de comunicación entre los nodos "/rl_agent" y "/rl_env", ya en el caso anterior la comunicación era interna al representarse en un mismo nodo. En esta comunicación, el nodo "/rl_env" remite la descripción de los límites (llamada 2), el estado inicial (llamada 4) y el estado durante el proceso de aprendizaje (llamada 8). En el sentido contrario, el nodo "/rl_agent" remite la acción seleccionada al entorno (llamada 6) y la información del episodio concluido (llamada 9).

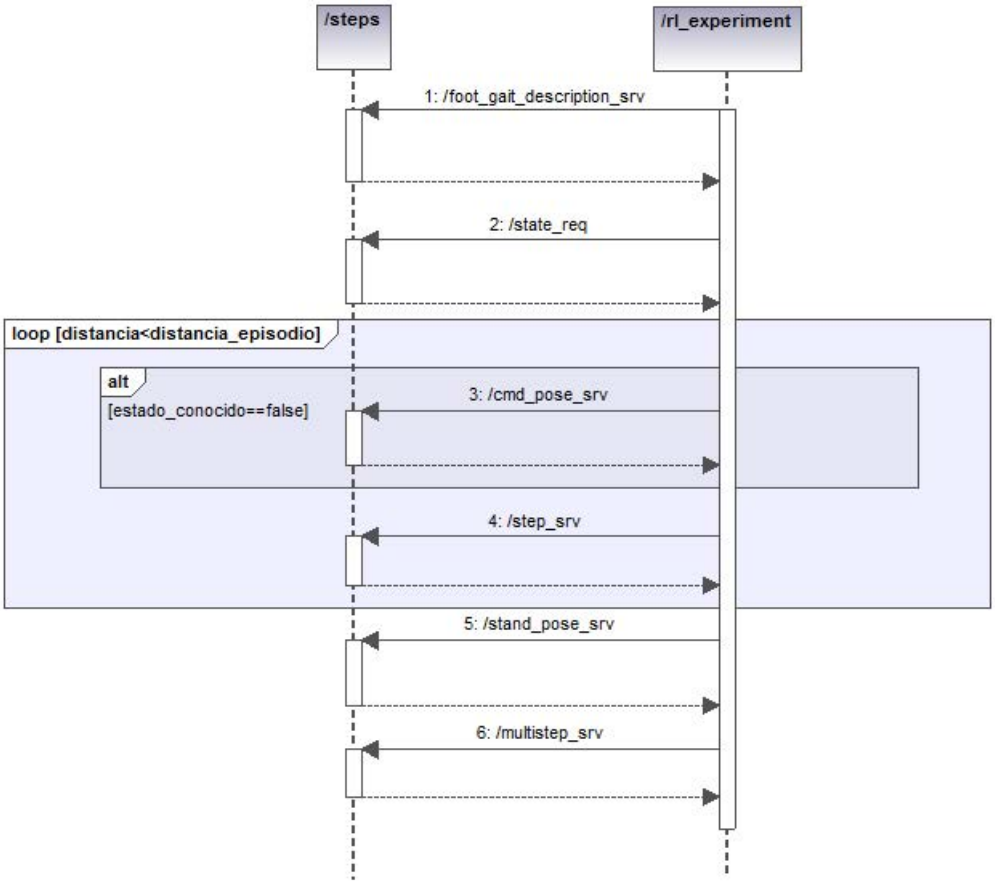


Figura C.1: Proceso de comunicación con 2 nodos

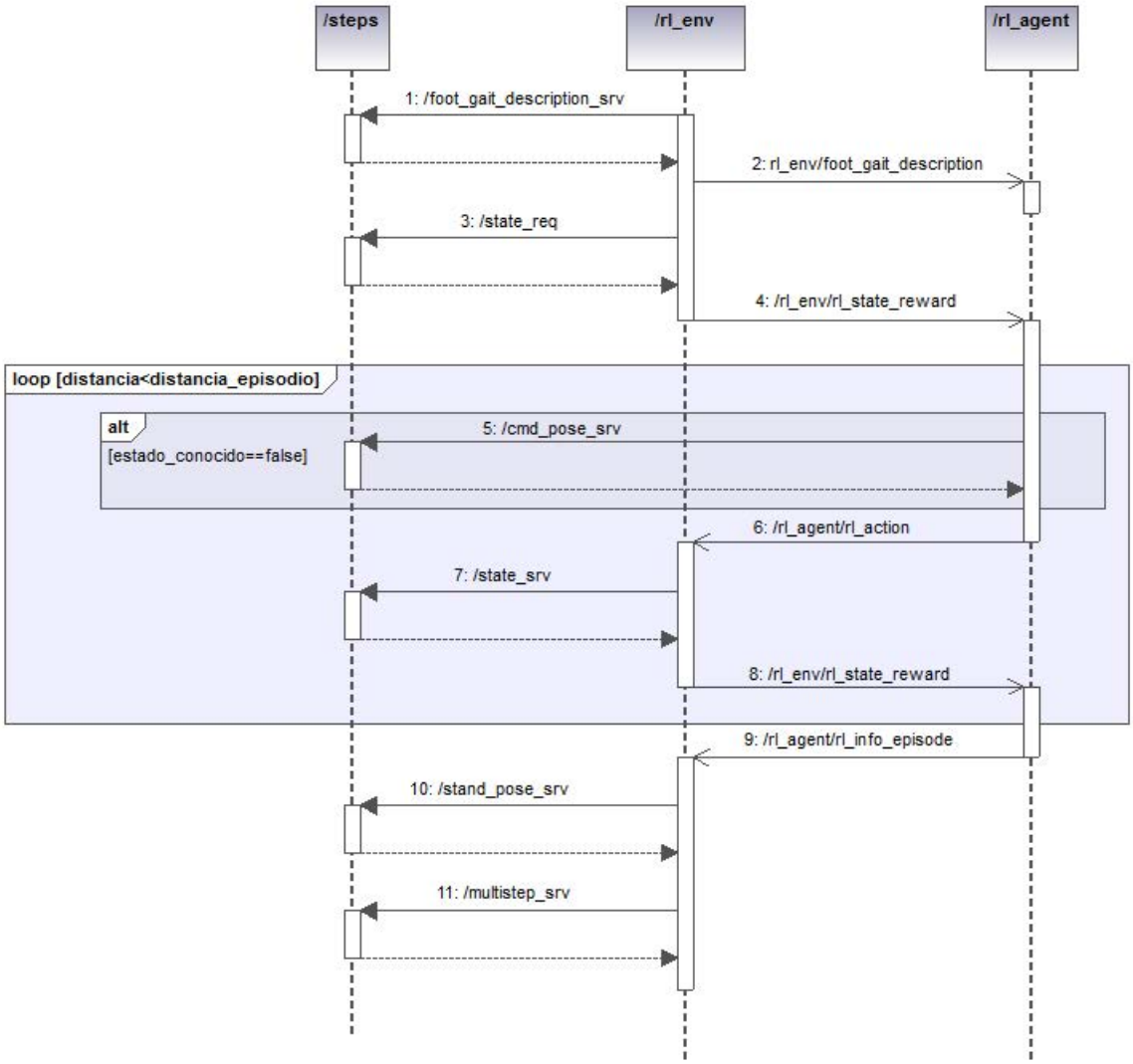


Figura C.2: Proceso de comunicación con 3 nodos

Apéndice D

Instalación y configuración del entorno

En este anexo se detalla los pasos para la instalación y configuración del entorno de ejecución, que está compuesto por el software del robot *NAOqi*, el software *ROS*, el repositorio del robot *NAO* y el software de *RL* desarrollado.

La instalación del entorno se realiza para una distribución del sistema operativo *Ubuntu*, inferior a la versión 11.10.

D.1. Instalación de *NAOqi*

En esta sección se describe la instalación del software del robot llamado *NAOqi*.

La instalación de este software consiste en descomprimir el archivo descargado desde la página de *Aldebaran Robotics* en un determinado directorio. El directorio utilizado es el "/opt/naoqi".

Para ejecutar esta librería es necesario ejecutar el siguiente comando desde un terminal.

./naoqi

D.2. Instalación y configuración de *ROS*

En esta sección se comenta la instalación del software que dispone de los servicios necesarios para la ejecución del software desarrollado.

La distribución de *ROS* utilizada es la *Electric*, la cual está soportada para las distribuciones de *Ubuntu*, 10.04 (*Lucid*), 10.10 (*Maverick*), 11.04 (*Natty*) y 11.10 (*Oneiric*). El sistema *Ubuntu* debe estar instalado en una

máquina virtual (por ejemplo, *Oracle VM VirtualBox*) sobre el sistema operativo *Windows*, en el caso de utilizar simulación.

Los pasos para la instalación de *ROS* en el sistema operativo *Ubuntu*[40] son:

1. Abrir un terminal de comandos.
2. En el caso de que se desconozca la distribución de *Ubuntu* instalada, ejecutar en el terminal el siguiente comando.
   cat /etc/issue
3. Actualizar los repositorios de ROS, para la versión correspondiente de *Ubuntu*.
   - *Ubuntu 10.04 (Lucid)*
     sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu lucid main" > /etc/apt/sources.list.d/ros-latest.list'
   - *Ubuntu 10.10 (Maverick)*
     sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu maverick main" > /etc/apt/sources.list.d/ros-latest.list'
   - *Ubuntu 11.04 (Natty)*
     sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu natty main" > /etc/apt/sources.list.d/ros-latest.list'
   - *Ubuntu 11.10 (Oneiric)*
     sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu oneiric main" > /etc/apt/sources.list.d/ros-latest.list'
4. Recuperar las claves del sitio web que proporciona el software.
   wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
5. Actualizar los repositorios para asegurar que se han incluido.
   sudo apt-get update
6. Instalar el paquete completo de *ROS*.
   sudo apt-get install ros-electric-desktop-full

Una vez terminada la instalación del paquete completo de *ROS*, se procede a la configuración de *ROS*.

1. Incluir la ruta de las variables de entorno de *ROS* en el sistema.
   echo "source /opt/ros/electric/setup.bash" >> `~/.bashrc`
   `. ~/.bashrc`

2. Abrir otro terminal, para disponer de las variables definidas en el paso anterior.

3. Acceder al directorio de instalación de *ROS*.
   cd /opt/ros/electric

4. Modificar el contenido del archivo "setup.sh" por el mostrado a continuación.

   #!/bin/sh

   export ROS_ROOT=`/opt/ros/electric/ros`

   export PATH=`${ROS_ROOT}/bin:${PATH}`

   export LD_LIBRARY_PATH=`/directorio_NAOqi/lib`

   export NAOQI_PATH=`/directorio_NAOqi/lib`

   export ROS_WORKSPACE=`~/ros_workspace`

   export PYTHONPATH=`${ROS_ROOT}/core/roslib/src:${NAOQI_PATH}:${PYTHONPATH}`

   export ROS_PACKAGE_PATH=`/opt/ros/electric/stacks:${ROS_WORKSPACE}`

   `if[!"$ROS_MASTER_URI"];thenexportROS_MASTER_URI=http://localhost:11311;fi`

5. Crear la carpeta "ros_workspace", en el directorio "/home/nombre_usuario" para almacenar los software realizados.

D.3. Instalación y configuración del paquete del robot *NAO*

En esta sección se describe la instalación y configuración del paquete definido en *ROS* para la utilización de los comportamiento del robot *NAO*.

Para realizar la instalación de este repositorio, es necesario el comando *rosinstall*, para su instalación se utiliza el comando:

sudo apt-get install python-rosinstall

Los pasos a realizar para la instalación y configuración del repositorio[41] son:

1. Crear un fichero nombrado "rosinstall.txt" en el directorio "/home", con el siguiente contenido.

   ```
   -svn:

   uri:https://alufr-ros-pkg.googlecode.com/svn/trunk/humanoid_
   stacks/humanoid_msgs

   local-name:stacks/humanoid_msgs

   -svn:

   uri:https://alufr-ros-pkg.googlecode.com/svn/trunk/humanoid_
   stacks/nao_robot

   local-name:stacks/nao_robot

   -svn:

   uri:https://alufr-ros-pkg.googlecode.com/svn/trunk/humanoid_
   stacks/nao_common

   local-name:stacks/nao_common
   ```

2. Ejecutar en un terminal el comando para descargar el repositorio.
   rosinstall . /opt/ros/electric rosinstall.txt

3. Copiar el repositorio descargado al directorio de paquetes de *ROS*.
   sudo cp -r ./stack /opt/ros/electric

4. Instalar el paquete *joystick-drivers*, necesario para la compilación del repositorio.
   sudo apt-get install ros-electric-joystick-drivers

5. Establecer permisos a la carpeta "/opt/ros/electric/stack".
   sudo chown propietario:propierario ./stack -R
   sudo chmod 777 ./stack -R

6. Compilar los paquetes del robot *NAO*.
   rosmake –rosdep-install humanoid_msgs nao_robot nao_common

Tras estos pasos, se dispone del repositorio del robot *NAO* para su utilización.

D.4. Instalación y configuración del paquete de Aprendizaje por Refuerzo

En esta sección se detalla la instalación y configuración del paquete realizado para la aplicación de algoritmos *RL* al comportamiento de caminar del robot *NAO*.

Los pasos para instalar este paquete son:

1. Incluir los archivos proporcionados en la carpeta "/home/nombre_usuario/ros_workspace
2. Instalar las librerías *GSL* (**G**NU **S**cientific **L**ibrary).
   sudo apt-get install libgsl0-dev
3. Compilar los paquetes.
   rosmake reinforcement_learning
4. Aplicar permisos de ejecución al ejecutable.
   sudo chmod 777 `~/ros_workspace/humanoid_step/humanoid_step_launch/scripts/nao_steps.py`

Apéndice E

Simulador *NAOsim*

En este apéndice se detalla la instalación y la interfaz del simulador utilizado para la simulación del robot *NAO*, como es *NAOsim*.

NAOsim es un simulador desarrollado por *Aldebaran Robotics* para lanzar un robot *NAO* simulado en un mundo virtual. Este simulador soporta la mayor parte de los sensores y actuadores del robot *NAO*, como son:

- Motores de posición (sensores y actuadores).
- Sistema de navegación inercial.
- Sensores de contacto (*bumpers*).
- Cámaras.

Este simulador sólo es compatible con *Windows*, y desarrollado hasta la versión 1.12.5 de la librería de *NAOqi*, siendo sustituido por el simulador *Webots* en la versión 1.14.

E.1. Instalación

En esta sección se describe el proceso de instalación del simulador en *Windows*.

El proceso de instalación de este simulador[42] son:

1. Descargar la última versión del software desde el área de desarrolladores de *Aldebaran Robotics*.
2. Doble *click* en el fichero descargado.
3. Seguir los pasos establecidos en el proceso de instalación.

Una vez instalado, se puede crear el mundo virtual donde situar al robot *NAO* simulado.

E.2. Interfaz del simulador

En este apartado se describe la interfaz del simulador *NAOsim*.

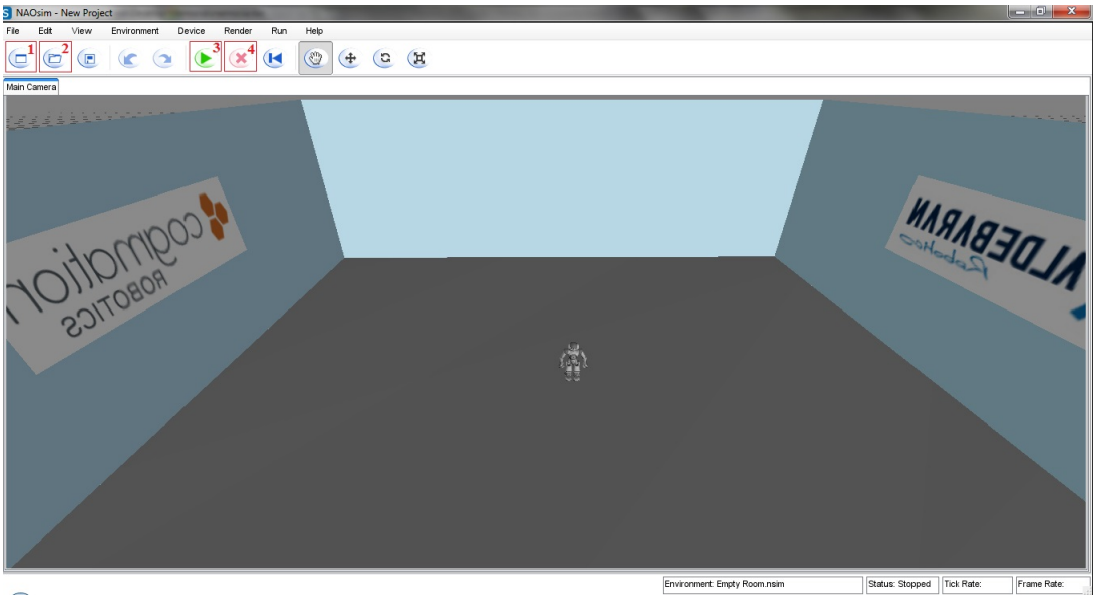


Figura E.1: Interfaz del simulador *NAOsim*

En la figura E.1 se muestra la interfaz del simulador, donde se han seleccionado cuatro botones:

1. Creación de un nuevo mundo virtual.

2. Recuperación de un mundo guardado anteriormente.

3. Inicialización de la simulación, mediante la activación de la librería *NAOqi*.

4. Parada de la simulación.

Figura E.2: Habitación pequeña en el simulador *NAOsim*

En la creación de un nuevo mundo virtual, se dispone de 3 escenarios por defecto, como son:

1. Una habitación vacía, mostrada en la figura E.1
2. Un pequeño apartamento, mostrada en la figura E.2.
3. Un estorno totalmente vacío.

Apéndice F

Ejecución del software

En este apéndice se describe los ejecutables definidos para lanzar los nodos con sus respectivos parámetros y después se muestra los pasos para ejecutar el software realizado, ya sea mediante el uso del simulador *NAOsim* o el robot real.

F.1. Descripción de los *scripts*

Para lanzar los nodos que forman parte del proceso de comunicación, se han definido 3 *scripts*:

- `/home/ros_workspace/humanoid_step/humanoid_step_launch/launch/humanoid_step.launch`: para lanzar el nodo que conecta con el robot (/nao_steps).
- `/home/ros_workspace/reinforcement_learning/rl_experiment/launch/rl_experiment.launch`: para lanzar el nodo que representa al agente y al entorno de manera conjunta (/rl_experiment).
- `/home/ros_workspace/reinforcement_learning/rl_experiment/launch/rl_experiment_2.launch`: para lanzar los nodos del agente (/rl_agent) y del entorno (/rl_env), por separado.

Cada uno de estos scripts, disponen de una serie de parámetros para configurar el proceso. A continuación, se especifica los parámetros de cada uno de estos *scripts*.

F.1.1. *humanoid_step.launch*

Este *script* dispone de dos parámetros que permite la conexión con el robot. Estos parámetros son:

Nombre	Descripción
nao_ip	Dirección ip del robot
nao_port	Puerto de conexión del robot, por defecto el 9559

Tabla F.1: Parámetros para el *script humanoid_step.launch*

F.1.2. *rl_experiment.launch*

Este *script* dispone de los parámetros que permiten configurar a los agentes y a los entornos. En este caso, se muestran los parámetros utilizados para la ejecución de los algoritmos *PI-SRL* y *PR-SRL*.

Nombre	Descripción
type_agent	Tipo del agente a utilizar
type_env	Tipo de entorno a utilizar
dir_exe	Directorio donde se encuentran los archivos de datos
episode	El número de episodios a ejecutar
distance	La distancia a recorrer en cada episodio
theta	El parámetro de la función de riesgo
alpha	El parámetro de actualización del proceso de aprendizaje
thresUpdate	El parámetro de actualización de la base de casos
discountfactor	El parámetro de actualización del proceso de aprendizaje
maxTotalRewardEpisodio	El mejor refuerzo acumulado obtenido
numState	La dimensión del espacio de estado
numAction	La dimensión del espacio de acción
risk	El parámetro de riesgo del algoritmo *PI-SRL* o *PR-SRL*
distanceBaseline	La distancia a recorrer en el comportamiento base
k	El parámetro k del algoritmo *PR-SRL*
configMax	Configuración máxima para el comportamiento base
configDef	Configuración por defecto para el comportamiento base
debug	Activación de la opción de *debug*

Tabla F.2: Parámetros para el *script rl_experiment.launch*

F.1.3. *rl_experiment_2.launch*

Este *script* contiene dos ejecutables, que corresponden al nodo de agentes y al de entornos. Cada uno de ellos dispone de un conjunto de parámetros, los parámetros del agente se muestra en la tabla F.3 y los del entorno en la tabla F.4.

Nombre	Descripción
type_agent	Tipo del agente a utilizar
dir_exe	Directorio donde se encuentran los archivos de datos
episode	El número de episodios a ejecutar
distance	La distancia a recorrer en cada episodio
theta	El parámetro de la función de riesgo
alpha	El parámetro de actualización del proceso de aprendizaje
thresUpdate	El parámetro de actualización de la base de casos
discountfactor	El parámetro de actualización del proceso de aprendizaje
maxTotalRewardEpisodio	El mejor refuerzo acumulado obtenido
numState	La dimensión del espacio de estado
numAction	La dimensión del espacio de acción
risk	El parámetro de riesgo del algoritmo *PI-SRL* o *PR-SRL*
distanceBaseline	La distancia a recorrer en el comportamiento base
k	El parámetro k del algoritmo *PR-SRL*
configMax	Configuración máxima para el comportamiento base
configDef	Configuración por defecto para el comportamiento base
debug	Activación de la opción de *debug*

Tabla F.3: Parámetros para el *script rl_experiment_2.launch* - Agentes

Nombre	Descripción
type_env	Tipo de entorno a utilizar
dir_exe	Directorio donde se encuentran los archivos de datos
episode	El número de episodios a ejecutar
distance	La distancia a recorrer en cada episodio
numState	La dimensión del espacio de estado
numAction	La dimensión del espacio de acción
debug	Activación de la opción de *debug*

Tabla F.4: Parámetros para el *script rl_experiment_2.launch* - Entornos

F.2. Ejecución con simulador *NAOsim*

En esta sección se detalla los pasos para la utilización del software sobre un entorno simulado.

La utilización de un entorno simulado aporta una complejidad, como es la utilización de una maquina virtual con el entorno descrito en el anexo D y el simulador en el sistema *Windows*. Además, el robot se debe encontrar de forma ergida.

Disponiendo de esta configuración, los pasos para la ejecución son:

1. Abrir el simulador *NAOsim* en *Windows*.
2. Cargar un mundo virtual del simulador, mediante el botón 2 de la figura E.1.
3. Consultar la dirección ip asignada al sistema *Windows*.

   Ejecutar > cmd > ipconfig
4. Modificar el parámetro "nao_ip" del *script humanoid_step.launch* (tabla F.1), con la ip obtenida en el paso 3.
5. Activar el simulador NAOsim, pulsando el bóton 3 de la figura E.1.
6. Abrir en la máquina virtual dos terminales.
7. Configurar los parámetros del agente y del entorno, que se muestra en la tabla F.2, F.3 y F.4, pertenecientes a los *scripts*:

   `/home/ros_workspace/reinforcement_learning/rl_experiment/launch/rl_experiment.launch` → Utilizando el nodo "rl_experiment"

   `/home/ros_workspace/reinforcement_learning/rl_experiment/launch/rl_experiment_2.launch` → Utilizando el nodo "rl_agent" y "rl_env"
8. Ejecutar en un terminal el comando para iniciar el nodo "/nao_steps".

   roslaunch humanoid_step_launch humanoid_step_launch.launch
9. Ejecutar en el otro terminal el comando para iniciar el agente y el entorno.

   - Iniciar el nodo "rl_experiment".
     roslaunch rl_experiment rl_experiment.launch
   - Iniciar los nodos "rl_agent" y "rl_env" en dos nodos diferentes.
     roslaunch rl_experiment rl_experiment_2.launch

F.3. Ejecución con robot *NAO* real

En esta sección se detalla los pasos para la utilización del software con el robot *NAO* real.

En este caso, el proceso se realiza desde una versión de escritorio de *Ubuntu*, no necesariamente en una máquina virtual. La versión sobre la que está probado es la 1.12.5 del software del robot.

Para los pasos que se muestran a continuación se asume que el robot se encuentra ergido. Estos pasos son:

1. Conectar el ordenador a la red en la que está conectado el robot *NAO*.
2. Consultar la dirección ip del ordenador, ejecutando en un terminal el siguiente comando.

   ifconfig
3. Modificar el parámetro "nao_ip" del *script humanoid_step.launch* (tabla F.1), con la ip obtenida en el paso 2.
4. Abrir dos terminales.
5. Configurar los parámetros del agente y del entorno, que se muestra en la tabla F.2, F.3 y F.4, pertenecientes a los *scripts*:

   `/home/ros_workspace/reinforcement_learning/rl_experiment/launch/rl_experiment.launch` → Utilizando el nodo "rl_experiment"

   `/home/ros_workspace/reinforcement_learning/rl_experiment/launch/rl_experiment_2.launch` → Utilizando el nodo "rl_agent" y "rl_env"
6. Ejecutar en un terminal el comando para iniciar el nodo "/nao_steps".

   roslaunch humanoid_step_launch humanoid_step_launch.launch
7. Ejecutar en el otro terminal el comando para iniciar el agente y el entorno.

   - Iniciar el nodo "rl_experiment".
     roslaunch rl_experiment rl_experiment.launch
   - Iniciar los nodos "rl_agent" y "rl_env" en dos nodos diferentes.
     roslaunch rl_experiment rl_experiment_2.launch

Apéndice G

Glosario

- **Bumpers**:Dispositivo mecánico que se utiliza para la detección de obstaculos por contacto directo
- **CBR**: Case-Based Reasoning
- **CoG**: Center of Gravity
- **CoP**: Center of Pressure
- **Diagrama de Gantt**: Herramienta gráfica que muestra el tiempo dedicado para diferentes tareas o actividades a lo largo de un tiempo determinado
- **float32**: Tipo de dato, que se corresponde con float en C++ o Java
- **float64**: Tipo de dato, que se corresponde con double en C++ o Java
- **framework**: Estructura conceptual y tecnológica que sirve de base para la organización y desarrollo de software
- **GSL**: GNU Scientific Library
- **LIPM**: Linear Inverted Pendulum Mode
- **MDP**: Markov Decision Process
- **mt**: Unidad de medida de distancia, metros
- **n,ul**: Unidad normalizada, que dispone de valores entre 0 y 1
- **NAOqi**: Librería que proporciona los mecanismos para acceder a las funcionalidades del robot *NAO*

- **NAOsim**: Simulador desarrollado por la empresa *Aldebaran Robotics* para la representación del robot *NAO* en un mundo virtual.
- **PI-SRL**: Policy Improvement thought Safe Reinforcement Learning
- **PR-SRL**: Policy Reuse for Safe Reinforcement Learning
- **rad**: Unidad de medida angular, radianes
- **RL**: Reinforcement Learning
- **ROS**: Robot Operating System
- **seg**: Unidad de medida de tiempo, segundos
- **sonar**: Dispositivo mecánico que utiliza el sonido para el cálculo de la distancia con un objeto
- **spline**: Curva diferenciable definida en porciones mediante polinomios
- **uint8**: Tipo de dato, que se corresponde con el char en C++ o Java
- **ZMP**: Zero Moment Point

Bibliografía

[1] Meriçli, C., Veloso, M. (2010). Biped walk learning through playback and corrective demonstration. In *AAAI 2010: Twenty-Fourth Conference on Artificial Intelligence* (2010). [online] http://www.cs.cmu.edu/~mmv/papers/10aaai-walklearning.pdf

[2] Farchy, A., Barrett, S., MacAlpine, P., Stone. P. (2013). Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proc. of 12th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)* (2013). [online] http://www.cs.utexas.edu/users/pstone/Papers/bib2html-links/AAMAS13-Farchy.pdf

[3] Lee, J., Oh, J-O. (2007). Biped walking pattern generation using reinforcement learning. In *7th IEEE-RAS International Conference on Humanoid Robots* (2007), pp. 416 - 421. [online] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=4813903

[4] Kulk, J., Welsh, J. S. (2011). Evaluation of walk optimisation techniques for the nao robot. In *11th IEEE-RAS International Conference on Humanoid Robots* (2011), Humanoids '11, pp. 306 - 311. [online] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6100827&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6100827

[5] Geibel, P., Wysotzki, F. (2005). Risk-sensitive Reinforcement Learning Applied to Control under Constraints. In *Journal of Artificial Intelligence Research 24* (2005), JAIR '05, pp. 81108. [online] http://www.jair.org/media/1666/live-1666-2420-jair.pdf

[6] García, J., Fernández, F. (2012). Safe Exploration of State and Action Spaces in Reinforcement Learning. In *Journal of Artificial Intelligence Research 45* (2012), JAIR '12, pp. 515-564. [online] http://www.jair.org/media/3761/live-3761-6687-jair.pdf

[7] García, J., Fernández, F. (2013). Probabilistic Policy Reuse for Safe Reinforcement Learning. Unpublished.

[8] Capek, K: "R.U.R. (Robots Universales de Rossum)", 1920.

[9] Asimov, I: "Runaround", In *Astounding Science Fiction* (1942).

[10] Sánchez, F.M., Millán F., Salvador J., Palou J., Rodríguez F., Esquena S., Villavicencio H. (2007). Historia de la robótica: de Arquitas de Tarento al robot Da Vinci (Parte I). En *Actas Urológicas Españolas 31(2)*, pp. 69-76. [online] `http://scielo.isciii.es/pdf/aue/v31n2/original1.pdf`

[11] Sánchez, F.M., Millán F., Salvador J., Palou J., Rodríguez F., Monllau V., Villavicencio H. (2007). Historia de la robótica: de Arquitas de Tarento al robot Da Vinci (Parte II). En *Actas Urológicas Españolas 31(3)*, pp. 185-196. [online] `http://digital.csic.es/handle/10261/12832`

[12] Historia de los autómatas. `http://timerime.com/es/evento/1340763/Cresibio+inventa+un+clepsidra/`. Visitado en Junio 2013.

[13] Historia reciente de la robótica. `http://en.wikipedia.org/wiki/Robotics`. Visitado en Junio 2013.

[14] Robots históricos desde 1920. `http://www.roboticspot.com/especial/historia/his2004b.php`. Visitado en Junio 2013.

[15] Galería de autómatas `http://crosser.byethost18.com/gallery.html`. Visitado en Junio 2013.

[16] Pato de Jacques de Vaucanson `http://contradiccions.wordpress.com/2008/06/11/jacques-de-vaucanson-y-su-automata-comilon/`. Visitado en Junio 2013.

[17] Robot Lunokhod 1 `http://en.wikipedia.org/wiki/Lunokhod_1`. Visitado en Junio 2013.

[18] Robot QRIO `http://es.wikipedia.org/wiki/QRIO`. Visitado en Junio 2013.

[19] Robot ASIMO `http://www.diseno-art.com/encyclopedia/archive/ASIMO.html`. Visitado en Junio 2013.

[20] Especificaciones hardware del *NAO* Robot, `http://www.aldebaran-robotics.com//documentation/family/nao_h25/index_h25.html#nao-h25`. Visitado en Mayo 2013.

[21] Dimensiones del *NAO* Robot, http://www.aldebaran-robotics.com/documentation/family/nao_h25/dimensions_h25_v33.html. Visitado en Mayo 2013.

[22] Esquema de los actuadores del *NAO* Robot, https://code.google.com/p/uhrobocup/downloads/detail?name=nao_h25_hardware_joint_name.jpg&can=2&q=. Visitado en Mayo 2013.

[23] Página oficial de *Aldebaran Robotics*, http://www.aldebaran-robotics.com/en/. Visitado en Mayo 2013.

[24] Software *NAOqi*, http://www.aldebaran-robotics.com/en/Discover-NAO/Key-Features/NAOqi.html. Visitado en Mayo 2013.

[25] Nao Next Gen, http://www.aldebaran-robotics.com/en/Pressroom/Photography/nao.html#3. Visitado en Mayo 2013.

[26] Iniciativa de uso del robot *NAO* para niños autistas, http://www.aldebaran-robotics.com/en/Solutions/For-Autism/The-Ask-NAO-initiative.html. Visitado en Mayo 2013.

[27] Proyecto *Romeo* del robot *NAO*, http://www.aldebaran-robotics.com/en/Projects/romeo.html.Visitado en Mayo 2013. Visitado en Mayo 2013.

[28] Gouaillier, D., Collette, C., Kilner, C. (2010). Omni-directional Closed-loop Walk for NAO. In *Proceedings of 10th IEEE-RAS International Conference on Humanoid Robots*, Humanoids '10, pp. 448-454. [online] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5686291

[29] Xue, F., Chen, X., Liu, J., Nardi, D (2011). Real Time Biped Walking Gait Pattern Generator for a Real Robot In *RoboCup 2011: Robot Soccer World Cup XV*, pp. 210-221. [online] http://www.wrighteagle.org/publication/BipedWalking.pdf

[30] Aaron James Soon Beng Tay: Walking Nao Omnidirectional Bipedal Locomotion. The university of New South Wales, 2009. [online] http://www.cse.unsw.edu.au/~robocup/2009site/reports/TayThesisB.pdf

[31] Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K and Hirukawa, H.(2003). Biped Walking Pattern Generation by using Preview Control of Zero-Moment Point. In *IEEE - International*

Conference on Robotics and Automation 2003, pp. 1620-1626. [online] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1241826&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1241826

[32] Wieber, P.B. (2006). Trajectory free linear model predictive control for stable walking in the presence of strong perturbation. In *IEEE - International Conference on Humanoids* 2006, pp. 137-142. [online] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4115592&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D4115592

[33] Control del mecanismo de caminar del robot *NAO*, http://www.aldebaran-robotics.com/documentation/naoqi/motion/control-walk.html. Visitado en Mayo 2013.

[34] Fernández, F., Borrajo, D. Aprendizaje por Refuerzo (Asignatura de Aprendizaje Automático) (2010). Visitado en Junio 2013.

[35] Paquete Reinforcement Learning en *ROS*, http://www.ros.org/wiki/reinforcement_learning. Visitado en Mayo 2013.

[36] Paquete de los mensajes del *NAO* (nao_msgs) en *ROS*, http://ros.org/wiki/nao_msgs. Visitado en Mayo 2013.

[37] Paquete de mensajes geométricos (geometry_msgs) en *ROS*, http://www.ros.org/wiki/geometry_msgs. Visitado en Mayo 2013.

[38] Definición de árboles KD para *C++*, https://code.google.com/p/kdtree/. Visitado en Febrero 2013.

[39] *ROS*, **R**obot **O**perating **S**ystem, http://www.ros.org/wiki/ROS. Visitado en Febrero 2013.

[40] Instalación de la distribución *ROS Electric* para *Ubuntu*, http://www.ros.org/wiki/electric/Installation/Ubuntu. Visitado en Septiembre 2012.

[41] Instalación del repositorio del robot *NAO* para *ROS*, http://www.ros.org/wiki/nao/Installation. Visitado en Septiembre 2012.

[42] Instalación del simulador *NAOsim*, https://community.aldebaran-robotics.com/doc/1-12/software/naosim/index.html. Visitado en Septiembre 2012.