



Departamento de computación

Facultad de Informática

UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Desarrollo de un módulo de redescipción de modelos de utilidad mediante Deep Learning para robótica cognitiva

Estudiante: Yeray Méndez Romero

Directores: Francisco Javier Bellas Bouza
Alejandro Romero Montero

A Coruña, 13 de febrero de 2019.

A mi familia y amigos.

Agradecimientos

A los investigadores Francisco Javier Bellas Bouza y Alejandro Romero Montero, por guiarme y ser un factor determinante en el desarrollo de este proyecto.

A mi familia, a mis padres y a mi hermano, por ser un apoyo constante en la consecución de esta etapa.

A mis amigos y compañeros de clase, por haberme permitido aprender y crecer con ellos durante estos últimos cuatro años.

Resumen

Este trabajo de investigación se enmarca en el campo de la robótica cognitiva y se centra en el desarrollo e integración de dos módulos software dentro del sistema motivacional MotivEn, desarrollado por el Grupo Integrado de Ingeniería de la UDC para el proyecto europeo DREAM. Por un lado, y dada la importancia que poseen los entornos de simulación a la hora de evaluar sistemas autónomos, se desarrolló un módulo responsable de gestionar el comportamiento robótico y el intercambio de información entre MotivEn y los diversos entornos de simulación. En concreto, se desarrolló un entorno de simulación experimental empleando el simulador 3D Gazebo. Al permitir la realización de experimentos más realistas, se ha proporcionado una herramienta muy útil para analizar y mejorar el sistema motivacional.

Por otro lado, un sistema basado en motivaciones permite establecer qué acciones son las prioritarias para el robot en su estado actual. MotivEn realiza este cometido empleando una entidad denominada Value Function. Dicha entidad, realiza un mapeado entre el estado perceptivo del robot y la recompensa o utilidad de cada posible acción. En este ámbito, se realizó el desarrollo de un sistema de aprendizaje de Value Functions empleando redes neuronales profundas mediante la biblioteca de Deep Learning TensorFlow. En este caso, la principal innovación se basó en la posibilidad de emplear algoritmos de aprendizaje online. Esta clase de algoritmos permiten realizar el aprendizaje de los datos en tiempo real, es decir, mientras el robot desempeña su actividad. Lo que se busca con esta alternativa es analizar un paradigma de aprendizaje no contemplado hasta el momento en el ámbito de MotivEn. Por lo tanto, el segundo módulo desarrollado será el responsable de gestionar las tareas a realizar por este proceso de aprendizaje.

A lo largo de este documento se explicará cómo se han desarrollado los distintos componentes y cómo se ha realizado la integración de los módulos comentados. Para evaluar la correcta integración de los módulos en el sistema, se empleará un experimento robótico estudiado anteriormente por el Grupo Integrado de Ingeniería.

Palabras clave:

- robótica
- cognición
- motivaciones
- deep learning
- ros
- simulador
- gazebo
- aprendizaje online
- value functions

Índice general

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	3
1.3	Organización de la memoria	4
2	Fundamentos teóricos	7
2.1	MotivEn	7
2.1.1	Conceptos generales	8
2.1.2	Formulación de los conceptos utilizados en MotivEn	8
2.1.3	Arquitectura actual de MotivEn	18
2.2	Aprendizaje en línea	22
2.2.1	Aprendizaje automático tradicional vs Aprendizaje online	22
2.2.2	Descenso del gradiente	23
2.2.3	Descenso del gradiente por lotes	24
2.2.4	Descenso del gradiente estocástico (SGD)	25
2.2.5	Descenso del gradiente por mini-lotes	26
2.2.6	Algoritmos de optimización de descenso de gradiente	26
2.2.7	AdaGrad	26
2.2.8	RMSProp	27
2.2.9	Adam	28
3	Antecedentes	31
3.1	Otros sistemas basados en motivaciones	31
3.1.1	Sistemas basados en motivaciones intrínsecas	31
3.1.2	Sistemas motivados intrínseca y extrínsecamente	32
3.2	Otros sistemas basados en Value Functions	35
3.2.1	El aprendizaje por refuerzo en robótica	35
3.2.2	Value Functions en robótica	37

3.2.3	Adaptación de las Value Functions en MotivEn	39
4	Fundamentos tecnológicos	41
4.1	ROS	41
4.1.1	Características de ROS	41
4.1.2	Componentes más relevantes en el proyecto	43
4.1.3	Papel de ROS en este trabajo	46
4.2	TensorFlow	46
4.2.1	Características de TensorFlow	47
4.2.2	Papel de TensorFlow en este trabajo	47
4.3	Gazebo	47
4.3.1	Características principales	48
4.3.2	Integración con ROS	48
4.3.3	Papel de Gazebo en este trabajo	49
5	Metodología	51
5.1	Roles en el proyecto	51
5.2	Estructuración del proyecto	51
5.2.1	Iteración	52
5.3	Gestión del proyecto	52
5.3.1	Estimación	53
5.3.2	Planificación	53
5.3.3	Recursos	53
5.3.4	Gestión de riesgos	54
6	Desarrollo	55
6.1	Aspectos del desarrollo	55
6.1.1	Requisitos funcionales	55
6.1.2	Requisitos no funcionales	55
6.2	Iteración 1: Análisis global de la arquitectura de MotivEn	56
6.2.1	Información básica de la iteración	56
6.2.2	Resumen	56
6.2.3	Análisis de la iteración	58
6.3	Iteración 2: Desarrollo y análisis de la VF	58
6.3.1	Información básica de la iteración	58
6.3.2	Resumen	59
6.3.3	Análisis de la iteración	59
6.4	Iteración 3: Integración del módulo <i>Value Functions Module</i>	60

ÍNDICE GENERAL

6.4.1	Información básica de la iteración	60
6.4.2	Resumen	60
6.4.3	Análisis de la iteración	62
6.5	Iteración 4: Diseño del módulo <i>Simulator Module</i>	62
6.5.1	Información básica de la iteración	62
6.5.2	Resumen	63
6.5.3	Análisis de la iteración	64
6.6	Iteración 5: Diseño del entorno virtual y desarrollo del comportamiento robótico	64
6.6.1	Información básica de la iteración	64
6.6.2	Resumen	65
6.6.3	Análisis de la iteración	69
6.7	Iteración 6: Desarrollo de la configuración del sistema motivacional	70
6.7.1	Información básica de la iteración	70
6.7.2	Resumen	71
6.7.3	Análisis de la iteración	72
7	Experimento y pruebas realizadas	75
7.1	Experimento	75
7.1.1	Explicación del experimento	75
7.1.2	Adaptación del experimento en Gazebo	77
7.2	Pruebas realizadas sobre la VF	78
7.2.1	Validación cruzada	78
7.2.2	Parámetros evaluados	79
7.2.3	<i>10-fold cross validation</i> empleando el algoritmo de descenso del gradiente	80
7.2.4	<i>10-fold cross validation</i> empleando el algoritmo Adam	84
7.3	Experimento realizado en el simulador Gazebo	90
7.3.1	Experimento entrenando la VF a partir de las SURs	90
7.3.2	Experimento entrenando la VF desde cero	94
8	Conclusiones	99
8.1	Trabajo futuro	100
A	Glosario de acrónimos	103
B	Manual de integración	105
B.1	Integración del paquete ROS	105

C Manual de configuración	109
C.1 Configuración de MotivEn	109
C.2 Configuración de los simuladores	110
D Manual de ejecución	113
E Contenido del CD	115
Bibliografía	117

Índice de figuras

2.1	Un ejemplo de un esquema basado en SURs.	15
2.2	Esquema seguido para asignar la utilidad de cada estado sensorial de una traza.	17
2.3	Arquitectura del motor motivacional de MotivEn.	18
2.4	Diagrama de flujo de la arquitectura de MotivEn.	21
2.5	Función de error.	24
2.6	Función de error empleando el SGD.	25
4.1	Componentes de ROS	43
4.2	Comunicación básica entre los nodos de ROS	43
4.3	Comunicación entre los nodos para el ejemplo comentado.	45
4.4	Ejemplo de simulación en Gazebo en el que hay un robot móvil y un objeto.	48
6.1	Arquitectura resultante propuesta.	57
6.2	Tareas realizadas en la Iteración 1.	58
6.3	Tareas realizadas en la Iteración 2.	60
6.4	Diagrama UML del módulo <i>Value Functions Module</i>	61
6.5	Tareas realizadas en la Iteración 3.	62
6.6	Diagrama de clases del módulo <i>Simulator Module</i>	63
6.7	Tareas realizadas en la Iteración 4.	64
6.8	Archivo <i>model.config</i>	66
6.9	Plugin en el archivo <i>model.sdf</i>	66
6.10	Archivo <i>simulator.launch</i>	67
6.11	Diagrama de la comunicación ROS entre el nodo simulator y el nodo que controla las ruedas del robot móvil.	68
6.12	Tareas realizadas en la Iteración 5.	70
6.13	Configuración del fichero <i>Configuration</i>	71
6.14	Configuración del fichero <i>SimulatorConfiguration</i>	72
6.15	Tareas realizadas en la Iteración 6.	73

7.1	Escenario del experimento. En él se pueden ver los robots Baxter y Robobo, el cilindro y la cesta.	76
7.2	Escenario del experimento en el simulador Gazebo. En él se pueden ver al robot Baxter, al robot móvil (color blanco), el cubo (color azul) y la zona objetivo (color rojo).	77
7.3	Datos empleados en la validación cruzada.	81
7.4	Gráficas de los errores y del tiempo de entrenamiento.	83
7.5	Gráficas de la red con el peor error de entrenamiento y sus predicciones. . . .	84
7.6	Gráficas del error y tiempo de entrenamiento empleando el Adam.	89
7.7	Gráficas de la red con el peor error de entrenamiento y sus predicciones empleando el Adam.	89
7.8	Áreas de la mesa en la que el Baxter (región roja) y robot móvil (región azul) desempeñan su actividad.	91
7.9	Secuencia de acciones para que el robot móvil recoga el bloque y se lo acerque al brazo del Baxter.	92
7.10	El Baxter recoge el bloque y lo coloca en la posición objetivo (<i>goal</i>)	93
7.11	Gráficas del error e iteraciones empleadas empleando la VF entrenada a partir de las SURs.	93
7.12	Gráficas del error e iteraciones empleadas en la primera prueba.	95
7.13	Gráficas del error e iteraciones empleadas en la segunda prueba.	96
7.14	Gráficas del error e iteraciones empleadas en la tercera prueba.	97
B.1	Resultado de ejecutar del comando <i>catkin_make</i>	106
B.2	Configuración necesaria para comunicarse con el modelo virtual del Baxter. .	107
B.3	Ejecución del script <i>baxter.sh</i> y del entorno de simulación del experimento. .	107
B.4	Entorno de simulación cargado.	108
C.1	Possible configuración de MotivEn.	110
C.2	Script que permite cargar la configuración para comunicarse con el robot Baxter.	110
C.3	Configuración del script <i>start_motivEN.sh</i>	110
C.4	Possible configuración a emplear en alguno de los simuladores desarrollados. .	111
D.1	Ejecución del script <i>baxter_start.sh</i> y del entorno de simulación del experimento.	113
D.2	Ejecución de MotivEn.	113

Índice de tablas

5.1	Riesgos identificados en el proyecto.	54
7.1	Tabla de las configuraciones y pruebas realizadas con el descenso del gradiente.	82
7.2	Tabla de las configuraciones y pruebas realizadas empleando el Adam.	86
7.3	Tabla de las mejores configuraciones ordenadas a partir de su error medio. . .	87
7.4	Configuraciones candidatas para establecer la VF.	88
7.5	Información de las pruebas realizadas.	93
7.6	Información recopilada en las pruebas realizadas.	98

Capítulo 1

Introducción

1.1 Motivación

La robótica cognitiva es una rama de la robótica que se fundamenta en el uso de modelos cognitivos bioinspirados para lograr que los robots se desarrollen de forma autónoma a partir de su experiencia en el entorno [1].

Dentro de este campo se encuentra el proyecto de investigación Deferred Restructuring of Experience in Autonomous Machines (DREAM). DREAM es un proyecto de la convocatoria de Tecnologías Futuras y Emergentes en el que están involucradas instituciones europeas de cuatro países diferentes, entre las que se encuentra el Grupo Integrado de Ingeniería (GII) de la Universidade da Coruña [2][3]. El proyecto está financiado por el programa de investigación e innovación Horizon 2020 de la Unión Europea.

La principal innovación de este proyecto es la incorporación del sueño y sus procesos dentro de la robótica cognitiva. Esta etapa de sueño será la encargada de reestructurar la información que el robot recopila durante el “día”, desarrollando modelos que permitan su adaptación y desenvolvimiento en cualquier tipo de dominio desconocido y cambiante [2].

Una de las partes vitales para que DREAM logre sus objetivos, es que el robot guíe su comportamiento en función de sus metas a alcanzar, las cuales debe descubrir de forma autónoma. Se establece así un símil con el comportamiento animal, en el que éstos realizan una serie de acciones para satisfacer sus impulsos. Estos impulsos reciben aquí el nombre de motivaciones y permitirán al sistema autónomo determinar en qué tareas concentrar sus recursos limitados [2].

El módulo responsable de esta labor es el sistema motivacional, al que nos referiremos con el nombre de MotivEn. Este módulo gestiona las motivaciones del sistema y dispone de mecanismos que permiten al robot descubrir sus necesidades automáticamente e identificar sus objetivos a partir de ellas [4]. De esta forma, cuando se descubra una nueva meta, MotivEn tratará de obtener un modelo de utilidad que permita al robot obtener una representación del

“camino” hacia el objetivo. Este modelo proporcionará la utilidad esperada para un conjunto de acciones en función del estado actual, de entre las cuales, aquella que posea un valor máximo represente la mejor opción para alcanzar la meta. El Grupo Integrado de Ingeniería (GII) ha desarrollado hasta el momento un par de representaciones para los modelos de utilidad que realizan esta tarea. Uno de ellos son las Separable Utility Regions, a las que nos referiremos como SURs [4][5] y que se explicarán en otra sección de este documento. Este tipo de representación permite alcanzar el objetivo siguiendo la correlación activa de cierto sensor. Aunque no son precisas, proporcionan una alta robustez a cambios en el entorno y sirven como precursores del otro modelo desarrollado, las Value Functions [4]. A diferencia de las anteriores, estas funciones permiten obtener valores de utilidad continuos, y dentro de MotivEn realizan un aprendizaje de los datos recopilados por las SURs mediante redes de neuronas artificiales. A pesar de los múltiples experimentos realizados hasta la fecha en DREAM, las Value Functions no ofrecen una solución válida a un gran conjunto de problemas, fundamentalmente porque las técnicas de aprendizaje utilizadas no son las adecuadas. De ahí surge la necesidad de explorar una nueva alternativa que trate de mejorar el funcionamiento actual de este módulo, basada en Deep Learning, siendo este el objetivo básico de este trabajo de fin de grado.

Otro aspecto de gran relevancia para DREAM es el entorno en el cual se realiza la actividad del robot. MotivEn dispone de un simulador en 2D que por el momento ha permitido obtener los datos necesarios para definir los modelos de utilidad y realizar diferentes pruebas. Sin embargo, se trata de un aspecto que también se pretende mejorar en este trabajo a través de la utilización de un entorno de simulación 3D que permita el uso de modelos de robots reales y, a su vez, disminuir el “reality gap”, es decir, la diferencia que existe entre el entorno real y el simulado. El uso de un modelo en simulación más realista no se podía afrontar hasta el momento en DREAM al no tener técnicas de aprendizaje suficientemente potentes, como Deep Learning.

A partir de lo comentado anteriormente, los objetivos principales de este trabajo serán: integrar dentro de MotivEn un componente software que permita realizar el aprendizaje de modelos de utilidad del tipo Value Functions utilizando Deep Learning con el objetivo de mejorar el rendimiento actual; e incorporar un componente software para la simulación de escenarios 3D, con el fin de obtener una mejor representación de la información y del entorno. Se realizará un experimento basado en un problema que ha sido probado con el sistema motivacional actual, y con el que se analizará el comportamiento de MotivEn tras la integración de los respectivos módulos.

1.2 Objetivos

El objetivo principal de este trabajo es el desarrollo e integración de dos componentes software en el sistema motivacional MotivEn. Uno de ellos se encargará de la construcción de los modelos de utilidad del tipo Value Function mediante técnicas de Deep Learning. El segundo será un simulador 3D que representará el entorno y al robot de forma realista. Ambos se implementarán con el objetivo de suplir las deficiencias mencionadas y mejorar el funcionamiento global del sistema motivacional.

Para lograr este objetivo, se plantean los siguientes subobjetivos:

- **Integración de los componentes software:** Tanto el sistema encargado de la obtención de los modelos de utilidad como el simulador, deben proporcionar la información solicitada por el sistema motivacional. Además, estos componentes deberán ser compatibles con lo ya desarrollado por los investigadores del GII.
- **Análisis de la integración y de los resultados:** Una vez realizada la integración, se realizará un experimento orientado a evaluar el comportamiento del sistema motivacional y buscar posibles mejoras.
- **Mantenibilidad:** Los componentes a desarrollar deberán permitir fácilmente su modificación o sustitución evitando tener que modificar el funcionamiento de MotivEn.

1.3 Organización de la memoria

Para la estructura de esta memoria se optó por dividirla en los siguientes capítulos:

- **Capítulo 1, Introducción:** Se explica el contexto en el que se enmarca el trabajo, así como los problemas y objetivos a lograr.
- **Capítulo 2, Fundamentos teóricos:** Descripción de la arquitectura y elementos que componen el sistema motivacional. También se comentan y analizan los algoritmos más empleados en el campo del aprendizaje online.
- **Capítulo 3, Antecedentes:** Se comentan algunos de los sistemas basados en motivaciones empleados en robótica y el uso de las Value Functions en el aprendizaje por refuerzo.
- **Capítulo 4, Fundamentos tecnológicos:** Descripción y justificación de las herramientas empleadas para la consecución de los objetivos iniciales.
- **Capítulo 5, Metodología:** Se detallan los procesos de ingeniería realizados para la gestión del proyecto.
- **Capítulo 6, Desarrollo:** Descripción de los procedimientos realizados para desarrollar los componentes software a integrar en el sistema.
- **Capítulo 7, Experimento y pruebas realizadas:** Descripción del experimento empleado para evaluar el sistema motivacional final y análisis de las pruebas realizadas y resultados obtenidos.
- **Capítulo 8, Conclusiones:** Evaluación global de los conocimientos aprendidos a partir del trabajo realizado.

También se añaden los siguientes apéndices:

- **Apéndice A, Acrónimos y siglas:** Significado de los acrónimos y siglas que aparecen en esta memoria.
- **Apéndice B, Manual de integración:** Se explican los pasos a seguir para integrar el entorno de simulación.
- **Apéndice C, Manual de configuración:** Descripción de los ficheros de configuración empleados por el sistema motivacional y los parámetros a configurar.
- **Apéndice D, Manual de ejecución:** Instrucciones para ejecutar el sistema motivacional junto con el entorno virtual de Gazebo.

CAPÍTULO 1. INTRODUCCIÓN

- **Bibliografía:** Conjunto de referencias utilizadas.

1.3. Organización de la memoria

Capítulo 2

Fundamentos teóricos

A lo largo de este capítulo se explican los principales conceptos teóricos en los que se ha apoyado este trabajo, fundamentalmente, el sistema motivacional MotivEn y los algoritmos de aprendizaje en línea para Deep Learning.

2.1 MotivEn

MotivEn es el módulo principal que conforma el sistema motivacional de DREAM [2][4]. Antes de continuar con la explicación, se aclara que el robot que emplea este tipo de sistema dispone sensores y actuadores propios

La función principal del sistema motivacional de un robot cognitivo es proporcionar la información necesaria para que el sistema logre una autonomía óptima, permitiendo desenvolver y adaptar su actividad al dominio en el que está inmerso. Para esto, se deben proporcionar herramientas que permitan [2][4][6]:

- Evaluar el estado perceptual del robot.
- Descubrir los distintos objetivos y relacionarlos con los estados y acciones del robot.

Se podría decir que el sistema motivacional debe responder a las siguientes cuestiones:

- *¿En qué estado se encuentra el robot?*
- *¿Cuál es el objetivo de su operación?*
- *¿Qué tiene que hacer el robot para llegar desde el estado actual al estado objetivo?*

Tras comentar las funciones de un sistema motivacional, a continuación se explica el funcionamiento interno de MotivEn describiendo los componentes del sistema desde un punto de vista conceptual.

2.1.1 Conceptos generales

MotivEn constituye el sistema motivacional desarrollado por el Grupo Integrado de Ingeniería [2][4].

Los conceptos que definen, de manera general, el funcionamiento de este sistema son los siguientes:

- Motivaciones.
- Objetivos del robot.
- Modelos de utilidad.

2.1.2 Formulación de los conceptos utilizados en MotivEn

Antes de explicar los mecanismos empleados por MotivEn en su funcionamiento, se introducen los conceptos teóricos que son utilizados por el sistema [4][5]:

- **Estado perceptivo o sensorial (S):** Array con los valores de los sensores del robot. Los valores son números reales ya que MotivEn opera en dominios reales.
- **Acción (A):** Array de valores transmitidos a los actuadores del robot.
- **Impulso:** una medida simplificada de lo cerca, o lejos, que está el sistema de alcanzar un objetivo. Su salida suele ser (aunque no necesariamente) un número real que aumenta con la distancia entre espacio de estados y la meta.
- **Utilidad (u):** el aumento real alcanzado en la satisfacción del impulso en un estado perceptivo $S(t)$. La utilidad está determinada por la interacción del sistema con el entorno y no se conoce a priori.
- **Objetivo innato:** un punto o área en el espacio perceptivo que el diseñador establece en el momento del diseño como un estado deseable para el robot.
- **Motivación intrínseca (I_b):** propiedad innata que guía el comportamiento del robot hacia el descubrimiento de estados sensoriales no visitados, que opera como un proceso exploratorio.
- **Objetivo:** un estado perceptual $S(t)$ en el que el robot obtiene utilidad.
- **Utilidad esperada (e_u):** probabilidad de obtener utilidad a partir de un estado perceptivo dado $S(t)$ modulado por la cantidad de utilidad obtenida. Cuando la utilidad real se obtiene en un estado determinado, los valores de utilidad real y esperada deben ser los mismos.

- **Modelo de utilidad (UM):** función que proporciona la utilidad esperada para cualquier punto en el espacio de estados. Es un modelo interno de la utilidad real utilizada por el sistema para establecer pistas sobre cómo alcanzar los objetivos.
- **Value Function (VF):** función que representa la utilidad esperada exacta (e_u) para cualquier estado perceptivo:

$$e_u(t + 1) = VF(S(t + 1))$$

Será el tipo de modelo de utilidad empleado en MotivEn.

- **Episodio (E):** es una muestra de la respuesta del entorno a las acciones del robot:

$$E = [S(t); A(t); S(t + 1); e_u(t + 1)]$$

- **Buffer de episodios (EB):** memoria en la que se almacenan los episodios “vividos” del robot. Tiene un tamaño predefinido.
- **Trayectoria ($T_{i,j}$):** secuencia de episodios por los que el sistema ha pasado entre t_i y t_j .
- **Traza (t_l):** trayectoria discretizada en función de la longitud de las muestras incluyendo su utilidad.
- **Buffer de trazas (TB):** memoria que almacena un número predefinido de trazas. Esta memoria se utiliza para aprender los modelos de utilidad.

Motivaciones

Las motivaciones, en el marco de la robótica cognitiva, son una de las entidades que más dificultad presentan para el diseño de sistemas de aprendizaje abierto. Esto se debe a la complejidad que requiere establecer estructuras de conocimiento capaces de relacionar las acciones con las necesidades a satisfacer. Estas estructuras posibilitan que el robot pueda aprender y elegir una gran variedad de tareas, además de facilitar su adaptación en entornos que sean cambiantes y/o desconocidos. Dentro de MotivEn se definen dos tipos de motivaciones: intrínsecas y extrínsecas. Las primeras representan impulsos para que el robot explore el espacio de estados. Las extrínsecas están ligadas a la mejora en la consecución de una meta, es decir, explotar dicho espacio [4]. Inicialmente, el robot solo es consciente de sus motivaciones intrínsecas, las cuales son especificadas de alguna manera por el diseñador del sistema. Estos impulsos lo incitan a explorar el entorno en busca de objetivos que aprender y generar sus

motivaciones extrínsecas. Un ejemplo de esto sería el siguiente: el sistema tiene un sensor que mide la temperatura ambiente, y estar en una zona de calor le proporciona satisfacción (motivación extrínseca que el robot desconoce inicialmente). Entre sus motivaciones intrínsecas se encuentra la necesidad de conocer zonas de distinta temperatura. Estos impulsos provocan que el robot explore su entorno hasta encontrar de forma semi-aleatoria el punto de calor. Una vez encontrado, su objetivo será encontrar su zona de confort de manera óptima. La idea es que estos tipos de motivaciones unidos a las diversas características del dominio, permitan lograr que el sistema sea autónomo. En MotivEn, las motivaciones intrínsecas se relacionan con el movimiento aleatorio sobre el entorno y con el descubrimiento de eventos novedosos y desconocidos, es decir, ayudan al descubrimiento de objetivos. Las motivaciones extrínsecas en MotivEn deben ser descubiertas de forma autónoma, y proporcionan al robot una recompensa en los casos en los que alcance un objetivo. Con esta clase de motivaciones se intenta que el robot logre llegar a su objetivo el mayor número de veces de forma autónoma [4][6].

Objetivos del robot

Como dijo el ex-atleta estadounidense, James Ryun: "La motivación nos impulsa a comenzar y el hábito nos permite continuar". De una manera similar, el sistema inicia su actividad guiado por motivaciones intrínsecas con el propósito de alcanzar su meta guiado por motivaciones extrínsecas. La meta u objetivo de cualquier acción, se puede decir que, desde el punto de vista humano, es satisfacer o mejorar una necesidad o aptitud. Esta afirmación, dentro del marco de la robótica, se define como el punto dentro del espacio de estados, al que el robot quiere llegar, es decir, el estado sensorial en que desea estar. El espacio de estados está representado por el entorno y la configuración actual del robot. De este hecho se puede inferir que determinar los puntos objetivos no es una tarea trivial, ya que dependen tanto del entorno, del cual no se tiene porqué tener información concreta, como del estado actual del robot.

Esto obliga al sistema motivacional a definir estructuras que permitan al robot explorar el espacio de estados con el fin de establecer sus objetivos. Tras descubrir su meta, el robot debe encontrar una ruta que lo guíe hacia su consecución. La ruta entre el estado sensorial actual y estado ideal puede definirse como el conjunto de puntos (estados) por los que debe pasar el robot para llegar al estado final, lo que también es equivalente a decir que el camino óptimo es el conjunto de acciones que lo conducen al objetivo. A partir de ambas definiciones se deduce que es necesario desarrollar un mecanismo que permita estimar el valor de cada estado posible en función del actual, lo que es de nuevo equivalente a evaluar la recompensa esperada de cada posible acción a partir de un estado dado. Dentro del sistema motivacional esto da lugar a los modelos de utilidad.

Modelos de utilidad

Una de las razones por las que resulta altamente complejo definir modelos de utilidad precisos se debe a que los dominios de actuación son continuos y de varias dimensiones. Por este motivo, los modelos implementados en diversas investigaciones se han basado en la consecución de objetivos locales centrados en áreas reducidas del espacio de estados. La jerarquía de subobjetivos es uno de los enfoques más populares. Esta idea propone que, una vez se descubran una serie de subobjetivos, el robot pueda utilizarlos y desplazarse entre ellos hasta llegar a su objetivo final. Esta aproximación permitiría establecer acciones básicas y su reutilización en objetivos diferentes, simplificando así los modelos de utilidad [4]. Basándose en esta idea, el GII propone una metodología que se basa en el establecimiento progresivo de funciones de utilidad mediante el empleo de modelos más robustos y menos precisos en etapas iniciales que permitan construir funciones de utilidad más precisas conforme aumenta el conocimiento del sistema. Para ello, han diseñado el modelo de utilidad Separable Utility Region (SUR) [5]. Este tipo de funciones permiten identificar áreas en el mapa de estados en las que hay una correlación entre alguno de los sensores y la dirección en la que el sistema debe desplazarse si quiere alcanzar su objetivo. Esta formulación, a pesar de no obtener un valor de utilidad demasiado preciso, ofrece una mayor robustez al ruido percibido por los sensores y a las variaciones en entornos complejos, permitiendo establecer regiones de certeza y allanar el camino para establecer modelos más precisos como son las Value Functions [4].

Un factor relevante para el desarrollo de modelos de utilidad robustos es la cantidad de estados sensoriales visitados. En las etapas iniciales solo hay un pequeño número de trayectorias disponibles, por lo que la fiabilidad de los modelos de utilidad es baja en la mayor parte del espacio de estados, ya que éste aún no se ha explorado. Entonces, para que los modelos de utilidad puedan proporcionar cierto valor de certidumbre se ha definido una función de certeza, la cual asocia a cada estado sensorial un valor de certeza de llegar al objetivo. Esta función se comporta como un mapa de densidad basado en los estados visitados que se almacenan como trazas en el EB. Dicha función calcula la certeza de un punto en función de su distancia a los estados que realmente se han explorado y que se obtienen del conjunto de trazas del sistema [4][5]. Estas funciones se denominan áreas o mapas de certeza y permiten establecer una jerarquía de subobjetivos que posibilite reutilizar diferentes SURs para intentar alcanzar el estado meta [5]. De una manera similar, también permitirán a MotivEn seleccionar un modelo de utilidad (SUR o VF), lo que permitiría realizar un encadenamiento de diferentes funciones de utilidad y crear una especie de jerarquía de modelos de utilidad [4][5].

A continuación se explica la formulación matemática necesaria para definir los mapas de certeza.

Áreas o mapas de certeza

Para crear los mapas de certeza, el sistema maneja tres tipos de trazas [5]:

- Trazas exitosas (*trazas-s*): creadas cuando el robot, estando bajo la influencia de la motivación extrínseca, alcanza el objetivo dentro de un área de certeza. Tienen una mayor influencia sobre el valor de certeza.
- Trazas fallidas (*trazas-f*): creadas cuando el robot, estando bajo la influencia de motivación extrínseca, no alcanza el objetivo dentro de un área de certeza. Tienen una influencia negativa para el área de certeza.
- Trazas débiles (*trazas-w*): creadas cuando el robot alcanza el objetivo pero no estaba bajo la influencia de la motivación extrínseca. Tienen una influencia positiva menor que el de las *trazas-s*.

Un mapa de certeza se puede ver como un conjunto de regiones que se van ampliando o reduciendo en función del tiempo y de cada tipo de traza. La adición de *trazas-s* amplía el área de valores de alta certeza, mientras que la adición de *trazas-f* lo reduce. Inicialmente se parte un área lo más amplia posible que cubre la mayor parte del espacio de estados. Esto tiene sentido, ya que a priori se desconocen los distintos objetivos y las diferentes trayectorias para alcanzarlos, por lo tanto resulta lógico creer que se puede alcanzar la meta desde cualquier punto del espacio. Este mapa se ajusta gradualmente en regiones que se correlacionan con los estados sensoriales explorados. El siguiente modelo matemático define la implementación utilizada para la creación de los mapas de certeza. En primer lugar, se proporcionan las siguientes definiciones básicas:

- $T \equiv t_1, t_2, t_3, \dots$, es el conjunto de *trazas-s* utilizadas para definir un mapa de certeza. Cada traza está definida por n componentes (ya que el robot posee n sensores), siendo igual a $t_j \equiv c_1^j, c_2^j, c_3^j, \dots, c_n^j$. Lo mismo ocurre con las trazas fallidas y débiles.
- T_{ord}^m es un vector que contiene todos los componentes de todas las trazas ordenadas ascendenteamente.
- $D^m = \max_{int}(T_{ord}^m)$. Representa la máxima distancia entre valores consecutivos de T_{ord}^m .
- N_t es el número de trazas efectivas disponibles. Se calcula sumando el número de *trazas-s* (N_{st}) con el número de *trazas-w* (N_{wt}) y ponderándolo con el coeficiente ($c_f < 1$).

$$N_T = N_{st} + c_f \cdot N_{wt}.$$
- N_{FT} es el número de *trazas-f* almacenadas en memoria.

- L_m^{sup} y L_m^{inf} son los límites superior e inferior del valor del sensor m^{th} .
- $Dr^m = \max \min_j(|c_m^j - L_m^{sup}|), \min_j(|c_m^j - L_m^{inf}|)$ es el valor máximo de las distancias mínimas a los límites del m^{th} sensor.
- $p = p_1, p_2, p_3, \dots, p_n$, es el punto del espacio de estados para el cual se debe calcular la certeza.
- $h_m^j = |p_m - c_m^j|$ es la distancia en la dimensión m^{th} entre el punto de traza j^{th} y un punto p . La definición de las áreas de certeza se basará en la distancia a todos los puntos de las trazas disponibles. Para delimitar esas regiones se define una distancia umbral para cada dimensión, H_{lim}^m :

$$H_{lim}^m \begin{cases} \frac{D^m + (D_r^m - D^m) \cdot K^{N_T - 1}}{2}, & Dr^m > D^m, \\ \frac{D^m}{2}, & \text{sino} \end{cases}$$

donde $K = 0.05^{\frac{1}{N_{T5\%}-1}}$ y $N_{T5\%}$ es el número de trazas (ya sea N_{FT} o N_T) para reducir H_{lim}^m a $1.05 \cdot D^m$ ($N_{T5\%} = 4$ por defecto). Basándose en este límite, la distancia efectiva (la distancia finalmente utilizada para calcular las áreas de certeza) en la dimensión m^{th} entre el punto de traza $j-th$ y cualquier punto p , hn_m^j se calcula utilizando la siguiente expresión:

$$hn_m^j \begin{cases} h_{m'}^j, & h_m^j < H_{lim}^m, \\ H_{lim}^m + (h_m^j - H_{lim}^m) \cdot M, & \text{sino} \end{cases}$$

siendo M , el coeficiente de aumento que multiplicará el valor de la distancia que excede H_{lim}^m para incrementarlo más rápido y anular así su influencia.

Finalmente, el peso, W_J de un punto t_j de las trazas-s se calcula para un punto p de la siguiente forma:

$$W_j = \max(0.1 - d_{jnorm}^{cert})(d_{jnorm} + \frac{1}{N_{sp}})$$

Esta ecuación modela un peso que crece exponencialmente a medida que la distancia al punto de las trazas más cercano se reduce a un máximo que es igual a N_{sp} , por lo que N_{sp} representa el número de puntos de traza opuestos que serán necesarios para eliminar completamente el efecto de un punto de traza previamente almacenado. Siendo d_{jnorm}^{cert} y d_{jnorm} las distancias euclidianas basadas en la distancia h_j^m y en la distancia efectiva hn_j^m normalizadas de acuerdo con los límites de las dimensiones.

Para los puntos (W_j^*) de las trazas-w y de las trazas-f (Z_j), los pesos se calculan de la misma forma. Finalmente, el valor de certeza para un punto p es proporcionado por la combinación de los pesos de las trazas-s (W_J), las trazas-w (W_j^*) y las trazas-f (Z_j)

$$C = \text{sign}(A) \cdot |\tanh A|^{0.1},$$

donde

$$A = c_p \cdot \sum_T (W_j + c_w \cdot W_j^* - c_a \cdot Z_j)$$

siendo c_a y c_w los factores de peso para las *trazas-f* y las *trazas-w* y c_p un coeficiente de amplificación para todo el valor de certeza.

Separable Utility Regions

La idea que el GII ha propuesto con las SURs se basa en que el proceso o ruta definido para alcanzar un objetivo puede describirse como un encadenamiento de acciones sobre el espacio de estados, las cuales pueden aumentar o disminuir el valor devuelto por un sensor. Este tipo de modelos proporcionan caminos hacia la meta empleando las correlaciones de los sensores del robot. De esta forma, se pueden generar diferentes regiones que se activan en función de cada sensor, creando una jerarquía de correlaciones que guían al sistema autónomo hacia su objetivo. Por lo tanto, es necesario determinar la tendencia de cada sensor en cada instante de tiempo, seleccionar la tendencia más fuerte (tendencia del sensor activo) correlacionándola con la acción deseada en el estado sensorial, y definir a partir de dicha correlación un área de certeza. El área de certeza proporciona un valor de certidumbre indicando si para ese estado es aplicable dicha correlación. Si la correlación es aplicable, la SUR proporciona un valor de utilidad esperada que aumenta o disminuye en función del sensor seleccionado, es decir, en función de la distancia al objetivo que mida el respectivo sensor. Bajo este enfoque, el robot puede estar en tres situaciones en el instante de seleccionar una posible acción [5]:

- Si el robot ha seleccionado previamente una tendencia del sensor (una tendencia del sensor está activa), la evaluación del estado usa dicha tendencia.
- Si el robot no ha seleccionado una tendencia del sensor anteriormente (no hay una tendencia activa del sensor), pero hay algún estado anterior que para alguna de las tendencias presenta un valor de certeza mayor que cero, el robot selecciona dicha tendencia y evalúa los demás estados. Estos casos suelen relacionarse con puntos del espacio de estados que aún no se han explorado pero que están próximos a un área conocida.
- Si el robot no ha seleccionado una tendencia del sensor antes (no hay una tendencia activa del sensor) y no hay ningún estado para el que alguna de las tendencias proporciona certidumbre, entonces los demás estados se evalúan utilizando la motivación intrínseca(I_b). Este caso podría darse para un estado que el robot nunca haya visitado

y tampoco se encuentre próximo a algún estado conocido, es decir, está en una zona inexplorada hasta el momento.

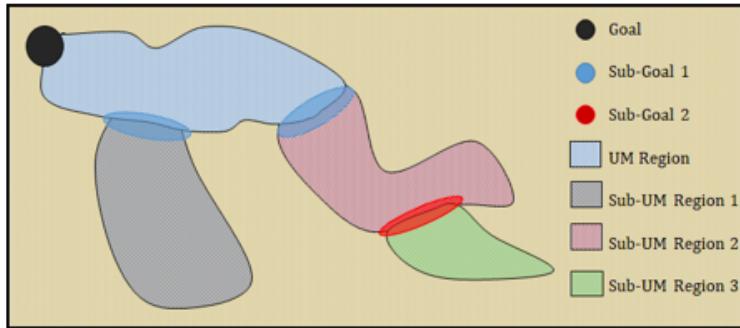


Figura 2.1: Un ejemplo de un esquema basado en SURs.

Las áreas de certeza están limitadas a un subespacio de estados próximos al objetivo, dejando una gran parte del espacio de estados sin asignar al área de certeza de la SUR. Para completar el componente de motivación extrínseca de MotivEn y proporcionar una ruta hacia la meta desde cualquier punto del entorno, es necesario combinar diferentes áreas de certeza con sus respectivos modelos de utilidad. Por esta razón, el concepto de subobjetivo se utiliza en esta nueva representación. Una vez se aprende un modelo de utilidad, una SUR en este caso, se considera un subobjetivo el alcanzar su área de certeza, y las trazas para alcanzarlo se asocian a un nueva función extrínseca. En la Figura 2.1 se muestra la idea comentada anteriormente. Si el robot se encontrase en uno de los estados del área roja, verde o gris, el subobjetivo, en caso de no alcanzar la meta, sería llegar a una SUR que posea un mayor valor de certeza para llegar al objetivo, lo cual concuerda con la idea inicial de la jerarquía SUR. De este modo, los subobjetivos permiten reducir la complejidad del problema y permiten reutilizar el conocimiento al permitir la creación de cadenas de SURs que conducen al objetivo recompensado [4][5].

Value Functions

El concepto de Value Functions ha sido muy utilizado en el contexto del aprendizaje por refuerzo, ya que representan un conjunto de funciones que permiten lograr la política óptima del sistema. Dentro de MotivEn, se modelan como una función que mapea la utilidad esperada para cada uno de los puntos del espacio de estados. Para ello, este tipo de funciones emplean durante su etapa de aprendizaje las trazas generadas por el simulador. En la arquitectura de MotivEn, las Value Functions se implementan mediante una red neuronal que recibe como entrada el estado actual y produce como salida la utilidad esperada para ese estado.

tura actual de MotivEn, dichos modelos de utilidad emplean de forma progresiva las distintas trazas modeladas a partir de las SURs. Una vez que se obtiene una representación basada en la jerarquía SUR de un *UM*, las trazas generadas empleando las regiones de utilidad local son menos ambiguas, ya que el robot guía sus acciones en función de las correlaciones de los sensores. Con este enfoque, una VF puede proporcionar un valor preciso de utilidad para los estados contenidos en el mapa de certeza de la SUR y en consecuencia seleccionar la acción que proporcione al sistema la mayor utilidad [4].

Aunque sería posible que el sistema motivacional iniciara su actividad empleando las Value Functions, ignorar la información de las SURs, y emplear una función de utilidad que logre satisfacer el objetivo del sistema, esto no es lo más recomendado cuando se está aprendiendo una tarea desde cero. El motivo es que MotivEn desconoce cuales son los objetivos y cómo alcanzarlos, por lo que tendrá que descubrirlos y asociarlos a puntos en su espacio de estados durante la marcha, estableciendo así la respectiva VF. Debido a las características del aprendizaje abierto y a las del entorno (que puede ser desconocido y cambiante), los diferentes elementos del escenario pueden ubicarse inicialmente en cualquier lugar y mostrar diferentes configuraciones durante la actividad del robot, lo que combinado con el método de rastreo de elegibilidad utilizado para asignar la utilidad a cada traza, genera un conjunto de trazas multivalor. Este hecho induce a ambigüedades cuando se intenta aprender una función de utilidad precisa sin poseer previamente una región de certeza, ya que el objetivo puede situarse sobre diferentes puntos del espacio de estados. Por lo tanto, esta alternativa queda descartada inicialmente. Lo que sí se podría analizar es el comportamiento del sistema empleando una VF entrenada de forma offline y con una área de certeza definida [4].

La primera aproximación desarrollada por el GII para el aprendizaje de este tipo de funciones se basa en emplear una ANN Feed Forward con tantas entradas como SURs empleadas y una salida, el valor de utilidad. Las entradas se corresponden con los valores de los sensores de las respectivas SURs. La red es entrenada mediante el algoritmo de descenso del gradiente clásico y emplea las trazas de cada SUR, las cuales son almacenadas en una memoria, y se le asignan un valor de utilidad decreciente siguiendo el esquema clásico utilizado en aprendizaje por refuerzo. En la Figura 2.2 se muestra un ejemplo de como sería asignada la utilidad siguiendo el esquema comentado. Una vez se entrena la VF, ésta sustituye a las SURs empleadas. A partir de ese momento la VF pasa a modelar el comportamiento y las áreas de certezas de las SURs. Cada vez que se reciba una nueva traza, ésta se almacena y se actualiza el mapa de certeza. Las trazas almacenadas se emplean para realizar etapas de reentrenamiento de la VF, ya que con esta aproximación el aprendizaje no se puede realizar de manera online. Además, debido a que es necesario almacenar cada traza nueva para realizar la actualización de la red, se produce un gran coste en memoria. Por estos motivos se propone el enfoque desarrollado en este trabajo, el cual se basa en el concepto de aprendizaje online que se presenta

a continuación. La VF a desarrollar se basa en una ANN con varias capas ocultas que emplea el algoritmo de aprendizaje online Adam. Esto le permite solventar el problema de memoria, ya que este algoritmo es capaz de actualizar los parámetros de la red cada vez que recibe una nueva traza. Además, en lugar de substituir las SURs empleadas durante el entrenamiento, en esta aproximación la VF tendrá su propia área de certeza. Esto permite a MotivEn emplear las SURs o la VF a voluntad en función de la certeza que posea cada modelo [4].

Traces memory	
p_{80}	1.0
p_{79}	0.8
p_{78}	0.6
p_{77}	0.4
p_{76}	0.2
p_{55}	1.0
p_{54}	0.8
p_{53}	0.6
p_{52}	0.4
p_{51}	0.2
p_{10}	1.0
p_9	0.8
p_8	0.6
p_7	0.4
p_6	0.2

Figura 2.2: Esquema seguido para asignar la utilidad de cada estado sensorial de una traza.

2.1.3 Arquitectura actual de MotivEn

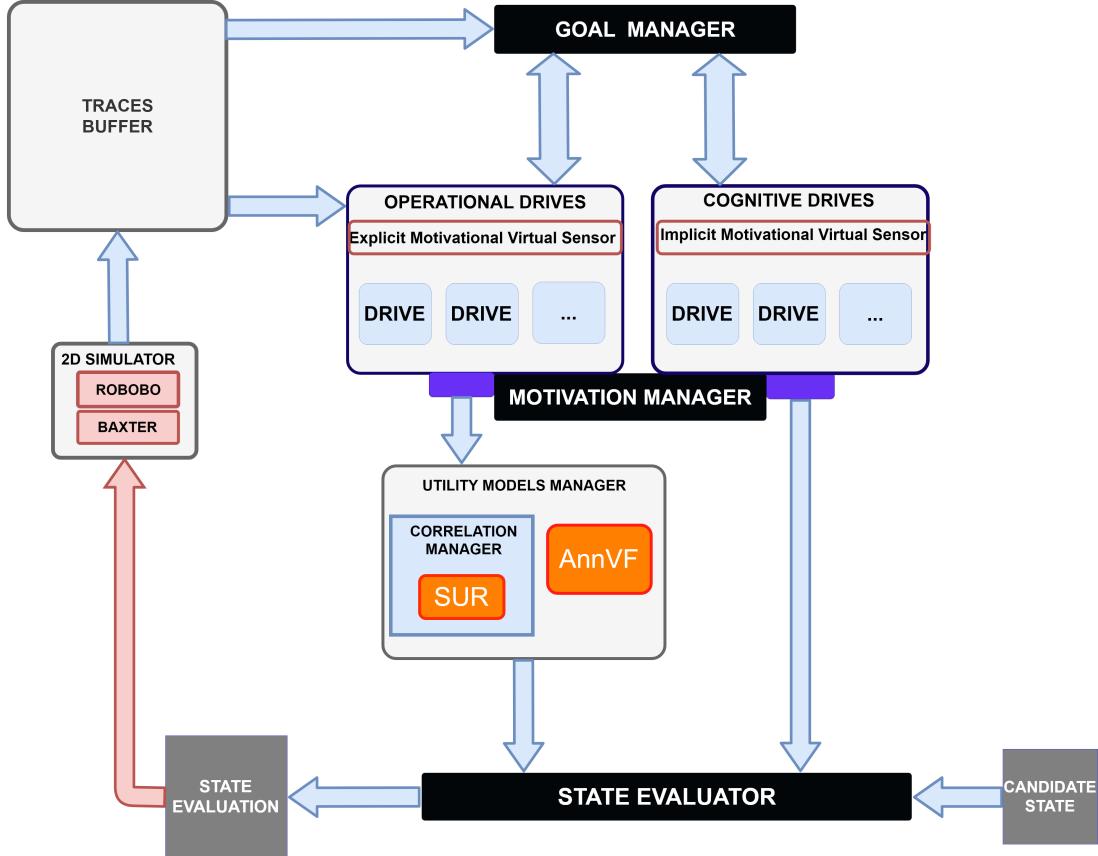


Figura 2.3: Arquitectura del motor motivacional de MotivEn.

El sistema motivacional de un robot cognitivo debe decidir cómo asignar los recursos del robot. Para ello, debe proporcionar medios para evaluar los estados perceptivos, proporcionando una utilidad esperada que le permita decidir la acción más deseable para ese estado. Además, se encarga de descubrir los estados objetivo y asociarlos con los impulsos para que el robot pueda ser realmente autónomo [4].

En la Figura 2.3 se muestra la arquitectura cognitiva de MotivEn. El 2D Simulator representa el entorno de simulación y contiene, en este caso, el modelo virtual del robot Baxter [7] y Robobo [8]. Este simulador proporciona la información del entorno que es utilizada por los componentes del sistema motivacional para proporcionar la acción que mejora la utilidad a largo plazo. La información del simulador se almacena en el *TB* o *Traces Buffer*. Cada una de las trazas generadas, sirve para modelar una “ruta” de utilidad que sea empleada por los modelos en su aprendizaje. Esta tarea es realizada por el *Goal Manager*. Este componente se encarga de reestructurar la información y asignar los modelos de utilidad a sus respectivos

objetivos.

Para que la arquitectura cognitiva pueda determinar la mejor acción para el robot, ésta debe seleccionar qué motivaciones están actualmente activas de acuerdo con un conjunto de reglas administradas por el elemento *Motivation Manager*. Este componente actúa como un filtro motivacional, creando un vector que contiene solo aquellas motivaciones que afectan a la respuesta del robot en un instante de tiempo. Dicho vector se compone de impulsos operativos (*HF-Drive*) e impulsos cognitivos (*E-Drive*). La unidad operativa se asocia al objetivo de la tarea y es controlado por el usuario humano, es decir, la utilidad se proporciona como una retroalimentación humana de manera explícita. En este caso, se asigna empleando el esquema del aprendizaje por refuerzo. Por otro lado, el impulso cognitivo es exploratorio, centrado en explorar el espacio de estados para descubrir el objetivo y mejorar el modelo de utilidad. Por lo tanto, dicho vector de motivaciones está compuesto por motivaciones intrínsecas (*E-Drive*) y extrínsecas (*HF-Drive*). El *Motivation Manager* es el responsable de regular la exploración de los objetivos y la explotación de los mismos con la intención de maximizar la satisfacción del sistema a largo plazo. Para ello:

- Se emplea el *E-Drive* para evaluar los estados sensoriales candidatos en los casos en que el modelo de utilidad no pueda evaluar el estado. Esto puede suceder porque todavía no se ha comenzado a aprender un modelo de utilidad, porque aún no se ha alcanzado el objetivo o porque el estado candidato está fuera del dominio del modelo de utilidad.
- El *HF-Drive* se utiliza para evaluar los estados sensoriales candidatos si el estado está dentro del dominio de un modelo de utilidad.

El vector generado por el *Motivation Manager*, determina el tipo de motivación a emplear en la conducta del robot. En el caso de la motivación extrínseca, éste le indicará al módulo *Utility Models Manager* que proporcione al *State Evaluator* la función de utilidad con mayor certidumbre de llegar a la meta. Si la motivación activa es la intrínseca, el *State Evaluator* selecciona un estado de forma aleatoria. El *Utility Models Manager* gestiona los modelos de utilidad del sistema motivacional. Actualmente éste cuenta con el módulo *Correlations Manager*, el cual es el encargado de la gestión de las SURs, y la AnnVF, la red neuronal desarrollada por el GII. El *State Evaluator* es el módulo encargado de generar los estados candidatos y emplear la motivación indicada por el *Motivation Manager* para evaluar dichos estados y determinar aquel que maximiza la utilidad a largo plazo. Una vez se obtiene el mejor estado candidato, es posible determinar qué acción es la idónea para el robot.

Comentados los componentes principales, el comportamiento del sistema motivacional puede resumirse de la siguiente forma: en las etapas iniciales, cuando aún no se ha descubierto un objetivo, el *Motivation Manager* guía el comportamiento del robot hacia el descubrimiento de estados perceptivos no visitados a través del *E-drive*. Una vez que se alcance el objetivo por

primera vez, el *Goal Manager* proporciona una representación de la ruta hacia la meta. Dicha representación es utilizada por el modelo de utilidad (SUR y/o VF) configurado en MotivEn. En función del aprendizaje realizado por el modelo de utilidad y de su respectiva área de certeza, el *Motivation Manager* selecciona qué motivación debe dominar el comportamiento del robot en cada instante de tiempo. Cuando la respectiva función de utilidad logre un valor de certeza superior a un determinado umbral, el *Motivation Manager* indica que es posible emplear la motivación extrínseca para guiar al robot sobre su espacio de estados empleando el *HF-Drive*, es decir, el *State Evaluator* puede utilizar la función de utilidad para evaluar los posibles estados candidatos. En caso contrario, el robot explora el entorno aleatoriamente bajo la influencia de la motivación intrínseca hasta que se cumpla el caso anterior [4]. El diagrama de flujo de la Figura 2.4 refleja el comportamiento mencionado.

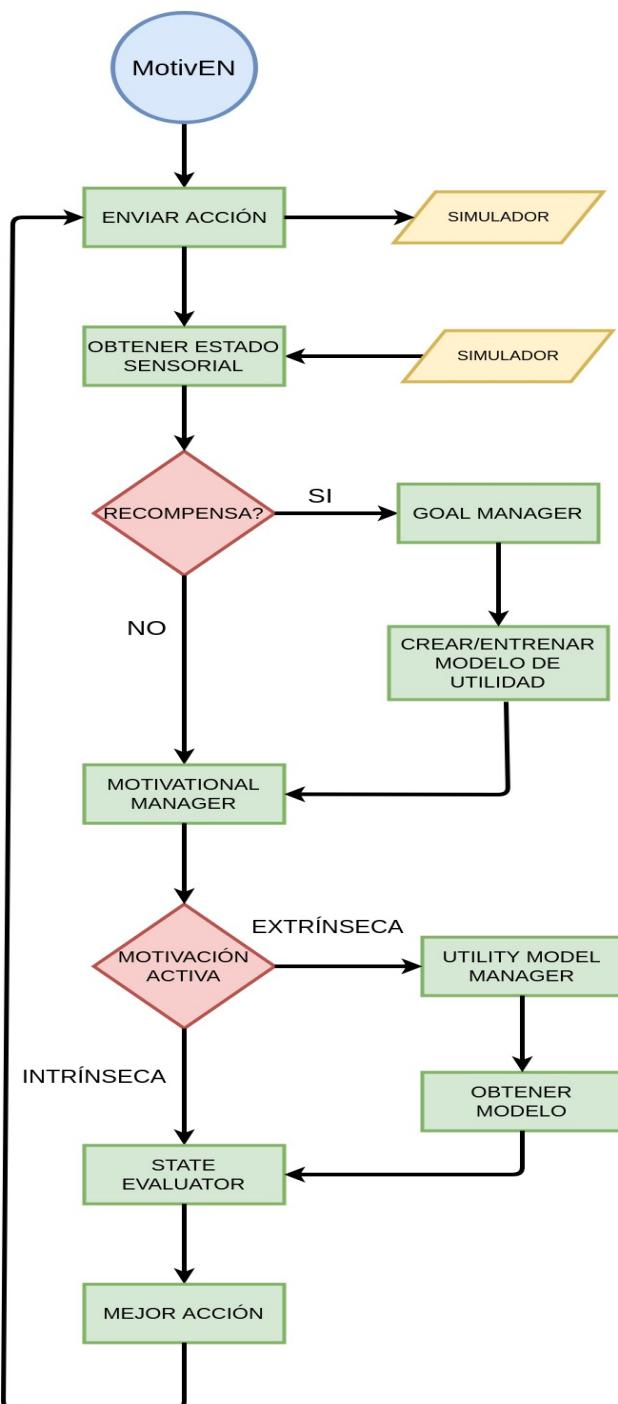


Figura 2.4: Diagrama de flujo de la arquitectura de MotivEn.

2.2 Aprendizaje en línea

El aprendizaje online (Online Learning) es un subcampo del aprendizaje automático que se basa en entrenar modelos cuyos datos son obtenidos secuencialmente a partir del dominio de aplicación. Esta técnica surge de la necesidad de solventar las deficiencias que presentan las metodologías tradicionales de aprendizaje máquina al ser aplicadas a problemas en tiempo real [9].

2.2.1 Aprendizaje automático tradicional vs Aprendizaje online

Los paradigmas tradicionales de aprendizaje automático se basan en el entrenamiento de un modelo a partir de unos datos previamente recopilados. Los datos son presentados al modelo en forma de lotes y éste se entrena empleando un algoritmo de aprendizaje. Una vez se obtiene el modelo que mejor predice la información a partir del conjunto de datos proporcionado, se implementa su funcionamiento dentro del respectivo sistema. En escenarios estáticos en los que la naturaleza de los datos es ya conocida, los paradigmas clásicos representan una solución válida, sin embargo, la aplicación de este tipo aprendizaje en aplicaciones del mundo real en las que los datos no siempre están disponibles y existe la posibilidad de que cambien, presenta una baja eficiencia para aplicaciones a gran escala y en tiempo real, debido a que es necesario repetir el entrenamiento para que el modelo aprenda los nuevos datos [9].

El aprendizaje online intenta resolver este problema realizando un aprendizaje de manera secuencial. El modelo es entrenado con lotes de datos proporcionados de manera secuencial desde el dominio. Esta característica le permite actualizar instantáneamente el modelo para cualquier nueva instancia, a diferencia de los paradigmas de aprendizaje tradicional, logrando así que el aprendizaje automático sea escalable y práctico.

Al igual que los métodos tradicionales de aprendizaje automático (por lotes), las técnicas de aprendizaje en línea se pueden aplicar para resolver gran variedad de tareas en una amplia gama de aplicaciones del mundo real. Algunas de las tareas a resolver son las siguientes:

- Tareas de aprendizaje supervisado como pueden ser la clasificación o la regresión.
- Tareas de aprendizaje no supervisado como un problema de clustering.
- Tareas en sistemas de aprendizaje por refuerzo.

Las técnicas tradicionales de aprendizaje automático suelen emplear redes de neuronas artificiales para resolver problemas de aprendizaje supervisado. Una ANN es un modelo matemático inspirado en el comportamiento biológico de las neuronas y en cómo se organizan formando la estructura del cerebro. Una red neuronal está determinada por su topología (número de capas ocultas y neuronas por cada capa) y un algoritmo de aprendizaje. Los algoritmos

de aprendizaje representan a un conjunto de funciones de optimización mediante las cuales se modifican los diferentes parámetros de la red neuronal (normalmente los bias y pesos) con el objetivo de optimizar una función de error [9].

Volviendo de nuevo a la perspectiva de MotivEn, el aprendizaje de Value Functions se puede clasificar como un problema de regresión dentro del aprendizaje supervisado. Los problemas de regresión se basan en un proceso de aprendizaje mediante el cual se establece una relación entre un grupo de variables (normalmente entre una variable dependiente y una o más variables independientes). Aplicando un análisis regresivo a las Value Functions, el objetivo sería establecer una asociación entre la utilidad esperada y el estado sensorial del robot. Esto permitiría seleccionar las acciones que llevan a estados de mayor utilidad y que conducen al robot hacia su objetivo. En esta clase de problemas, y en general dentro del campo del aprendizaje automático, uno de los algoritmos más utilizados para resolver estos tipos de problemas es el algoritmo del descenso del gradiente [9][10].

2.2.2 Descenso del gradiente

El descenso del gradiente es una forma de minimizar una función objetivo $J(\theta)$ parametrizada por un modelo de parámetros $\theta \in \mathbb{R}^d$ actualizando los parámetros en la dirección opuesta a la pendiente de la función objetivo $\nabla_{\theta}J(\theta)$ respecto a los parámetros. La tasa de aprendizaje η determina el tamaño de los pasos que tomamos para alcanzar el mínimo de la función. En otras palabras, se sigue la dirección de la pendiente creada por la función objetivo hacia abajo hasta que ésta se estabiliza. El descenso del gradiente se basa en el cálculo de la primera derivada sobre la función de pérdida. La derivada representa el valor de la pendiente en la tangente con la función de error, de esta forma se puede determinar cuan lejos se está de alcanzar el mínimo global. Una magnitud elevada en la pendiente indica que es necesario una modificación mayor en los pesos de la red, por el contrario, una pendiente menor indica que se está próximo del mínimo de la función [9][11]. Un ejemplo de esto se puede ver en la Figura 2.5.

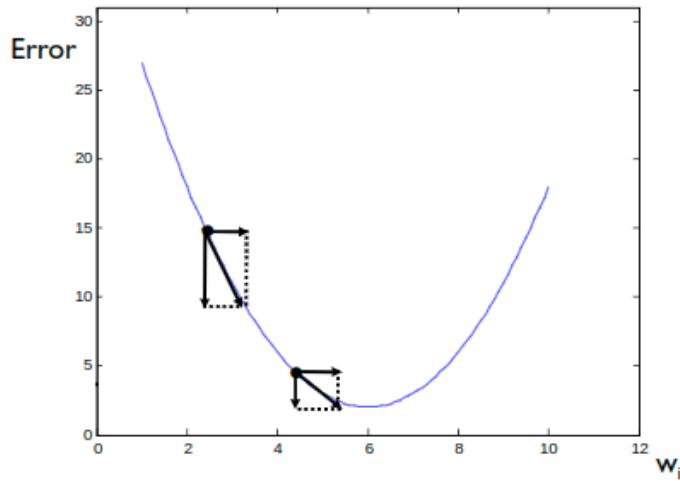


Figura 2.5: Función de error.

Este método de optimización ha sido estudiado de forma exhaustiva dando lugar a las siguientes variantes [9][11]:

- Descenso del gradiente por lotes.
- Descenso del gradiente estocástico.
- Descenso del gradiente utilizando mini-lotes.

La diferencia principal entre estos métodos se basa en la cantidad de datos necesarios para actualizar los parámetros del modelo.

2.2.3 Descenso del gradiente por lotes

El descenso del gradiente por lotes calcula el gradiente de la función de error empleando todo el conjunto de datos de entrenamiento [9]:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta) \quad (2.1)$$

Los parámetros se actualizan siguiendo la dirección de los gradientes, mientras que la tasa de aprendizaje determina cuan grande es la actualización a realizar. El descenso gradiente por lotes converja al mínimo global para superficies de error convexo y a un mínimo local para superficies no convexas. Debido a que calcula el gradiente sobre todo el conjunto de datos para realizar solo una actualización, este método es muy lento y no es aplicable para grandes conjuntos de datos. Además, debido a este motivo no puede emplearse para actualizar el modelo sobre la marcha [9][11].

2.2.4 Descenso del gradiente estocástico (SGD)

Este método, a diferencia del anterior, realiza el cálculo del gradiente y la actualización de los parámetros para cada ejemplo de entrenamiento x con etiqueta y [9][11].

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^i; y^i) \quad (2.2)$$

Podría decirse que el descenso del gradiente por lotes realiza cálculos redundantes para grandes conjuntos de datos, ya que en cada iteración calcula gradientes para ejemplos similares. SGD elimina esta redundancia realizando una actualización por cada ejemplo [9]. Por lo tanto, suele ser mucho más rápido y puede usarse en el aprendizaje online. Además, este algoritmo realiza actualizaciones frecuentes con una alta variación que hace que la función de error fluctúe bruscamente como se puede apreciar en la Figura 2.6.

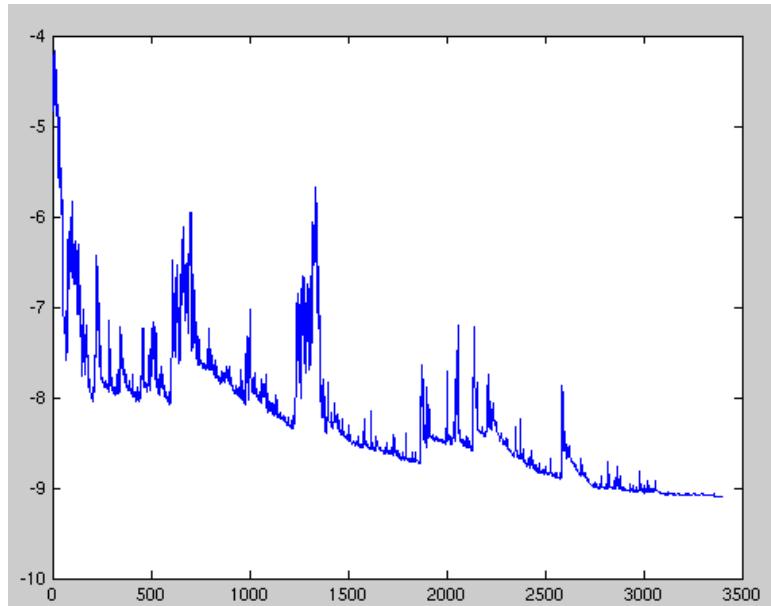


Figura 2.6: Función de error empleando el SGD.

Mientras que el descenso de gradiente por lotes converge al mínimo de la función de error, la variación del SGD, por un lado, le permite saltar a nuevos mínimos locales, y por otro lado, complica la convergencia al mínimo exacto. Sin embargo, se ha demostrado que si la tasa de aprendizaje es suficientemente baja, la SGD muestra el mismo comportamiento de convergencia que el descenso de gradiente de lotes [9].

2.2.5 Descenso del gradiente por mini-lotes

Este algoritmo combina las ventajas de los anteriores y actualiza los parámetros sobre un lote pequeño de n datos de entrenamiento [9][11].

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{i:i+n}; y^{i:i+n}) \quad (2.3)$$

De esta manera, se reduce la varianza de las actualizaciones de parámetros, lo que puede llevar a una convergencia más estable [11].

2.2.6 Algoritmos de optimización de descenso de gradiente

Las variantes comentadas, aunque logran mejorar el comportamiento original del algoritmo de descenso del gradiente, no siempre logran la convergencia. El factor principal de este problema es la tasa de aprendizaje, ya que seleccionar una velocidad de aprendizaje adecuada no es trivial. Si la tasa es demasiado pequeña conduce a una convergencia muy lenta, mientras que una velocidad de aprendizaje demasiado alta puede dificultar la convergencia y hacer que la función de error fluctúe alrededor del mínimo o incluso divergir. Existen esquemas de aprendizaje que permiten ajustar la velocidad de aprendizaje durante el entrenamiento. Sin embargo, estos esquemas y sus respectivos umbrales deben definirse por adelantado, y por lo tanto, no pueden adaptarse a las características de los datos, sobre todo si éstos no están disponibles antes de iniciarse el aprendizaje. Además, la misma tasa de aprendizaje se aplica en todas las actualizaciones de parámetros. Si los datos están dispersos y las funciones tienen frecuencias muy diferentes, no se deberían actualizar todos los parámetros en la misma medida. Otro problema que dificulta la convergencia empleando el SGD se debe a la existencia de puntos en los que una dimensión se inclina hacia arriba y la otra hacia abajo (sillas de montar). Estos puntos generalmente están rodeados por una meseta con el mismo valor de error, lo que hace que sea muy difícil que SGD logre saltar a un mínimo, ya que el gradiente está cerca de cero en todas las dimensiones [9].

Para intentar solucionar estos problemas se han propuestos optimizaciones del SGD empleando tasas de aprendizaje adaptativas, varias de las cuales se comentan a continuación.

2.2.7 AdaGrad

El AdaGrad [9][11] es un algoritmo para la optimización basada en el gradiente estocástico. Este método se basa en modificar la tasa de aprendizaje de los parámetros durante el entrenamiento. Adapta la velocidad de aprendizaje de los parámetros, realizando actualizaciones más grandes para parámetros poco frecuentes y más pequeñas para parámetros frecuentes. Como AdaGrad utiliza una tasa de aprendizaje diferente para cada parámetro θ_i en cada paso

de tiempo t , primero se calcula la actualización de dicho parámetro y posteriormente se vectoriza. Se define $g_{t,i}$ como el gradiente de la función objetivo respecto al parámetro θ_i en el paso de tiempo t .

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}) \quad (2.4)$$

La actualización de SGD para cada parámetro $\theta_{t,i}$ en cada instante t sería:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i} \quad (2.5)$$

En su regla de actualización, AdaGrad modifica la tasa de aprendizaje general η en cada paso de tiempo t para cada parámetro θ_i basado en los gradientes pasados que se han calculado para:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \quad (2.6)$$

$G_t \in \mathbb{R}^{d \times d}$ es una matriz diagonal donde cada elemento i, i de la diagonal es la suma de los cuadrados de los gradientes respecto a θ_i en cada paso de tiempo t , mientras que ϵ es un término de suavizado para evitar la división por cero. Como la matriz G_t contiene la suma de los gradientes pasados, la operación anterior se puede vectorizar realizando una multiplicación \odot de vector-matriz de elementos entre G_t y g_t :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \quad (2.7)$$

La principal ventaja de AdaGrad es que evita tener que seleccionar la tasa de aprendizaje de forma manual. El principal inconveniente surge de las acumulaciones de los gradientes en el denominador. Esto provoca que la tasa de aprendizaje disminuya constantemente lo que puede ralentizar el entrenamiento o la adquisición de conocimiento adicional [9][11].

2.2.8 RMSProp

El algoritmo RMSProp [9][11] trata de solventar el defecto del AdaGrad. En su lugar, utiliza la varianza, la cual le permite descartar los gradientes distantes que pueden no ser muy informativos.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + \gamma g_t^2 \quad (2.8)$$

La varianza ahora se convierte en una aproximación a la estimación local. Esto asegura un progreso óptimo del aprendizaje incluso después de que se hayan realizado una gran cantidad

de iteraciones y actualizaciones.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \quad (2.9)$$

Empíricamente, se ha demostrado que RMSProp es un algoritmo de optimización efectivo para configuraciones en línea y no estacionarias. Ha sido visto como uno de los métodos más populares para la capacitación de modelos de aprendizaje profundo por parte de los profesionales [9].

2.2.9 Adam

Adam [9][11] es otro método que calcula las tasas de aprendizaje de manera adaptativa para cada parámetro. Además de almacenar un promedio de decaimiento exponencial de los gradientes cuadrados pasados (varianza) v_t como AdaGrad y RMSProp, Adam también mantiene una media de los gradientes pasados m_t , similar a las técnicas del Momentum [9]:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \end{aligned} \quad (2.10)$$

m_t y v_t son estimaciones del primer momento (la media) y el segundo momento (la varianza no centrada) de los gradientes respectivamente. Cuando los momentos se inicializan a 0, los bias de la red sufren el mismo comportamiento. Este hecho se produce normalmente en las etapas iniciales del entrenamiento o cuando las tasas de disminución β_1 y β_2 son pequeñas. Esto se solventa realizando una corrección de m_t y v_t .

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \quad (2.11)$$

Las correcciones son empleadas para actualizar los parámetros de manera similar en RMSProp:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t \quad (2.12)$$

Los autores [9] proponen valores predeterminados de 0.9 para β_1 , 0.999 para β_2 y 10^{-8} para ϵ , mostrando empíricamente que Adam funciona bien en la práctica y se compara favorablemente con otros algoritmos adaptativos.

Este algoritmo es el que se ha empleado en los modelos neuronales desarrollados en este trabajo. Como se muestra en la ecuación 2.12, la actualización de la tasa de aprendizaje global se realiza empleando las correcciones de los momentos \hat{v} y \hat{m} . Al emplear la varianza centrada, el Adam permite realizar un aprendizaje óptimo evitando ralentizar excesivamente.

te la velocidad del mismo. Por otro lado, al emplear un promedio de los gradientes pasados, logra realizar actualizaciones que varían en función de la frecuencia de modificación del parámetro. Se puede decir que el Adam posee las ventajas de los dos algoritmos de aprendizaje previamente comentados, y por este motivo fue utilizado en las pruebas llevadas a cabo en este trabajo.

Capítulo 3

Antecedentes

3.1 Otros sistemas basados en motivaciones

Las motivaciones en el ámbito de la Inteligencia Artificial, y en particular, dentro del campo de la robótica, representan un concepto importante para lograr sistemas autónomos capaces de aprender cualquier tarea independientemente del entorno en los que éstos se sitúen.

3.1.1 Sistemas basados en motivaciones intrínsecas

La motivación intrínseca en el ámbito de la robótica se podría definir como el “refuerzo interno” que los agentes utilizan para aprender habilidades que puedan ser combinadas para realizar tareas específicas [6]. Uno de los tipos de motivación intrínseca más popular dentro de la robótica es la motivación para explorar y aprender del entorno. Bajo esta premisa se han desarrollado sistemas como Intelligent Adaptive Curiosity (IAC) [12] o Category-Based Intrinsic Motivations (CBIM) [13]. En este tipo de sistemas, el robot emplea mecanismos para predecir los resultados de sus acciones evaluando al mismo tiempo su capacidad de predicción. De esta forma, el robot es recompensado en el caso de mejorar sus predicciones, motivándolo a realizar acciones para comprender mejor las repercusiones de las mismas con el entorno. Esto refleja que el objetivo de las motivaciones intrínsecas es permitir que el sistema descubra acciones novedosas y obtenga información que le permita mejorar su respuesta con el entorno. Sin embargo, dichas motivaciones no permiten evaluar si una determinada acción es la más idónea para cada circunstancia. Para ello se presenta el siguiente ejemplo: Consideré un robot intrínsecamente motivado colocado sobre una mesa, el robot deambula sobre ella aprendiendo acerca de su entorno y luego se cae de la mesa. El hecho de precipitarse fuera de la mesa, además de causar daños al robot, seguramente no sea uno de los comportamientos deseados, sin embargo, debido a que únicamente las motivaciones intrínsecas guían su comportamiento, el robot se precipitará siempre que ésta represente una acción novedosa para él. En este punto es donde se necesita una especie de supervisor que determine si el robot debe

o no realizar dicha acción dando lugar a la inclusión de motivaciones extrínsecas. Una de las formas comúnmente empleadas, es la de determinar un valor de castigo que contrarreste los impulsos de robot que no le resulten beneficiosos. Las motivaciones intrínsecas se pueden ver como una fase exploratoria que permite la adaptabilidad al entorno y el descubrimiento de tareas, mientras que las extrínsecas posibilitarán mejorar el comportamiento del robot “supervisando” las acciones que éste realice. Este hecho contrasta con la opinión de multitud de expertos en robótica, ya que en el desarrollo de comportamientos robóticos, siempre se trata de minimizar lo máximo posible la supervisión sobre el sistema [6].

3.1.2 Sistemas motivados intrínseca y extrínsecamente

En trabajos como el de Intrinsically Motivated Reinforcement Learning (IMRL) [14] se ha empleado un sistema motivacional que incorpora ambos tipos de motivación. Este sistema se basa en los principios del aprendizaje por refuerzo, pero el robot puede recibir tanto recompensas extrínsecas por lograr ciertos estados en el entorno como recompensas intrínsecas por aprender a predecir ciertos eventos (cambios de iluminación o de sonido). El robot se colocó en un entorno simulado que contenía varios objetos con los que podía interactuar y que producían reacciones de diversa complejidad (algunos objetos siempre harían algo como cambiar la iluminación o hacer ruido, mientras que otros solo harían algo si se cumplían condiciones específicas). En este trabajo, el robot interactuó con ellos y los resultados mostraron que, cuando el robot estaba motivado intrínsecamente para aprender a predecir cambios significativos en el entorno y motivado extrínsecamente para desencadenar con éxito una acción, aprendió cómo desencadenar el comportamiento para ese objeto de forma más eficiente que cuando estaba motivado únicamente de manera extrínseca.

Los resultados con el sistema anterior demuestran cómo la combinación de la motivación intrínseca y extrínseca puede ser útil, pero no necesariamente desde una perspectiva del desarrollo de comportamiento robótico. La naturaleza de las fuentes de motivación extrínseca e intrínseca estaban especificadas por el entorno, ya que la recompensa extrínseca se basaba en la tarea que el robot estaba tratando de realizar, y la recompensa intrínseca dependía de qué cambios se consideraban importantes, algo que podía variar fácilmente. Además, los resultados muestran que la motivación intrínseca ayuda al robot a realizar una tarea específica, pero no muestran claramente cómo la motivación extrínseca afecta la capacidad del robot para aprender sobre el entorno [6].

Huang y Weng crearon un sistema que combina la motivación intrínseca y extrínseca desde la perspectiva del desarrollo robótico. Al igual que el IMRL, el sistema se basa en el aprendizaje por refuerzo, y el robot recibe tanto recompensas intrínsecas por acciones que producen estados nuevos (diferentes de lo que se esperaba) como recompensas extrínsecas. Sin embargo, en este caso, las recompensas intrínsecas se aplican a todas las acciones, no solo

a eventos destacados, y las recompensas extrínsecas se aplican manualmente por un humano (una GUI que el usuario puede emplear para recompensar o castigar al robot por su comportamiento). El sistema se probó con un robot en un entorno simulado en el que podía girar la cabeza a intervalos fijos y mirar el entorno con una cámara. Inicialmente, cuando se colocaba en un entorno estático (de modo que mirar en una dirección dada siempre daría la misma imagen), el robot observaba diferentes ángulos, aprendiendo acerca de cada uno. Pasado un tiempo y para tratar de buscar nueva información, el robot comenzó a realizar acciones aleatorias. El robot se ejecutó en el mismo entorno empleando una recompensa positiva y otra negativa, ambas emitidas por la interfaz. A medida que se realizaba el experimento se descubrió que el robot favorecía la acción recompensada positivamente una vez que no encontraba acciones novedosas. A continuación, el robot se colocó en un entorno en el que la imagen que veía contenía una imagen aleatoria de un juguete cada vez que miraba en una dirección. En este caso, el juguete era muy novedoso, y el robot generalmente prefería mirarlo sin motivación extrínseca. Sin embargo, cuando el robot fue castigado por girar la cabeza en la dirección del juguete y recompensado por alejar la cabeza del juguete, favoreció la acción recompensada aunque el juguete era más novedoso.

La investigación de Huang y Weng muestra cómo la adición de recompensas extrínsecas puede alterar el comportamiento de robots intrínsecamente motivados. El hecho de que las recompensas y los castigos extrínsecos se aplicaran manualmente a los robots puede verse como un nivel de supervisión que no es deseable para el objetivo de la robótica del desarrollo. Además, no se exploró el impacto de las recompensas externas en la capacidad de aprendizaje del robot.

También se propuso un sistema motivacional que combinase motivaciones intrínsecas y extrínsecas mediante el empleo del algoritmo Q-Learning para realizar el aprendizaje del robot [6].

El sistema utilizado es esencialmente una variación del algoritmo estándar Q-Learning. El robot se sitúa en un mundo con distintos estados y acciones, en cual es necesario realizar una acción para cambiar el estado actual. Dicha transición produce una recompensa para el sistema. La recompensa recibida es una suma ponderada de dos componentes distintos: un componente intrínseco, que representa cuánto aprendió el robot de la acción; y un componente extrínseco, que depende del entorno [6].

La motivación intrínseca se modela empleando redes neuronales, las cuales se emplean para predecir el resultado de cada acción. De esta forma, el robot dispone de un red neuronal por cada acción. Estas redes tienen solo dos capas cada una: una capa de entrada que contiene los parámetros del estado actual del robot; y una capa de salida que representa el estado en el que el robot espera estar después de realizar la acción. Cuando el robot realiza una acción, predice el estado resultante, calcula el error de la predicción y ajusta los pesos de la red neuro-

nal. Tras esto compara el error actual con el menor error obtenido para el estado predicho y la acción aplicada. En el caso de que el error actual sea inferior, el robot recibirá una recompensa igual a la a mejora del error obtenido y, en caso contrario, no recibirá recompensa intrínseca.

La recompensa extrínseca del robot está determinada por dos entradas: dolor y placer. Éstas no dependen del estado, ya que no están relacionadas con el entorno, por lo que no pueden ser predichas a partir de los resultados de las acciones. El dolor representa los eventos que el robot debe evitar (como las cosas que lo dañan), mientras que el placer representa las cosas que debe buscar (como recargar una batería baja). La recompensa extrínseca que recibe el robot cuando realiza una acción es igual a la diferencia entre el placer recibido y el dolor.

El experimento realizado se basó en situar al robot en un entorno simulado en el cual se dispusieron diferentes tipos de objetos con los que el robot podía interactuar. Estos objetos podían causar dolor o placer en función del tipo, por lo que el robot debía evitar el peligro y satisfacer sus necesidades o deseos. Los elementos dañinos eran el fuego y las paredes, por lo que el robot debía evitarlos. El objeto que proporcionaba placer era una pila, la cual proporcionaba más o menos cantidad de energía al robot en función de su estado, es decir, el robot recibía más energía (recompensa mayor) cuanto menor fuese batería. Se probaron diferentes configuraciones del sistema motivacional para poder analizar mejor las diferencias entre aplicar un tipo de motivación por separado o aplicar las motivaciones intrínsecas y extrínsecas de forma conjunta. Una prueba se realizó con el robot moviéndose completamente al azar, y cuatro se realizaron empleando una fase de exploración y diferentes tipos de motivación: una empleando solo la motivación intrínseca, otra utilizando solo la motivación extrínseca, una con ambos tipos de motivación ponderados por igual, y uno con motivación intrínseca con cinco veces más peso que la motivación extrínseca. Finalmente se realizaron varias pruebas con cada una de las configuraciones.

Los resultados mostraron que con cualquiera de las configuraciones el robot conseguía aprender su entorno, sin embargo, una vez finalizada la parte de exploración los datos obtenidos variaron. Aunque el error total cuando el robot realizó solamente movimiento aleatorio se mantuvo aproximadamente constante, las configuraciones con motivaciones intrínsecas, en general, parecen ser ligeramente mejores que aquellas con motivaciones extrínsecas, con ambos tipos de motivación y con movimiento aleatorio. Esto indica que la aplicación de este tipo de motivaciones puede ser beneficiosa para que el robot aprenda sobre su entorno [6]. En las configuraciones con motivaciones extrínsecas, se demostró que el robot sufría menos daño que en los casos en los que se lo motivaba intrínsecamente o en los que carecía de motivaciones (aleatorio). Las configuraciones con ambos tipos de motivaciones tuvieron un desempeño ligeramente peor que los que tenían solo la motivación extrínseca, lo que indica que la adición de la motivación intrínseca interfirió muy poco con la capacidad de los robots para evitar el peligro. En todos los casos, los robots recargaron su batería con frecuencia cuando ya estaba

casi llena, sin embargo, los robots motivados extrínsecamente, recargaron más veces su batería cuando su energía era media o baja por lo que fueron capaces de aprender a realizar mejor la tarea con respecto al resto de configuraciones, en las cuales el robot se quedó sin batería en más de la mitad de las veces.

En el caso de MotivEn, el sistema motivacional emplea ambos tipos de motivaciones pero de una forma diferente a los trabajos antes comentados. Ésto permite recalcar que el concepto de motivación, aunque parezca trivial, está abierto a diferentes modelados que permiten adaptar sus beneficios en función del comportamiento robótico a lograr.

3.2 Otros sistemas basados en Value Functions

3.2.1 El aprendizaje por refuerzo en robótica

Una gran variedad de problemas en robótica pueden solucionarse mediante la aplicación del aprendizaje por refuerzo. El aprendizaje por refuerzo (Reinforcement Learning) permite a un agente, robot en este caso, descubrir de forma autónoma un comportamiento óptimo a través de interacciones de prueba y error con su entorno. En lugar de detallar explícitamente la solución al problema, el diseñador del sistema de control (política) proporciona retroalimentación en términos de una función escalar que mide el rendimiento del sistema para cada acción [15].

Esta clase de aprendizaje está relacionado con la teoría del control óptimo clásico. Ambos paradigmas abordan el problema de encontrar una política óptima que optimiza una función objetivo (el coste o la recompensa acumulada), basándose en la noción de un sistema descrito por un conjunto subyacente de estados, acciones y un modelo que describe las transiciones entre los distintos estados. El diseño de una política óptima exige un gran conocimiento de las tareas del sistema, lo que a menudo requiere de gran dificultad. Otro factor que tiene gran influencia, es que los problemas en el ámbito de la robótica están definidos por estados y acciones continuas de gran dimensionalidad. Además, es inadecuado suponer que el estado del robot es predecible y carente de ruido. El sistema de aprendizaje no siempre podrá conocer con seguridad en qué estado se encuentra, ya que incluso estados muy diferentes pueden parecer muy similares. Por lo tanto, el aprendizaje por refuerzo robótico a menudo se modela como parcialmente observado, haciendo necesario establecer mecanismos para estimar el estado verdadero. Aunque resultaría de gran ayuda mantener información tanto del estado como de sus distintas estimaciones, esto resultaría costoso y difícilmente reproducible, lo que provoca que el aprendizaje se realice solo sobre el conjunto de acciones con mayor probabilidad de ejecutarse, desarrollando así sistemas orientados a tener robustez en lugar de un comportamiento variado. Otro desafío, es la generación de funciones de recompensa adecuadas. Las recompensas que guían al sistema de aprendizaje rápidamente hacia la meta son necesarias

para hacer frente al coste de la experiencia del mundo real. Este problema se llama *modelado de recompensa* [16] y representa una contribución manual sustancial. Especificar buenas funciones de recompensa en robótica requiere una buena cantidad de conocimiento del dominio, lo que puede resultar difícil en la práctica, ya que no todos los métodos de aprendizaje por refuerzo son igualmente adecuados dentro de la robótica [15].

En el aprendizaje por refuerzo, el robot trata de maximizar la recompensa acumulada durante su actividad. En este tipo de problemas, el agente y su entorno se suelen modelar en un estado $s \in S$ y pueden realizar acciones $a \in \mathbb{A}$. Los estados contienen la información de la situación actual para predecir estados futuros. Para cada posible acción, el agente recibe una recompensa R , la cual es un valor escalar y se supone que es una función del estado y la observación. El objetivo es encontrar un mapeo de estados a acciones, llamado política π^* , que selecciona acciones en determinados estados para maximizar la recompensa esperada acumulada. La política π es determinista o probabilística. La primera siempre usa la misma acción para un estado dado en la forma $a = \pi(s)$, la última toma una muestra de una distribución sobre acciones cuando llega a un estado, es decir, $a \sim \pi(s, a) = P(a|s)$. El agente necesita descubrir las relaciones entre estados, acciones y recompensas, por lo que es necesario una etapa de exploración. Esta puede integrarse directamente en la política o realizarse por separado y solo como parte del proceso de aprendizaje.

Los enfoques de aprendizaje de refuerzo clásico se basan en el supuesto de que tenemos un *Proceso de Decisión de Markov* [6][17] que consiste en el conjunto de estados S , el conjunto de acciones A , las recompensas R y las probabilidades de transición T que representan el comportamiento del sistema. Las probabilidades de transición $T(s', a, s) = P(s'|s, a)$ describen los efectos de las acciones en el estado. La propiedad de Markov requiere que el siguiente estado s' y la recompensa solo dependan del estado anterior s y de la acción a [6][17], y no de información adicional sobre los estados o acciones pasadas. Los diferentes tipos de funciones de recompensa se usan comúnmente, incluidas las recompensas que dependen solo del estado actual $R = R(s)$, las recompensas que dependen del estado actual y la acción $R = R(s, a)$, y las recompensas que incluyen las transiciones $R = R(s', a, s)$.

Una vez el sistema define la tarea de control que le permita alcanzar el objetivo, el siguiente paso sería mejorar dicho comportamiento. Esta etapa es el proceso de explotación clásico dentro del aprendizaje por refuerzo. Llegado a este punto el sistema debe obtener la política óptima π^* que para cada estado s seleccione la acción óptima a^* con la que obtiene un recompensa mayor a largo plazo, es decir, se trata de un problema de optimización. Para realizar dicha tarea se emplean los métodos de búsqueda de políticas o los métodos basados en funciones valor (Value functions), siendo estos últimos en los que nos centraremos.

3.2.2 Value Functions en robótica

Los métodos basados en funciones valor proponen la definición de la política óptima aproximando una función $V^*(s)$ [6]. Inicialmente se parte de una función de valor $V^\pi(s)$, en la cual no se selecciona la acción óptima a^* , sino que la acción es seleccionada por la política π

$$V^\pi(s) = \sum_a \pi(s, a) \left(R(s, a) - R + \sum_{s'} V^\pi(s') T(s, a, s') \right) \quad (3.1)$$

Esta función aplicada sobre el estado resultante de aplicar una acción se utiliza para definir la función de valor estado-acción $Q^*(s, a)$. De esta forma, el sistema deja de ser consciente únicamente del estado actual y pasa a tener información sobre los efectos de realizar una acción, definiendo la función

$$\begin{aligned} Q^*(s, a) &= R(s, a) - R + \sum_{s'} V^\pi(s') T(s, a, s') \\ &= R(s, a) - R + \sum_{s'} \left(\max_{a'} Q^*(s', a') \right) T(s, a, s') \end{aligned} \quad (3.2)$$

Una política óptima $\pi^*(s)$ se puede definir seleccionando siempre la acción a^* en el estado actual que lleva al estado s' con el valor más alto $V^*(s)$

$$\pi^*(s) = \arg \max_a \left(R(s, a) - R + \sum_{s'} V^*(s') T(s, a, s') \right) \quad (3.3)$$

De esta forma, si se conocen la función de valor óptima $V^*(s')$ y las probabilidades de transición $T(s, a, s')$ para los siguientes estados, determinar la política óptima es sencillo en una configuración con acciones discretas, ya que es posible realizar una búsqueda exhaustiva. Sin embargo en espacios continuos, determinar la acción óptima a^* es un problema de optimización en sí mismo. El uso de la función de valor de acción de estado $Q^*(s, a)$ en lugar de la función de valor $V^*(s)$ evita tener que calcular la suma ponderada sobre los estados sucesores y, por lo tanto, no se requiere ningún conocimiento de la función de transición. A partir de esta base, se han desarrollado una amplia variedad algoritmos de aprendizaje por refuerzo basados en funciones de valor que intentan estimar $V^*(s)$ o $Q^*(s, a)$ y se pueden dividir principalmente en las siguientes clases [6]:

- **Los métodos basados en Programación Dinámica** requieren un modelo de las probabilidades de transición $T(s', a, s)$ y la función de recompensa $R(s, a)$ para calcular la función de valor. No es necesario que el modelo se encuentre predeterminado, sino que también puede aprenderse de los datos de manera incremental. Estos métodos se denominan basados en modelos e incluyen la iteración de políticas y la iteración de valores.

La iteración de políticas alterna entre las fases de evaluación y mejora de políticas. En esta aproximación se parte inicialmente desde una política arbitraria y empleando la fase de evaluación se determina la función de valor para la política actual. Una vez obtenida la función de valor óptima, la mejora de políticas selecciona con avidez la mejor acción en cada estado en función de dicha función. Finalmente, la fase de evaluación de la política y la de mejora se repiten hasta que política ya no cambia.

- **Los métodos de Monte Carlo** realizan un muestreo para estimar la función de valor, reemplazando el paso de evaluación de políticas de los métodos basados en programación dinámica. Además, tales métodos carecen de modelo, por lo que no necesitan una función de transición explícita y van operando sobre la política actual. Las frecuencias de las transiciones y las recompensas se utilizan para modelar estimaciones de la función de valor.
- **Los métodos de Diferencia Temporal** utilizan los errores entre cada iteración para actualizar la función valor. Este error es la diferencia entre la estimación anterior y una nueva estimación de la función de valor, teniendo en cuenta la recompensa recibida en la muestra actual. Estos métodos no utilizan un modelo, al igual que los de Monte Carlo. En esta configuración, la función de valor se debe estimar a partir de las transiciones muestreadas en el PDM.

Aproximaciones de Value Functions

Las distintas aproximaciones a los métodos basados en funciones de valor han permitido proporcionar soluciones en dominios interesantes. Entre los diferentes enfoques destacan [6]:

- Las características inspiradas en la física han permitido aproximar la función de valor utilizando una combinación lineal de características. Esto se debe a algunos trabajos realizados en robótica como las características para la estabilización local y las características que describen la dinámica del cuerpo rígido [6].
- Otra de las diferentes alternativas son las redes neuronales. Debido a que es inusual disponer de buenas funciones hechas a mano, varios grupos han empleado redes neuronales como una aproximación global de función de valor no lineal. Se han aplicado muchos modelos diferentes de redes neuronales en el aprendizaje de refuerzo robótico. Por ejemplo, se utilizaron perceptrones multicapa para aprender un comportamiento errante y tareas visuales [18]. La combinación de redes neuronales y lógica difusa (Fuzzy neural networks) también han permitido a los robots aprender comportamientos de navegación básica [19].

- La aplicación de modelos locales permitieron aprender tareas de navegación con evitación de obstáculos [20] o acciones de hockey aéreo [21].

3.2.3 Adaptación de las Value Functions en MotivEn

El GII se ha basado en el concepto formal de los métodos basados en funciones valor para definir los modelos de utilidad Value Function de MotivEn. En MotivEn, las Value Functions son funciones que calculan un valor de utilidad para cada uno de los puntos del espacio de estados del robot cuyo objetivo es proporcionar una ruta (secuencia de acciones) para llegar a su objetivo. Estas funciones se modelaron empleando ANNs para obtener la relación entre cada estado y su respectiva utilidad. A través de esta aproximación se pretendía obtener la acción óptima que permitiese al agente alcanzar el estado deseado de la forma más eficiente posible, lo que permitiría aprender a realizar distintas tareas de forma óptima.

El empleo de ANNs es uno de los métodos más utilizados para la definición de políticas óptimas dentro del aprendizaje por refuerzo robótico. Esto se debe a que las ANNs permiten desarrollar comportamientos robóticos a partir de la información recopilada del entorno, proporcionando así un mecanismo que permite predecir el mejor estado futuro de acuerdo con el estado sensorial del robot. La combinación entre el aprendizaje por refuerzo y el uso de ANNs ha sido utilizado en trabajos relacionados con la planificación y la navegación en entornos con obstáculos [6][22][23]. Sin embargo, lo que no se ha empleado hasta el momento eran algoritmos de aprendizaje online para desarrollar Value Functions. Esto da lugar a una alternativa novedosa que puede llegar a ser un punto de inicio para nuevas investigaciones y experimentos en robótica.

Capítulo 4

Tecnología y herramientas

En este capítulo se describen y justifican las herramientas tecnológicas utilizadas a lo largo de este proyecto.

4.1 ROS

Las siglas de ROS¹ hacen referencia a *Robot Operating System*, un framework de código abierto que engloba a un conjunto de herramientas y librerías para el desarrollo de software en robótica [24]. En este apartado se comentan las diferentes posibilidades que ROS nos ofrece y se describen los módulos de su arquitectura que han sido más relevantes para el software integrado en el simulador.

4.1.1 Características de ROS

ROS es una de las herramientas más utilizadas en el área de investigación de la robótica debido a los siguientes motivos [24]:

- Se trata de software libre bajo la licencia BSD. Esta característica resulta determinante para extender su uso y consolidar los módulos software que conforman su arquitectura. Cuenta con el soporte de una gran comunidad, manteniendo el principio de software abierto orientado a la integración y documentación.
- Es compatible con herramientas de código abierto como Gazebo², Rviz³, MoveIt⁴ o OpenCV⁵, lo que ha facilitado el desarrollo de librerías.

¹<http://www.ros.org/about-ros/>

²<http://gazebosim.org>

³<http://sdk.rethinkrobotics.com/wiki/Rviz>

⁴<https://moveit.ros.org/>

⁵<https://opencv.org/>

- Posee un sistema modular, lo que permite a los desarrolladores construir y probar sistemas utilizando los módulos de ROS que consideren necesarios.
- Proporciona un amplio conjunto de herramientas para gestionar sistemas informáticos distribuidos:
 - Herramientas de configuración del sistema.
 - Sistema de envío de mensajes estándar para la comunicación con el robot.
 - Comandos de terminal para acceder al estado del sistema en caso de no disponer de una interfaz de usuario.
 - Herramientas para crear interfaces de usuario.
 - Paquetes para resolver problemas básicos como localización, mapeo y navegación.
 - Monitorización.
- Ofrece una amplia colección de bibliotecas para implementar las distintas funcionalidades de un sistema robótico, con un enfoque en la movilidad, la manipulación y la percepción.
- ROS proporciona servicios propios de un sistema operativo pero sin serlo, ya que se instala sobre otro, en general Linux (se recomienda Ubuntu), por lo que sus desarrolladores lo definen como un Meta-Sistema Operativo. Entre sus características destacan [24]:
 - Posee su propio manejador de paquetes mediante comandos desde terminal para la gestión, compilación y ejecución de archivos.
 - Permite abstracción del hardware y de los controladores utilizados.
 - Dispone de una infraestructura de mensajería de publicación/suscripción diseñada para respaldar la construcción rápida y sencilla de sistemas informáticos distribuidos. El paso de mensajes se basa en el modelo XML-RPC.
 - Cuenta con librerías cliente de ROS implementadas en varios lenguajes de programación para agilizar la comunicación con los módulos de ROS.
 - * Roscpp implementada en C++.
 - * Rospy implementada en Python.
 - * RosJava implementada en JAVA.

ROS dispone de diversas distribuciones estables, sin embargo, la que recomiendan es ROS Kinetic Kame, la cual se ha empleado en este proyecto.

4.1.2 Componentes más relevantes en el proyecto

En esta sección se comentan los módulos de ROS que han sido más representativos en el desarrollo del comportamiento del robot. La razón de no profundizar en todos los detalles de su arquitectura se debe a que este framework no es el objeto de estudio de este trabajo, sino una de las herramientas que han servido para su consecución. En la Figura 4.1 se pueden ver los componentes que forman la estructura general de ROS.

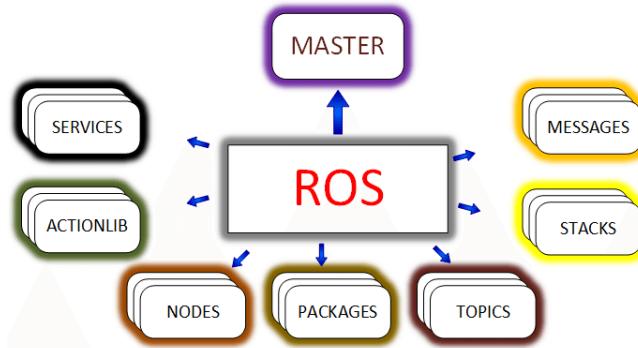


Figura 4.1: Componentes de ROS

La infraestructura de paso de mensajes es el componente clave que permite aprovechar las capacidades de ROS [24]. A continuación se describen los elementos que la componen y que permiten comprender su funcionamiento.

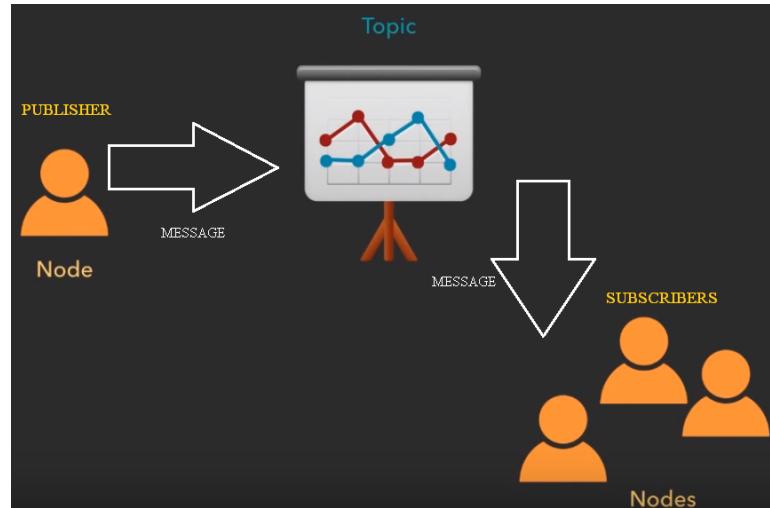


Figura 4.2: Comunicación básica entre los nodos de ROS

- **Nodos:** Como ya se mencionó previamente, ROS está organizado en módulos que realizan diversas funcionalidades, cada módulo es un nodo ROS y éste a su vez representa

a un proceso dentro de su grafo de cómputo. Existe un nodo en particular, el *Master*, sobre el cual se ejecutan los demás. El *Master* funciona como un servidor que conoce las direcciones de todos los nodos. Es lanzado cuando se ejecuta en la terminal el comando `roscore`, el cual carga una serie de nodos y programas necesarios en ROS, además de guardar la información de topics y servicios.

- **Topics:** Los topics (temas) son canales a través de los cuales los nodos envían o reciben mensajes. Normalmente los nodos que transmiten información reciben el nombre de *publishers* y los que escuchan el canal se llaman *subscribers*. Un mismo topic puede tener múltiples *publishers* y *subscribers*. Los *subscribers* y *publishers* no se conocen, el nodo *Master* es el responsable de avisar a los nodos que están escuchando un topic de que se ha enviado un nuevo mensaje, estableciendo así la comunicación. La Figura 4.2 ofrece una idea de la comunicación que se establece entre los distintos nodos, la cual es de tipo unidireccional. Cada topic está fuertemente tipado por el tipo de mensaje ROS usado para publicarlo por lo que los *subscribers* solo pueden recibir mensajes del tipo coincidente. El *Master* no impone la coherencia de tipos entre los *publishers*, pero los *subscribers* no establecerán el transporte de mensajes a menos que los tipos coincidan.
- **Mensajes:** Estructuras de datos simples que son utilizados para publicar información en los topics. Existen tipos primitivos estándar (Integer, Boolean ...), y pueden crearse mensajes personalizados.
- **Servicios:** Los servicios se basan en el envío de un par de mensajes, uno enviado por el cliente, el nodo que solicita una petición, y otro enviado a modo de respuesta por el nodo que ejecuta el servicio, el servidor. Los servicios son necesarios para permitir las interacciones de solicitud/respuesta RPC, que a menudo se requieren en un sistema distribuido. Un nodo ROS ofrece un servicio bajo un nombre y un cliente llama al servicio enviando el mensaje de solicitud y esperando la respuesta.

Una vez se inicializa el nodo *Master*, los nodos pueden comunicarse. Para crear un nodo que se comunique con el modelo del robot, sensor o cualquier elemento, es necesario utilizar una de las librerías cliente ROS que fueron mencionadas.

Para aclarar lo máximo posible el funcionamiento del sistema de mensajes de ROS se propone el siguiente ejemplo:

- Disponemos de un robot móvil que queremos desplazar una distancia concreta hacia delante. Para realizar esta tarea, nuestro algoritmo podría ser algo similar a esto:
 - Obtener posición actual.
 - Calcular posición final en función de la posición actual del robot.

- Desplazar el robot hacia delante hasta que la posición actual sea igual a la posición final que calculamos previamente.
- Este algoritmo en ROS necesitaría de un nodo que:
 - Reciba información de un topic en el que se publica la posición del robot.
 - Publique en otro topic la potencia necesaria para mover las ruedas del robot.
- Afortunadamente ROS ya dispone de estos dos topics, los cuales se explicarán más en profundidad cuando se analice el desarrollo del comportamiento del robot. El topic 1 proporcionará información relacionada con la odometría del robot mientras que el topic 2, permitirá enviar la potencia necesaria para mover las ruedas del robot. Primero se obtiene la posición del robot utilizando el topic 1 y se calcula la posición final del robot. Tras este paso, se envía al topic 2 un mensaje con la potencia que se debe aplicar a los motores del robot. Una vez el robot se comience a mover, basta con leer la información del topic 1 para conocer la situación del robot y compararla con la posición objetivo. Una vez llega éste a su destino, el nodo envía una potencia con valor cero al topic 2 para detener al robot. Un ejemplo de cómo podría ser la comunicación se puede ver en la Figura 4.3. En dicha representación, el nodo en el que se ejecuta el algoritmo es el node 2, el cual recibe la información del topic `/odom`, que es el topic 1, y envía la potencia al topic `/cmd_vel` que en la explicación anterior es el topic 2. El node 1 proporciona la posición de las ruedas del robot y el node 3 envía la potencia que se le indique a los motores del robot.

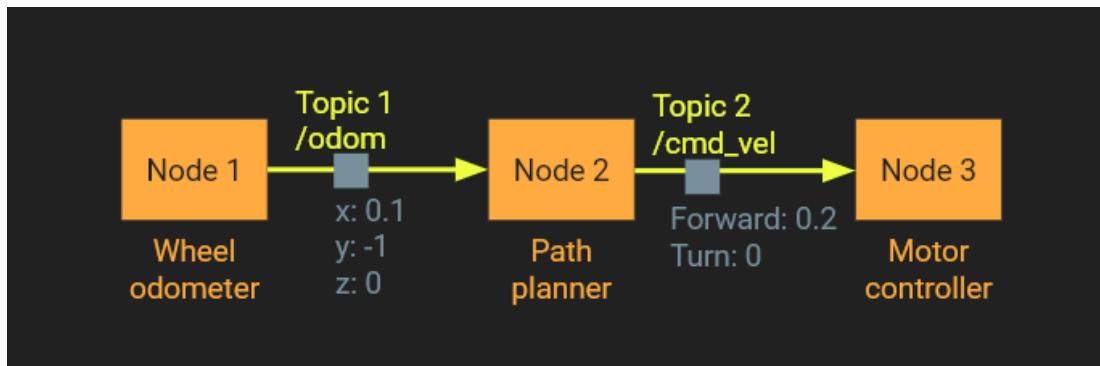


Figura 4.3: Comunicación entre los nodos para el ejemplo comentado.

Con este ejemplo se puede apreciar el potencial de ROS para definir comportamientos robóticos de una manera sencilla.

4.1.3 Papel de ROS en este trabajo

En este proyecto se ha utilizado la versión ROS Kinetic⁶, ya que era la empleada por los investigadores del GII para realizar los distintos experimentos. Esta decisión, aunque no es determinante para MotivEn, resulta sensata debido a que los investigadores que han solicitado la integración del simulador, son los responsables de este módulo de DREAM, por lo que el módulo a diseñar debe agilizar, en la medida de lo posible, su labor en el proyecto. Además, ROS dispone de las herramientas necesarias para comunicar al robot y al sistema motivacional de una forma modular y flexible que pueda ser fácilmente substituida si fuese necesario. Otro factor que se debe destacar, es la existencia de una gran comunidad que proporciona soporte y recomendaciones a seguir para que un desarrollo pueda ser reutilizable y mantenible, agilizando así determinados aspectos en proyectos de investigación.

ROS será el responsable de comunicar el módulo motivacional con el robot y definir sus funcionalidades. Debido a que el entorno del robot es un escenario 3D, se han combinado las características de ROS con las del simulador Gazebo con el objetivo de desarrollar el entorno de simulación [24].

Para definir el comportamiento del robot se ha utilizado la librería cliente de ROS en Python, rospy⁷. Esta decisión se ha realizado debido a que MotivEn está desarrollado actualmente en Python, por lo que resulta correcto seleccionar librerías que faciliten la integración del respectivo módulo y su posterior mantenimiento. Rospy es una API implementada en Python que proporciona funciones para gestionar el paso de mensajes de ROS, lo que permite crear nodos que publiquen y subscriban a los topics necesarios para guiar el comportamiento del sistema.

4.2 TensorFlow

TensorFlow⁸ es una librería software de código abierto diseñada por el equipo Google Brain Team y liberado en noviembre de 2015.

Se trata de una librería orientada al cómputo numérico mediante diagramas de flujo de datos. Los nodos del grafo representan operaciones matemáticas y las aristas matrices multidimensionales, a las que llamaremos tensores. Esta característica, junto con las diferentes funcionalidades de la biblioteca, permite desarrollar modelos neuronales con un gran número de capas de una manera flexible y eficiente, lo que hace que actualmente sea una de las bibliotecas más utilizadas en el área del Deep Learning [25].

⁶<http://wiki.ros.org/kinetic>

⁷<http://wiki.ros.org/rospy>

⁸www.tensorflow.org

4.2.1 Características de TensorFlow

Entre las características de esta biblioteca cabe destacar que [25]:

- Posee APIs de alto nivel en diferentes lenguajes de programación, permitiendo desarrollar diversos tipos de modelos neuronales.
- Dispone de herramientas para visualizar y ajustar redes neuronales, facilitando el desarrollo para el programador. Entre estas herramientas, las más utilizadas son TensorBoard y TensorFlow Playground ⁹.
- Cuenta con una gran comunidad que proporciona documentación y diversos proyectos en repositorios de GitHub¹⁰, así como tutoriales para iniciarse dentro del aprendizaje máquina.

4.2.2 Papel de TensorFlow en este trabajo

La librería TensorFlow ha sido utilizada para el aprendizaje de las Value Functions basadas en Deep Learning. En este trabajo se empleó la API de Python para TensorFlow, en concreto, la API Keras¹¹, la cual está orientada al desarrollo de redes neuronales. Keras es la librería recomendada para iniciarse en el Deep Learning debido a que permite diseñar modelos neuronales de manera sencilla y flexible [26]. En el caso de este proyecto, estos modelos emplean algoritmos estocásticos, característicos del aprendizaje abierto, para diseñar los modelos de utilidad. Otro factor clave para su elección, ha sido la disposición de una API en Python, lo que facilita la integración de este módulo con el software previamente implementado en el MotivEn.

4.3 Gazebo

Gazebo es un simulador de entornos 3D que permite simular y evaluar el comportamiento de uno o más robots sobre escenarios de diversa complejidad. Gazebo nace del proyecto Gazebo Project y es software libre financiado, en parte, por Willow Garage¹², pudiendo ser reconfigurado, ampliado y modificado [27].

⁹<https://playground.tensorflow.org/>

¹⁰<https://github.com/Hvass-Labs/TensorFlow-Tutorials>

¹¹www.tensorflow.org/guide/keras

¹²<http://www.willowgarage.com/>

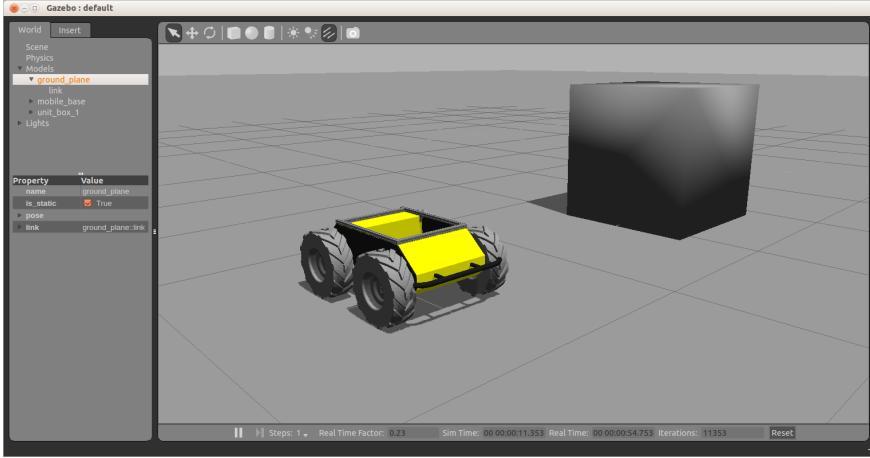


Figura 4.4: Ejemplo de simulación en Gazebo en el que hay un robot móvil y un objeto.

4.3.1 Características principales

A grandes rasgos, Gazebo dispone de las herramientas necesarias para analizar el comportamiento de experimentos robóticos, entre ellas se destacan las siguientes [27][28]:

- Permite crear y configurar mundos virtuales de acuerdo a las necesidades del experimento.
- Capacidad para diseñar modelos robóticos propios, además de posibilitar la importación y/o carga de modelos ya desarrollados. Además contiene plugins para agregar diferentes tipos de sensores a los modelos de robot.
- Soporta simulación de físicas sobre cuerpos rígidos, permitiendo la interacción entre el robot y los distintos elementos de su entorno soportando estos los efectos de la gravedad. La simulación de la cinemática y de la dinámica de cuerpos rígidos la realiza el motor de físicas *Open Dynamics Engine* (ODE).
- Es compatible con ROS, el cual permite lanzar el simulador y se encarga de la comunicación con el entorno.

4.3.2 Integración con ROS

Como ya se mencionó en el apartado 4.1.3, el simulador Gazebo se ha utilizado combinándolo con las características de ROS. La API de ROS dispone de un plugin que inicializa un nodo ROS llamado *gazebo*, una serie de topics (como el */gazebo/model_states* empleado en este trabajo) y luego integra el programador de devolución de llamadas ROS con el programador interno de Gazebo para proporcionar las interfaces ROS. Estas interfaces se corresponden con

los topics y servicios que componen el sistema de paso de mensajes de ROS, de esta forma se logra la compatibilidad y comunicación entre ROS y Gazebo. Además, ROS posee comandos para lanzar Gazebo, este es el caso del comando *roslaunch*, el cual ha sido empleado para ejecutar el simulador desarrollado en este trabajo [24][27].

4.3.3 Papel de Gazebo en este trabajo

En este trabajo se empleó Gazebo para definir problemas robóticos realistas en los que probar MotivEn. El uso de este simulador en el proyecto se debe principalmente a la compatibilidad con ROS, el cual permite que la comunicación con el robot sea flexible y eficiente.

Capítulo 5

Metodología

En este capítulo se explica la metodología empleada para la realización de este proyecto, detallándose los procedimientos realizados. El hecho de que este trabajo sea un proyecto de desarrollo en investigación hace que la adaptación de las metodologías de desarrollo clásico resulte más difícil, por lo tanto a lo largo de esta sección se explica como se ha abordado y solventado esta desventaja.

5.1 Roles en el proyecto

Aunque el concepto de rol es más popular en el ámbito de las metodologías de ingeniería del software, se definieron dos tipos de roles para clarificar el papel de los participantes en el proyecto. De esta forma tenemos:

- Un rol denominado *equipo de desarrollo*, el cual está representado por Yeray Méndez Romero. El *equipo de desarrollo* es el encargado de desarrollar los módulos software solicitados.
- El rol *investigadores* fue desempeñado por Francisco Javier Bellas Bouza y Alejandro Romero Montero, ambos investigadores del GII.

5.2 Estructuración del proyecto

En este proyecto se ha abordado la integración de una implementación de Value Functions basada en Deep Learning y de un entorno de simulación 3D en la arquitectura motivacional MotivEn. Para establecer una forma de gestionar las distintas tareas, y en consecuencia establecer una planificación, se tomó la decisión de descomponer la realización del proyecto en una serie de iteraciones.

El concepto de iteración es uno de los más empleados en las metodologías de desarrollo de proyectos software, un ejemplo de ello es la metodología ágil eXtreme Programming

[29]. En este caso, una iteración representa una serie de tareas realizadas para desarrollar y/o configurar uno o varios componentes. Esto permitirá tener una visión clara del progreso en el proyecto, lo que facilitará la planificación de las tareas pendientes. Para analizar la evolución del proyecto se celebraron reuniones entre los participantes. Dichas reuniones también permitieron aclarar dudas y establecer las tareas a realizar en la siguiente iteración.

5.2.1 Iteración

Una iteración, en el caso de este proyecto, se puede describir como un comentario genérico que contiene información sobre la fase de desarrollo en cuestión. Entre la información que contiene hay que destacar:

- Información básica de la iteración. Contiene la siguiente información:
 - Fecha de inicio y fecha de finalización. Indican el inicio y el final de la respectiva iteración.
 - Tareas. Se enumeran las tareas principales que se realizarán en dicha iteración.
 - Descripción breve. Se comentan brevemente las tareas y objetivos a realizar en dicha fase.
- Resumen. Descripción precisa de las actividades realizadas para lograr con éxito las tareas propuestas.
- Análisis de la iteración. Se evalúa el desarrollo de la iteración comparándola con la estimación inicial. Permitirá conocer el progreso global sobre el proyecto.

5.3 Gestión del proyecto

Para intentar controlar el desarrollo del proyecto se han empleado algunas de las herramientas de la asignatura *Xestión de Proxectos*, las cuales permiten:

- Satisfacer las necesidades de información de gestión.
- Desarrollar el proyecto de acuerdo a la planificación y costes establecidos.
- Optimizar el uso de recursos y la distribución de esfuerzo.

El motivo de apoyarse en esta metodología se debe a que está más enfocada a la administración de un proyecto que a la clase de proyecto a desarrollar.

5.3.1 Estimación

La estimación realizada para cada iteración se calculó en función del componente a realizar. En este apartado hay que destacar que las tecnologías empleadas eran desconocidas por el *equipo de desarrollo*, lo que representa un riesgo a tener en cuenta en la estimación de la tarea, y en consecuencia, en la planificación del proyecto. En contraposición con este aspecto, los requisitos ya habían sido estudiados y definidos por los *investigadores* cuando realizaron el simulador 2D y la primera aproximación de Value Functions, lo que disminuye la posibilidad de cambios o problemas relacionados con ellos. Sin embargo, se intentó no ser demasiado optimistas en la realización de las diversas tareas.

El tiempo dedicado a desarrollo fue de 5 horas/día debido a motivos laborales, lo que influyó en el tiempo de realización de las tareas.

5.3.2 Planificación

Las iteraciones que conforman el proyecto son las siguientes:

- **Iteración 1:** Análisis global de la arquitectura de MotivEn
- **Iteración 2:** Desarrollo y análisis de la VF.
- **Iteración 3:** Integración del módulo *Value Functions Module*.
- **Iteración 4:** Diseño del módulo *Simulator Module*.
- **Iteración 5:** Diseño del entorno virtual y desarrollo del comportamiento robótico.
- **Iteración 6:** Desarrollo de la configuración del sistema motivacional.

5.3.3 Recursos

El proyecto se ha realizado empleando dos recursos humanos:

- El *equipo de desarrollo*: Yeray Méndez Romero.
- Los directores: Francisco Javier Bellas Bouza y Alejandro Romero Montero. Además, representarán el rol de *investigadores* en el proyecto.

Los recursos de desarrollo son propiedad del *equipo*, exceptuando el sistema motivacional MotivEn, el cual fue proporcionado por los *investigadores* del GII.

5.3.4 Gestión de riesgos

Los riesgos representan acontecimientos que pueden afectar al desarrollo del proyecto. En este proyecto, el hecho de que los requisitos ya fueran establecidos en su día por los investigadores para realizar el sistema motivacional MotivEn minimiza los riesgos en gran medida. Sin embargo, continúa existiendo cierto peligro debido a los riesgos que se comentan a continuación.

Clasificación de riesgos

Los riesgos identificados en este proyecto se muestran en siguiente la tabla:

Código	Descripción	Probabilidad	Impacto	Exposición
R1	Problemas con la tecnología	Media	Media	Media
R2	Desarrollo incorrecto	Media	Baja	Media
R3	Mala estimación en la planificación	Media	Alta	Alta

Cuadro 5.1: Riesgos identificados en el proyecto.

Prevención

Los riesgos con un valor de exposición alta representan problemas que pueden retrasar en gran medida el desarrollo del proyecto. Por este motivo, dichos riesgos deben de ser analizados exhaustivamente para evitar su aparición.

- **R3 - Mala estimación en la planificación:** debido a la falta de experiencia para la planificación y con las tecnologías empleadas, es probable que la realización de las iteraciones se desvíe de lo planeado. Por lo tanto, se realizará una planificación inicial en cada iteración que permita analizar el progreso y evitar retrasos en la medida de lo posible.

Seguimiento sobre los riesgos de exposición media

Debido a que los riesgos con una exposición media pueden convertirse en riesgos de alta exposición, se realizará un seguimiento que controle dichos riesgos:

- **R1 - Problemas con la tecnología:** se analizará la tecnología a través de la documentación disponible.
- **R2 - Desarrollo incorrecto:** se evaluará el desarrollo mediante una serie de pruebas.

Capítulo 6

Desarrollo

6.1 Aspectos del desarrollo

6.1.1 Requisitos funcionales

El hecho de que los *investigadores* del GII desarrollaran la versión inicial de MotivEn ayudó a establecer los requisitos que debían cumplirse en este trabajo. De esta forma, se fijaron los siguientes requisitos de diseño:

- El código desarrollado se debe realizar empleando Python 2.7.x para mantener la compatibilidad con el código de MotivEn.
- La implementación de la Value Function se debe realizar empleando la biblioteca TensorFlow y empleando el algoritmo de optimización Adam como ejemplo de su correcto funcionamiento en las pruebas.
- El escenario de simulación 3D para las pruebas debe realizarse empleando Gazebo. En consecuencia, los modelos robóticos empleados deben ser compatibles con este último.
- El comportamiento robótico debe implementarse empleando el framework ROS.
- El sistema motivacional resultante debe mantener su correcto funcionamiento independientemente del entorno de simulación empleado.
- El sistema motivacional resultante debe poder emplear las Value Functions de las que disponga.

6.1.2 Requisitos no funcionales

Los requisitos no funcionales especifican características a cumplir por el sistema. Además, representan un criterio importante para tomar decisiones de diseño durante el desarrollo de los componentes. Se han definido los siguientes requisitos no funcionales:

- **Mantenibilidad:** Los componentes desarrollados deben facilitar su modificación o extensión.
- **Integración sencilla:** La módulos integrados deberán permitir la integración de nuevas implementaciones y configuraciones de forma sencilla.

6.2 Iteración 1: Análisis global de la arquitectura de MotivEn

6.2.1 Información básica de la iteración

- Fecha inicio: 23 de julio del 2018.
- Fecha de finalización: 26 de julio del 2018.
- Tareas:
 - Analizar la arquitectura de MotivEn.
 - Proponer posible arquitectura resultante.
- Descripción breve:
 - De cara a establecer la arquitectura resultante una vez se integren los componentes a desarrollar, se analizará la estructura actual de MotivEn.

6.2.2 Resumen

La primera fase del proyecto comenzó tras reunirse el *equipo de desarrollo* y los *investigadores* en el laboratorio del GII en el campus de Esteiro. Esta reunión sirvió para analizar el funcionamiento y tareas del sistema MotivEn. El objetivo a lograr en esta fase inicial, era obtener una visión global del sistema y poder diseñar una posible arquitectura que permitiese integrar los respectivos módulos.

Análisis de la arquitectura de MotivEn

El funcionamiento de MotivEn es realizado por los componentes explicados en el Apartado 2.1.3. La arquitectura cognitiva gestiona el descubrimiento de los distintos objetivos, las motivaciones asociadas a los mismos y la selección de la acción más idónea en función del estado actual del robot. Para ello, debe obtener la información necesaria del entorno del robot. En la arquitectura inicial, la información recopilada procedía del *2D Simulator*. Para realizar la integración del entorno empleando el simulador Gazebo, fue necesario establecer un forma de abstraer los distintos comportamientos robóticos. En cuanto al módulo *Utility Models Manager*, inicialmente solo disponía de una única VF. Dado que existía una gran probabilidad

de que MotivEn necesitara emplear diversos tipos de funciones de utilidad para lograr que el robot se adaptase a su entorno, era importante encapsular las diversas aproximaciones.

Arquitectura resultante propuesta

La arquitectura propuesta se basó en desarrollar dos módulos que encapsulasen los diversos tipos de Value Functions y la comunicación con los distintos entornos y comportamientos robóticos respectivamente. Con esta arquitectura se buscó aumentar la independencia entre los distintos módulos de MotivEn y abstraer las distintas implementaciones que pudieran ser necesarias integrar en los respectivos módulos.

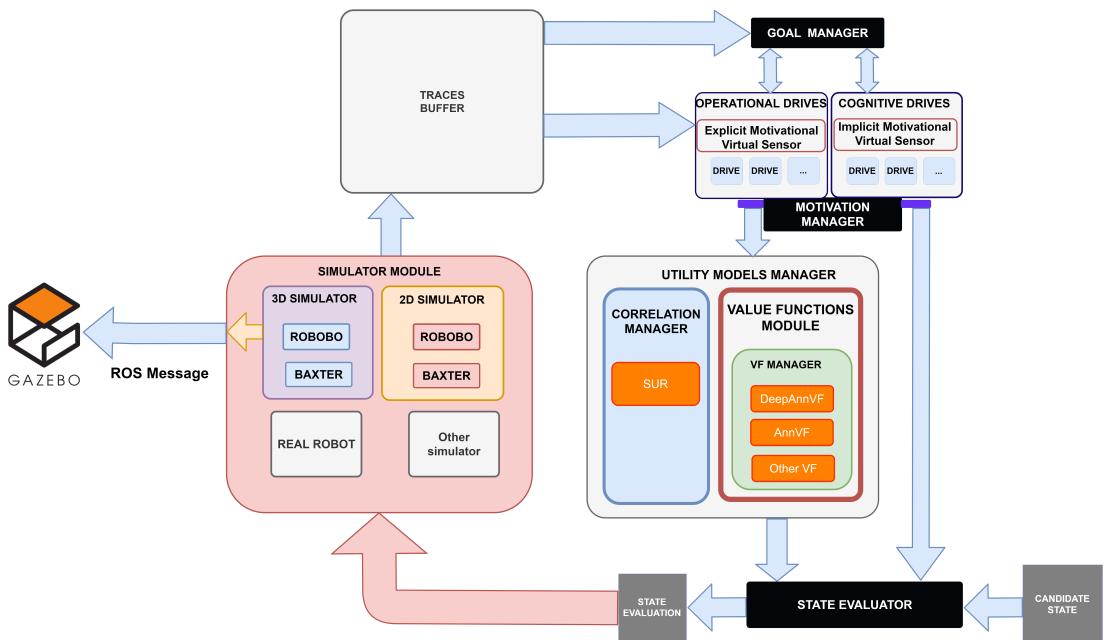


Figura 6.1: Arquitectura resultante propuesta.

La Figura 6.1 muestra la arquitectura posterior a la integración de los módulos comentados. En ella, se ha definido el *Simulator module* para encargarse de la comunicación con el entorno robótico, obtener información y transmitir al robot las acciones seleccionadas por el sistema motivacional. Mediante esta aproximación, se podría disponer de una multitud de experimentos sin necesidad de modificar el comportamiento de MotivEn. Esto mismo ocurriría con el módulo *Value Functions Module*, el cual gestionará las funciones de utilidad disponibles, proporcionando la VF asociada al objetivo actual. Con la arquitectura propuesta, se lograría encapsular el comportamiento de las distintas VFs y simuladores, lo que permitiría mantener e integrar nuevas implementaciones en el sistema de forma más sencilla y escalable.

6.2.3 Análisis de la iteración

Como se puede observar en la Figura 6.2, las tareas fueron realizadas dentro del tiempo previsto inicialmente. Esto se debió en gran parte a las explicaciones realizadas por los *investigadores* sobre el funcionamiento del sistema.

	Modo de tarea	Nombre de tarea	Duración	Duración de línea base	Comienzo de línea base	Fin	Fin de línea base	
✓	➡	▫ Proyecto	96 días	63,4 días	lun 23/07/18	lun 23/07/18	jue 03/12/18	jue 18/10/18
✓	➡	▫ Iteración 1	3,8 días	4,2 días	lun 23/07/18	lun 23/07/18	jue 26/07/18	vie 27/07/18
✓	➡	Reunión para analizar el sistema motivacional	0 días	0 días	lun 23/07/18	lun 23/07/18	lun 23/07/18	lun 23/07/18
✓	➡	Analizar arquitectura	7 horas	6 horas	lun 23/07/18	lun 23/07/18	mar 24/07/18	mar 24/07/18
✓	➡	Analizar componentes principales	3 horas	5 horas	mar 24/07/18	mar 24/07/18	mié 25/07/18	mié 25/07/18
✓	➡	Estudiar posibles soluciones	3 horas	5 horas	mié 25/07/18	mié 25/07/18	jue 26/07/18	jue 26/07/18
✓	➡	Diseñar arquitectura resultante	6 horas	5 horas	mié 25/07/18	jue 26/07/18	jue 26/07/18	vie 27/07/18
✓	➡	Reunión para analizar la arquitectura	0 días	0 días	jue 26/07/18	vie 27/07/18	jue 26/07/18	vie 27/07/18

Figura 6.2: Tareas realizadas en la Iteración 1.

6.3 Iteración 2: Desarrollo y análisis de la VF

6.3.1 Información básica de la iteración

- Fecha inicio: 26 de julio de 2018.
- Fecha de finalización: 6 de septiembre del 2018.
- Tareas:
 - Desarrollar un modelo de Value Function empleando TensorFlow.
 - Realizar pruebas para determinar la configuración de red que mejor se ajusta al experimento a resolver.
- Descripción:

- En esta iteración se estableció como objetivo principal el desarrollo de la nueva aproximación al aprendizaje de las Value Function empleando Deep Learning, en este caso, a través de TensorFlow y del algoritmo específico Adam. Una vez finalizada esta tarea se realizarían una serie de pruebas para determinar la topología y parámetros de la VF desarrollada.

6.3.2 Resumen

Una vez aceptada la arquitectura propuesta por los *investigadores*, éstos determinaron las tareas de la siguiente iteración durante la reunión celebrada. Además, se definieron las pruebas necesarias para establecer la configuración de la VF.

Desarrollo de la VF

Se empleó la API Keras de TensorFlow para desarrollar la implementación de la Value Function planteada en el trabajo. Keras dispone de los métodos necesarios para definir la topología de la red, entrenar el modelo y realizar predicciones. Sin embargo, para facilitar la integración, el *equipo de desarrollo* junto con los *investigadores* decidieron que la mejor opción se basaba en desarrollar una clase que emplease los métodos de dicha API.

Una vez finalizado el desarrollo, se realizaron pruebas basadas en la validación cruzada para determinar la topología y los hiperparámetros de la VF a utilizar en el experimento final. Dichas pruebas se explicarán en el Apartado 7.2.

6.3.3 Análisis de la iteración

En la planificación mostrada en la Figura 6.3, se puede apreciar un retraso notable en la fase de desarrollo y posteriormente en el tiempo requerido por las pruebas. El primer caso fue producido por la inexperiencia con la biblioteca TensorFlow, ya que fueron necesarias varias correcciones en la implementación antes de finalizar dicho desarrollo. Las pruebas realizadas, también sufrieron un gran retraso debido a diversas confusiones en las métricas a calcular y en el tiempo requerido para realizar la validación cruzada, el cual fue muy superior a lo inicialmente planificado. Esta iteración supuso un fuerte contratiempo en el progreso del proyecto.

Modo de tarea	Nombre de tarea	Duración	Duración de línea base	Comienzo	Comienzo de línea base	Fin	Fin de línea base
✓	Iteración 2	29,4 días	15 días	jue 26/07/18	vie 27/07/18	jue 06/09/18	vie 17/08/18
✓	Reunión	0 días	0 días	jue 26/07/18	vie 27/07/18	jue 26/07/18	vie 27/07/18
✓	Analizar documentación de TensorFlow	23 horas	15 horas	jue 26/07/18	vie 27/07/18	jue 02/08/18	mié 01/08/18
✓	Realizar implementación VF	57 horas	35 horas	jue 02/08/18	mié 01/08/18	vie 17/08/18	vie 10/08/18
✓	Realizar pruebas	67 horas	25 horas	vie 17/08/18	vie 10/08/18	jue 06/09/18	vie 17/08/18
✓	Reunión para evaluar los resultados	0 días	0 días	jue 06/09/18	vie 17/08/18	jue 06/09/18	vie 17/08/18

Figura 6.3: Tareas realizadas en la Iteración 2.

6.4 Iteración 3: Integración del módulo *Value Functions Module*

6.4.1 Información básica de la iteración

- Fecha inicio: 6 de septiembre del 2018.
- Fecha de finalización: 24 de septiembre del 2018.
- Tareas:
 - Desarrollar un manager que gestione las Value Functions del sistema.
 - Integrar el módulo de Value Functions en MotivEn.
- Descripción:
 - Para integrar las Value Functions desarrolladas y permitir que estas pudiesen ser empleadas por el sistema, se desarrolló el módulo *Value Functions Module*.

6.4.2 Resumen

En la reunión celebrada se analizaron cuales serían las funciones a realizar por las diversas VFs de cara a definir los diferentes componentes y responsabilidades del módulo.

Módulo *Value Functions Module*

Se desarrolló este módulo con el objetivo de gestionar las diferentes clases de Value Functions de una manera más sencilla e independiente desde el punto de vista del motor moti-

vacional. Para ello, se desarrolló un manager que proporcionaría al sistema motivacional la función de utilidad acorde con la motivación extrínseca y el objetivo actual del sistema. Para permitir que MotivEn emplease las distintas VFs de una manera sencilla, se definió una interfaz común para las distintas funciones de valor. Con esta decisión, resultaría más sencillo desarrollar integrar nuevos modelos de VFs, ya que no sería necesario modificar el comportamiento del motor de la aplicación.

Al igual que las VFs, se definió una interfaz para el *ValueFunctionsManager* que permitiría la comunicación entre el módulo *Value Functions Module* y los componentes principales del sistema. Este gestor, serviría de intermediario entre las Value Functions y el motor motivacional. De este modo, se logró una mayor abstracción entre las funciones de valor disponibles y se delegó dicha responsabilidad en el *ValueFunctionManager*, el cual debería proporcionar la VF con más certeza de alcanzar el objetivo. En la Figura 6.4 se encuentra el diagrama UML del módulo *Value Functions Module*. Una vez desarrollados los componentes comentados, se finalizó la integración del respectivo módulo.

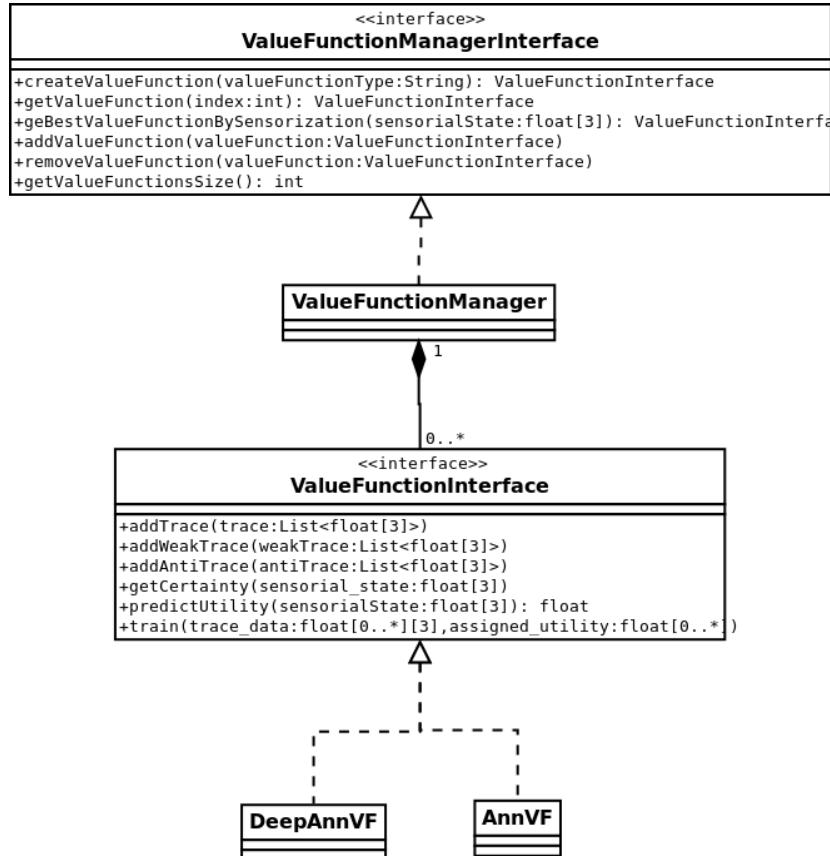


Figura 6.4: Diagrama UML del módulo *Value Functions Module*.

6.4.3 Análisis de la iteración

Las tareas de la iteración fueron finalizadas con dos días de retraso con respecto a lo inicialmente previsto. El desarrollo de los diferentes componentes y la posterior integración, no resultaron problemáticos ya que se habían analizado previamente las responsabilidades de los mismos.

Modo de tarea	Nombre de tarea	Duración	Duración de línea base	Comienzo	Comienzo de línea base	Fin	Fin de línea base
✓	Iteración 3	12 días	10 días	jue 06/09/18	vie 17/08/18	lun 24/09/18	vie 31/08/18
✓	Reunión	0 días	0 días	jue 06/09/18	vie 17/08/18	jue 06/09/18	vie 17/08/18
✓	Analizar componentes necesarios	3 horas	5 horas	jue 06/09/18	vie 17/08/18	jue 06/09/18	lun 20/08/18
✓	Desarrollar módulo VFs	17 horas	15 horas	jue 06/09/18	lun 20/08/18	mié 12/09/18	jue 23/08/18
✓	Desarrollar gestor de VFs	14 horas	10 horas	mié 12/09/18	jue 23/08/18	vie 14/09/18	lun 27/08/18
✓	Integrar VFs	7 horas	5 horas	lun 17/09/18	lun 27/08/18	mar 18/09/18	mar 28/08/18
✓	Refactorización	4 horas	5 horas	mar 18/09/18	mar 28/08/18	mié 19/09/18	mié 29/08/18
✓	Integrar módulo	15 horas	10 horas	mié 19/09/18	mié 29/08/18	lun 24/09/18	vie 31/08/18

Figura 6.5: Tareas realizadas en la Iteración 3.

6.5 Iteración 4: Diseño del módulo *Simulator Module*

6.5.1 Información básica de la iteración

- Fecha inicio: 24 de septiembre del 2018.
- Fecha de finalización: 9 de octubre del 2018.
- Tareas:
 - Compatibilizar la comunicación con el simulador 2D y el simulador 3D.
 - Desarrollar módulo *Simulator Module*.
- Descripción:
 - Para compatibilizar el funcionamiento del sistema con los futuros entornos de simulación, entre los que se encuentra el del escenario creado con Gazebo, se decidió desarrollar un módulo que encapsulara la comunicación con los distintos simuladores.

6.5.2 Resumen

En la reunión celebrada al comienzo de la iteración, se determinó que el sistema motivacional debía ser compatible con ambos simuladores (2D y 3D) y se analizaron algunas de las alternativas a seguir.

Módulo *Simulator Module*

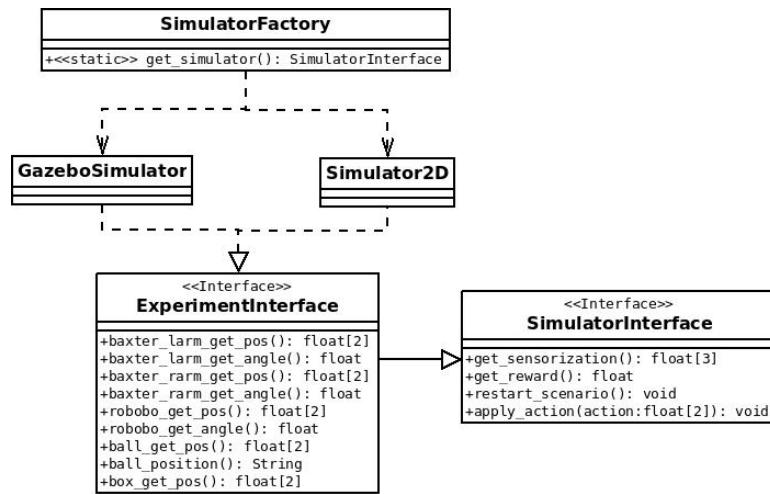


Figura 6.6: Diagrama de clases del módulo *Simulator Module*.

Para permitir la comunicación con los distintos simuladores se desarrolló la interfaz *SimulatorInterface*. Dicha interfaz, permitirá que el sistema pudiese recopilar la información necesaria sin necesidad de conocer su procedencia (entorno virtual o real), permitiendo desarrollar diferentes tipos de entornos e integrarlos con más facilidad. Teniendo en cuenta la variabilidad de los experimentos que se podrían realizar en cualquiera de los respectivos entornos, se decidió definir una interfaz que encapsulase el comportamiento robótico del experimento a realizar. Esta interfaz permitiría compatibilizar el funcionamiento del simulador 2D y el entorno del simulador Gazebo, ya que ambos realizarían el mismo experimento. Por lo tanto, se definió la interfaz *ExperimentInterface* con los métodos necesarios para que ambos simuladores pudiesen ser utilizados.

Para determinar qué componente se encargaría de la comunicación con el respectivo entorno robótico se decidió aplicar el patrón factoría. De esta forma, se abstrae la lógica del sistema del respectivo simulador. La factoría *SimulatorFactory* leería, por ejemplo, un parámetro de configuración de un fichero y proporcionaría la implementación de la clase encargada de la comunicación entre el simulador y el sistema motivacional. De este modo, los distintos componentes del sistema y el módulo *Simulator Module* pueden mostrarse agnósticos entre ellos,

lo que resultará útil para incluir nuevos experimentos. En la Figura 6.6 se muestra el diagrama UML con las clases e interfaces mencionadas.

6.5.3 Análisis de la iteración

El desarrollo e integración del módulo se llevó a cabo sin demasiados problemas. El retraso sufrido se debió a desviaciones sobre el tiempo inicialmente planificado para analizar y diseñar determinados componentes del módulo.

Modo de tarea	Nombre de tarea	Duración	Duración de línea base	Comienzo	Comienzo de línea base	Fin	Fin de línea base
✓	Iteración 4	11 días	8,4 días	lun 24/09/18	vie 31/08/18	mar 09/10/18	mié 12/09/18
✓	Reunión	0 días	0 días	lun 24/09/18	vie 31/08/18	lun 24/09/18	vie 31/08/18
✓	Anализar simulador desarrollado por el GII	13 horas	10 horas	lun 24/09/18	vie 31/08/18	mié 26/09/18	mar 04/09/18
✓	Anализar componentes	10 horas	7 horas	mié 26/09/18	mar 04/09/18	vie 28/09/18	mié 05/09/18
✓	Diseñar módulo Simulator	14 horas	10 horas	vie 28/09/18	mié 05/09/18	mié 03/10/18	vie 07/09/18
✓	Refactorización	3 horas	5 horas	mié 03/10/18	vie 07/09/18	jue 04/10/18	lun 10/09/18
✓	Integrar módulo	15 horas	10 horas	jue 04/10/18	lun 10/09/18	mar 09/10/18	mié 12/09/18

Figura 6.7: Tareas realizadas en la Iteración 4.

6.6 Iteración 5: Diseño del entorno virtual y desarrollo del comportamiento robótico

6.6.1 Información básica de la iteración

- Fecha inicio: 9 de octubre del 2018.

- Fecha de finalización: 9 de noviembre del 2018.

- Tareas:

- Desarrollar el entorno virtual.
- Diseñar modelo 3D del robot móvil.
- Desarrollar comportamiento robótico.

- Descripción:

- El entorno virtual 3D y el comportamiento robótico se desarrolló empleando el simulador Gazebo en combinación con ROS. El simulador a desarrollar estará compuesto por:
 - * Comportamiento robótico del experimento. Este archivo se encuentra dentro del módulo *Simulator Module*.
 - * Paquete ROS con el entorno y modelos virtuales, el cual se encuentra en el *workspace* de ROS en el equipo.

6.6.2 Resumen

En la reunión celebrada inicialmente, se analizaron posibles modelos de robots móviles que podían ser empleados durante las simulaciones y se estudió el comportamiento del experimento robótico a replicar. Esta información permitiría establecer qué robots se emplearían en el experimento y qué acciones debían ejecutar.

Entorno y modelos virtuales del simulador

Para definir el entorno y los modelos virtuales fue necesario definir un paquete ROS dentro de su *workspace*. Los paquetes ROS son directorios que contienen los archivos necesarios para realizar un experimento en simulación: modelos 3D, archivos que permiten cargar el entorno en el simulador Gazebo, scripts, plugins ... Se desarrolló el paquete *simulator*, el cual está compuesto por los siguientes archivos y carpetas:

- Directorio *models*. Carpeta con los modelos virtuales empleados: un cubo, un robot móvil, una mesa y una lámina (*goal*).
- Directorio *scripts*: contiene el script *simulator.py* el cual permitirá cargar los objetos 3D en Gazebo de forma más precisa.
- Directorio *launch*: este directorio contiene los archivos que permitirán lanzar aplicaciones ROS, además del simulador Gazebo. Contiene el archivo *simulator.launch* en el cual se selecciona el mundo virtual que se debe lanzar en Gazebo y carga el modelo virtual de los robots.

Los modelos que contiene el directorio *models* estaban disponibles en Gazebo, siendo la única excepción, el robot diseñado en este trabajo. Para diseñar el modelo de robot en Gazebo fue necesario crear los siguientes archivos:

- *model.config*. Contiene meta-information en formato XML del modelo del robot: nombre del modelo, autor, versión del modelo, versión del archivo sdf que describe el modelo, etc. En la Figura 6.8 se puede ver la definición del archivo realizada para el robot diseñado.



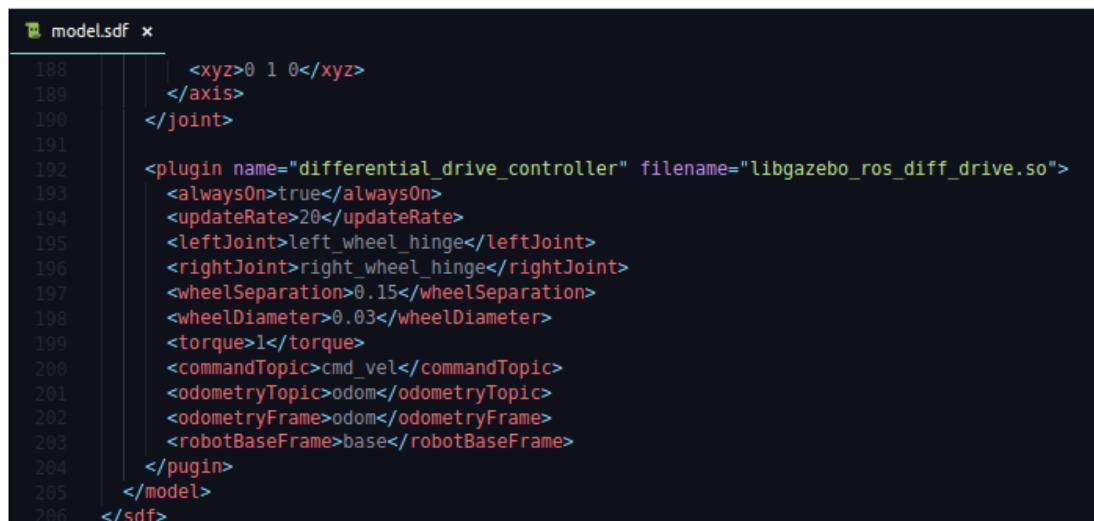
```

1  <?xml version="1.0"?>
2  <model>
3      <name>Robot Movil</name>
4      <version>1.0</version>
5      <sdf version='1.4'>model.sdf</sdf>
6
7      <author>
8          <name>Yeray</name>
9          <email></email>
10         </author>
11
12         <description>
13             Robot movil del experimento.
14         </description>
15     </model>

```

Figura 6.8: Archivo *model.config*.

- *model.sdf*: SDF es un formato XML para definir objetos y entornos en simuladores robóticos y es el formato más utilizado en Gazebo para definir los elementos anteriores. En este archivo es donde se han definido los distintos componentes del robot móvil (chasis, ruedas...). Además, para hacer que el robot fuese accesible, se integró el plugin *differential_drive_controller* de ROS. En la Figura 6.9 se muestra la carga del plugin en el fichero *model.sdf* del robot. Este plugin es un controlador que permite enviar comandos de velocidad a robots con ruedas motrices.



```

188     <xyz>0 1 0</xyz>
189     </axis>
190   </joint>
191
192   <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
193       <alwaysOn>true</alwaysOn>
194       <updateRate>20</updateRate>
195       <leftJoint>left_wheel_hinge</leftJoint>
196       <rightJoint>right_wheel_hinge</rightJoint>
197       <wheelSeparation>0.15</wheelSeparation>
198       <wheelDiameter>0.03</wheelDiameter>
199       <torque>1</torque>
200       <commandTopic>cmd_vel</commandTopic>
201       <odometryTopic>odom</odometryTopic>
202       <odometryFrame>odom</odometryFrame>
203       <robotBaseFrame>base</robotBaseFrame>
204   </plugin>
205 </model>
206 </sdf>

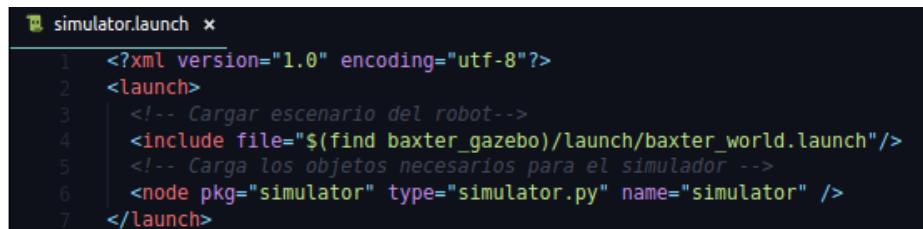
```

Figura 6.9: Plugin en el archivo *model.sdf*.

Para lanzar el simulador Gazebo a través de ROS, se utilizó el comando *roslaunch*, que sirve

para lanzar varios nodos de ROS de un paquete de forma automática y con una configuración específica impuesta por un fichero XML con extensión *.launch*. Este comando se ejecuta en una consola de terminal indicándole el paquete ROS (*simulator*) y el archivo con extensión *.launch* en el que se definió la configuración del entorno que se desea ejecutar. Los ficheros *.launch* son ficheros en formato XML que permiten definir en gran parte la configuración del entorno que se lanzará en Gazebo. Al igual que en XML, se emplean tags para definir la configuración deseada. El archivo *simulator.launch*, como se puede ver en la Figura 6.10, está formado por los siguientes tags:

- *launch*: representa el tag principal, actúa como un contenedor para el resto de elementos.
- *include*: permite importar otro archivo XML de lanzamiento en el archivo actual. En el archivo *simulator.launch*, se carga el fichero de lanzamiento *baxter_world.launch*, el cual carga el modelo virtual del Baxter junto con un modelo de mundo ya existente en Gazebo.
- *node*: especifica el nodo ROS que queremos lanzar. Este tag posee una serie de atributos que permitirán cargar el script *simulator.py*.
 - *pkg*: recibe el nombre del paquete donde se inicializa el nodo ROS.
 - *type*: se corresponde con el nombre del ejecutable.
 - *name*: nombre del nodo, debe coincidir con el nombre del nodo definido en el ejecutable.



```
simulator.launch x
1  <?xml version="1.0" encoding="utf-8"?>
2  <launch>
3  |  <!-- Cargar escenario del robot-->
4  |  <include file="$(find baxter_gazebo)/launch/baxter_world.launch"/>
5  |  <!-- Carga los objetos necesarios para el simulador -->
6  |  <node pkg="simulator" type="simulator.py" name="simulator" />
7  </launch>
```

Figura 6.10: Archivo *simulator.launch*.

Una vez se ejecute *rosrun*, se cargará el entorno de simulación en Gazebo y se arrancarán los nodos ROS que permitirán la comunicación con los robots.

Comportamiento robótico

El comportamiento robótico que se ha desarrollado se basa en el experimento empleado por el GII para realizar diferentes pruebas sobre MotivEn. El experimento se explicará en el Apartado 7.1.

En esta parte del proyecto se utilizó el sistema de comunicación de mensajes de ROS y su librería cliente *rospy* para implementar la comunicación y el comportamiento de los robots implicados. La implementación de estas tareas se desarrolló en la clase *GazeboSimulator*. La comunicación entre *GazeboSimulator* y los modelos de robots cargados en Gazebo se ha realizado empleando el sistema de mensajería entre nodos de ROS. También se ha empleado la API *baxter_interface* en Python, la cual proporciona una interfaz de alto nivel para comunicarse con el modelo virtual del Baxter y facilitar diferentes tareas como por ejemplo, el movimiento de los brazos del robot o agarrar un objeto. Para realizar dicha comunicación fue necesario definir un nodo ROS, que leyese (subscribiese) y enviase (publicase) información a determinados topics. El motivo de emplear esta librería se debe a que uno de los robots implicados en el experimento final era el robot industrial Baxter.

Como se mencionó previamente, el modelo de robot móvil desarrollado disponía de un plugin para controlar la potencia aplicada a las ruedas. Si nos fijamos de nuevo en la Figura 6.9, se puede ver que posee dos tags, *odometryTopic* y un *commandTopic* respectivamente. Estos tags indican, en el primer caso, el topic en el que se publicará la posición del robot; y en el segundo, el topic empleado para obtener la potencia que se aplicaran a las ruedas. Además, fue necesario el topic */gazebo/model_states*, ya que permitía conocer la posición de los elementos que participaban en la simulación.

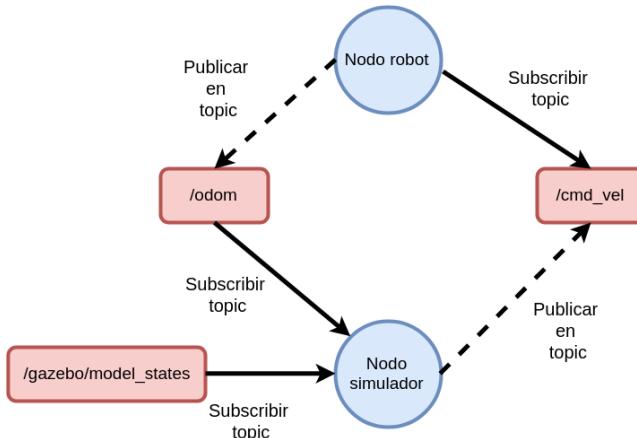


Figura 6.11: Diagrama de la comunicación ROS entre el nodo simulador y el nodo que controla las ruedas del robot móvil.

Se creó entonces un nodo llamado *gazeboExperiment* encargado de subscribir los topics */gazebo/model_states* y */odom*. El primer topic le proporcionaría las posiciones sobre el entorno tanto del objeto que debía agarrar el Baxter como de las coordenadas a las que debía llevarlo. Con el segundo, tendría información sobre la odometría del robot móvil, la cual podía ser utilizada para realizar acciones como la de colocar el objeto de interés sobre su base una

vez se encuentre cercano a él. Además, este nodo debía publicar mensajes con comandos de velocidad en el topic `/cmd_vel`, lo que permitió definir las distintas acciones que podía realizar dicho robot. En la Figura 6.11 se puede ver una representación de este proceso de comunicación, en la cual el nodo del simulador es el nodo `gazeboExperiment`.

En cuanto al Baxter, su API abstrae la comunicación con los distintos topics que controlan las articulaciones del robot, proporcionando métodos que posibilitaron definir los comportamientos deseados.

Una vez realizada la adaptación del comportamiento robótico del experimento, se dio por concluido el desarrollo del módulo *Simulator Module*.

6.6.3 Análisis de la iteración

Las desviaciones producidas se debieron a dos motivos. El primero fue el hecho de no disponer de un modelo de robot móvil que se adecuase al experimento. Esto provocó que se tuviese que crear un modelo desde cero e integrar el plugin que permitiera dotarlo de movimiento. El segundo fue que no se estimó correctamente el tiempo de desarrollar el comportamiento robótico, ya que cada vez que se realizaba una nueva modificación o ajuste en las acciones de los robots era necesario probarla en simulación para cerciorarse de su correcto funcionamiento. También hay que señalar que la inexperiencia con estas tecnologías alargó el tiempo global de la iteración.

6.7. Iteración 6: Desarrollo de la configuración del sistema motivacional

	Modo de tarea	Nombre de tarea	Duración	Duración de línea base	Comienzo	Comienzo de línea base	Fin	Fin de línea base
✓	➡	Iteración 5	23,4 días	18,4 días	mar 09/10/18	mié 12/09/18	vie 09/11/18	jun 08/10/18
✓	➡	Reunión	0 días	0 días	mar 09/10/18	mié 12/09/18	mar 09/10/18	mié 12/09/18
✓	➡	Corrección errores de la Iteración 4	3 horas	5 horas	mar 09/10/18	mié 12/09/18	mar 09/10/18	jue 13/09/18
✓	➡	Estudiar documentación de Gazebo/ROS	25 horas	20 horas	mar 09/10/18	jue 13/09/18	mar 16/10/18	mié 19/09/18
✓	➡	Desarrollo del entorno virtual	6 horas	10 horas	mar 16/10/18	mié 19/09/18	mié 17/10/18	vie 21/09/18
✓	➡	Desarrollar robot móvil	7 horas	5 horas	jue 18/10/18	vie 21/09/18	vie 19/10/18	jun 24/09/18
✓	➡	Integrar plugin en robot móvil	13 horas	5 horas	vie 19/10/18	lun 24/09/18	mar 23/10/18	mar 25/09/18
✓	➡	Integrar modelos virtuales	3 horas	7 horas	mié 24/10/18	mar 25/09/18	mié 24/10/18	mié 26/09/18
✓	➡	Analizar experimento	5 horas	5 horas	mié 24/10/18	jue 27/09/18	jue 25/10/18	jue 27/09/18
✓	➡	Implementar comportamiento robótico	45 horas	25 horas	jue 25/10/18	vie 28/09/18	mié 07/11/18	jue 04/10/18
✓	➡	Integrar entorno de simulación	10 horas	10 horas	mié 07/11/18	vie 05/10/18	vie 09/11/18	jun 08/10/18
✓	➡	Reunión para determinar el correcto	0 días	0 días	vie 09/11/18	lun 08/10/18	vie 09/11/18	jun 08/10/18

Figura 6.12: Tareas realizadas en la Iteración 5.

6.7 Iteración 6: Desarrollo de la configuración del sistema motivacional

6.7.1 Información básica de la iteración

- Fecha inicio: 9 de noviembre del 2018.
- Fecha de finalización: 3 de diciembre del 2018.
- Tareas:
 - Desarrollar la configuración de MotivEn.
 - Desarrollar la configuración para los simuladores empleados.
 - Realizar el experimento.

- Descripción:
 - Para configurar de manera sencilla y flexible el comportamiento del sistema motivacional y de los respectivos simuladores se desarrollaron un par de ficheros que contendrían las configuraciones necesarias. Dichas configuraciones fueron empleadas para evaluar el sistema motivacional en el experimento propuesto.

6.7.2 Resumen

Durante la primera reunión de la iteración, los *investigadores* comentaron las posibles configuraciones que podrían ser empleadas cuando se realizasen las pruebas del experimento. Una vez se obtuviesen los datos del experimento, estos permitirían evaluar el comportamiento de MotivEn con los componentes software integrados en este trabajo.

Configuración de MotivEn

Dado que el sistema motivacional todavía está en fase de desarrollo, establecer un mecanismo que permita definir el comportamiento del mismo, resultaría de gran utilidad a la hora de realizar diferentes tipos de pruebas. Por este motivo, se tomó la decisión de proporcionar a MotivEn una forma de determinar su comportamiento inicial. Se definió en primer lugar el fichero *Configuration.xml*. Este archivo contiene parámetros que permiten configurar, por ejemplo, el tipo de motivación inicial o los modelos de utilidad disponibles. Un parámetro de gran importancia será el que indique el simulador en el que se realizará la simulación. De esta forma, se pueden configurar fácilmente los aspectos más relevantes del sistema. En la Figura 6.13 se muestra una posible configuración para el sistema.

```
<?xml version="1.0"?>
<MDBCoreCofiguration>

    <initial_motivation>Ext</initial_motivation>
    <use_value_function>True</use_value_function>
    <use_surs>False</use_surs>
    <initial_utility_model>SUR</initial_utility_model>
    <simulator_type>gazebo</simulator_type>
    <! --
        type = gazebo
        type = 2d
    -->

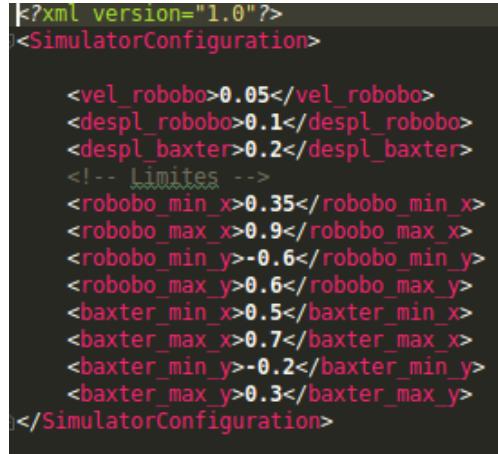
</MDBCoreCofiguration>
```

Figura 6.13: Configuración del fichero *Configuration*.

Configuración del *Simulator Module*

También se definió un fichero configuración que permitiese establecer diversas configuraciones sobre los elementos de los posibles entornos virtuales. En el archivo *SimulatorCon-*

figuration se definirían parámetros como la potencia que se aplicaría a las ruedas del robot móvil o los límites del entorno entre los que los robots realizan su actividad. En la Figura 6.14 se muestra una posible configuración para los simuladores desarrollados.



```
<?xml version="1.0"?>
<SimulatorConfiguration>

    <vel_robobo>0.05</vel_robobo>
    <despl_robobo>0.1</despl_robobo>
    <despl_baxter>0.2</despl_baxter>
    <!-- Límites -->
    <robobo_min_x>0.35</robobo_min_x>
    <robobo_max_x>0.9</robobo_max_x>
    <robobo_min_y>-0.6</robobo_min_y>
    <robobo_max_y>0.6</robobo_max_y>
    <baxter_min_x>0.5</baxter_min_x>
    <baxter_max_x>0.7</baxter_max_x>
    <baxter_min_y>-0.2</baxter_min_y>
    <baxter_max_y>0.3</baxter_max_y>
</SimulatorConfiguration>
```

Figura 6.14: Configuración del fichero *SimulatorConfiguration*.

Una vez finalizada las configuraciones necesarias, se realizaron una serie de pruebas basadas en el experimento del Apartado 7.1. Las pruebas y sus resultados se analizan en el Apartado 7.3.

6.7.3 Análisis de la iteración

La planificación inicial para la fase final del proyecto se desvió significativamente tal como se muestra en la Figura 6.15. Esto se debió a la falta de experiencia con esta clase de pruebas, ya que en la etapa de planificación no se tuvo en cuenta el tiempo que se podría necesitar para obtener los datos necesarios del experimento. Además, a medida que se realizaban diferentes simulaciones se fueron produciendo errores que no habían sido contemplados anteriormente, lo que obligaba a realizar diferentes correcciones que a su vez retrasaban continuamente el progreso de la tarea.

CAPÍTULO 6. DESARROLLO

Modo de tarea	Nombre de tarea	Duración	Duración de línea base	Comienzo	Comienzo de línea base	Fin	Fin de línea base
✓	Iteración 6	16,4 días	7,4 días	vie 09/11/18	lun 08/10/18	lun 03/12/18	jue 18/10/18
✓	Reunión para definir configuraciones	0 días	0 días	vie 09/11/18	lun 08/10/18	vie 09/11/18	lun 08/10/18
✓	Corrección de errores de la Iteración 5	7 horas	5 horas	vie 09/11/18	mar 09/10/18	lun 12/11/18	mar 09/10/18
✓	Definir configuración del sistema	10 horas	7 horas	mar 13/11/18	mié 10/10/18	mié 14/11/18	jue 11/10/18
✓	Realizar experimento robótico	65 horas	25 horas	jue 15/11/18	jue 11/10/18	lun 03/12/18	jue 18/10/18
✓	Reunión para evaluar los resultados	0 días	0 días	lun 03/12/18	jue 18/10/18	lun 03/12/18	jue 18/10/18

Figura 6.15: Tareas realizadas en la Iteración 6.

6.7. Iteración 6: Desarrollo de la configuración del sistema motivacional

Capítulo 7

Experimento y pruebas realizadas

7.1 Experimento

En este apartado se explica el experimento con el que se ha evaluado el comportamiento de MotivEn tras la integración de los respectivos módulos.

7.1.1 Explicación del experimento

El experimento aquí presentado ha sido planteado y analizado previamente por el GII[5]. Este experimento fue realizado inicialmente para tratar de demostrar las capacidades de las SURs para resolver diferentes tipos de tareas en escenarios en los que la presencia de ambigüedad y variabilidad produce inconsistencias respecto a la estimación de utilidad esperada. En este trabajo, el experimento será utilizado para demostrar la eficacia de la VF desarrollada y la integración del simulador Gazebo.

El problema que se plantea en dicho experimento se basa en recoger un objeto determinado de una mesa y colocarlo en una posición predefinida sin poseer ningún conocimiento previo de que éste fuese el objetivo del robot, sino que el sistema debía descubrirlo utilizando su sistema de motivaciones. Para ello, se utilizaron el robot Baxter[7] y el robot Robobo[8]. Ambos robots estaban controlados por una arquitectura cognitiva simplificada de MotivEn que se ejecutaba en el Baxter. En este experimento, los robots no desempeñaban su actividad independientemente, sino que el robot móvil se comportaba como un actuador más del Baxter, es decir, el sistema motivacional controlaba a los dos robots como si fueran uno solo. En el caso en el que el brazo del Baxter fuese incapaz de alcanzar determinadas regiones de la mesa, ambos robots debían colaborar para cumplir el objetivo[5]. El entorno del experimento se puede ver en la Figura 7.1 y estaba formado por una mesa sobre la que se colocaba una cesta marrón y un cilindro amarillo y verde.

El espacio de estados se definió en función de 3 sensores[5]:

- Sensor de distancia entre el cilindro y el robot móvil (dCR).

- Sensor de distancia entre el cilindro y el Baxter (dCB).
- Sensor de distancia entre el cilindro y la cesta (dCX).

Definiendo en cada iteración un estado sensorial:

$$S(t) = (dCR, dCB, dCX) \quad (7.1)$$

Estos sensores serán los que posibilitarán establecer las SURs que se emplearán para poder modelar las Value Functions.

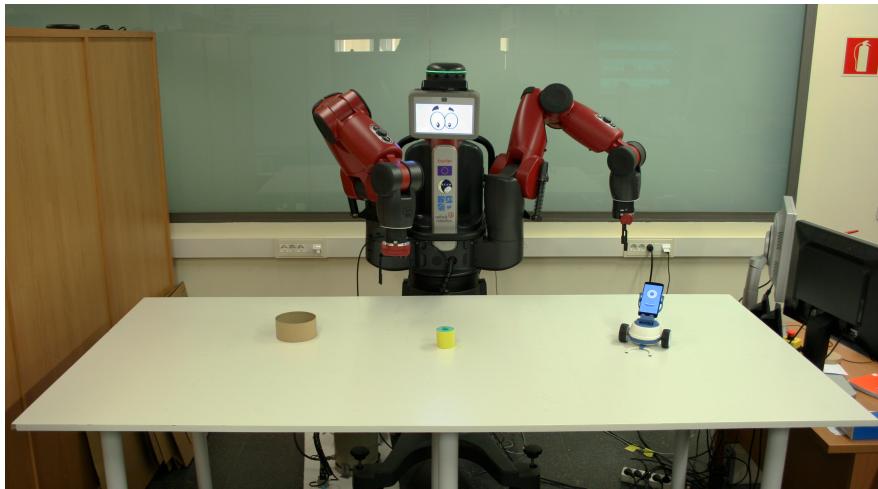


Figura 7.1: Escenario del experimento. En él se pueden ver los robots Baxter y Robobo, el cilindro y la cesta.

El conjunto de acciones $A(t)$ controlaban el movimiento del brazo derecho del Baxter y el movimiento del robot móvil sobre la mesa. La primera acción definía el movimiento en el brazo del Baxter a una altura constante sobre la mesa y a una velocidad fija. La segunda acción modificaba la dirección del robot móvil permitiéndole moverse libremente sobre la mesa a una velocidad constante. Existían algunas acciones predefinidas: si alguno de los robots llegaba al cilindro, lo recogía automáticamente. Del mismo modo, si el robot Baxter alcanzaba la cesta mientras transportaba el cilindro, ejecutaría una respuesta predefinida para dejarlo en la cesta[5].

En este experimento se definieron 2 estados objetivo:

- El Baxter logra llevar el cilindro a la cesta.
- El robot móvil lleva el cilindro a la cesta.

Tras llegar a alguno de estos objetivos, el sistema recibía una recompensa y se reiniciaba el escenario, situando cada uno de los objetos en una posición aleatoria. La cesta se situaba

siempre en una posición accesible para el Baxter, de modo que ambos robots pudiesen llegar hasta ella[5].

Desde la perspectiva de MotivEn, el sistema desconocía tanto los objetivos como la secuencia de acciones a realizar para alcanzarlos. Esta situación propiciaba un movimiento exploratorio (motivación intrínseca) con el que se identificaban los distintos objetivos y se asociaban a puntos del espacio de estados. Una vez realizado este paso, se definía una ruta que guiase al sistema hasta un estado de utilidad, lo que lo obligaba a aprender un modelo de utilidad.

7.1.2 Adaptación del experimento en Gazebo

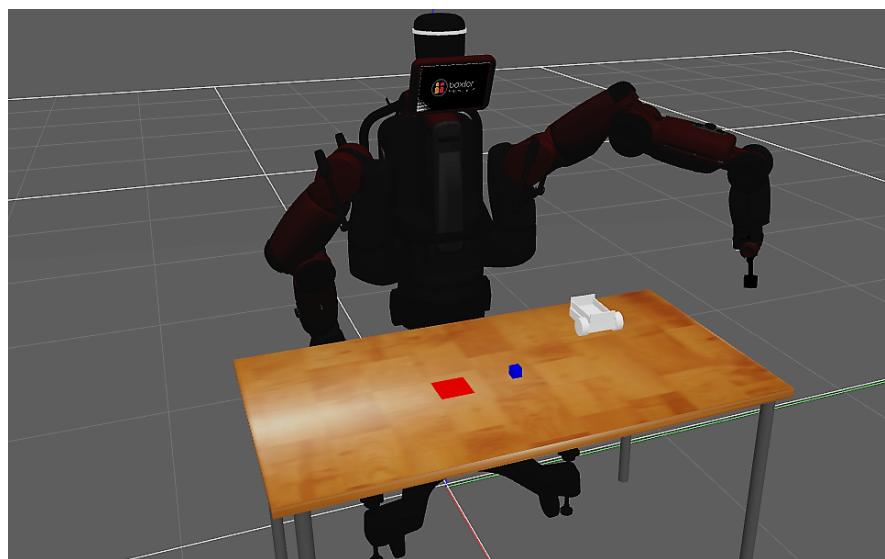


Figura 7.2: Escenario del experimento en el simulador Gazebo. En él se pueden ver al robot Baxter, al robot móvil (color blanco), el cubo (color azul) y la zona objetivo (color rojo).

El experimento que se realizó tras integrar los respectivos módulos software mantiene el comportamiento original, sin embargo, los objetivos establecidos inicialmente fueron modificados. Por simplicidad, solo se abordó que el Baxter fuese el responsable de llevar el objeto al punto predefinido (*goal*), eliminando la opción de que también lo entregase al otro robot.

También se realizaron cambios sobre el escenario del experimento y los robots utilizados. En primer lugar, el Robobo fue substituido por un modelo de robot diseñado específicamente para el simulador, tal como se explica en la Iteración 6.6. Este cambio se debe a que Gazebo no disponía de un modelo 3D de Robobo. El cilindro y la cesta fueron substituidos por un cubo y un área rectangular plana respectivamente. En la Figura 7.2 se puede ver el entorno resultante.

El espacio de estados y las acciones que guiaban el movimiento de los robots permanecieron inalteradas. Sin embargo, para agilizar la simulación y evitar determinados problemas, se

optó por colocar el cubo en la base superior del robot móvil cuando estos estuviesen próximos. Esta alternativa no supuso ninguna alteración significativa con respecto al experimento original. El funcionamiento global de la arquitectura cognitiva continuó siendo el mismo con la excepción de que, en este caso, se hizo uso del módulo *Value Functions Module* y se empleó la VF desarrollada en este trabajo.

En el Apartado 7.3 se analizarán las pruebas realizadas y resultados obtenidos a partir de este experimento tras integrar los módulos software desarrollados.

7.2 Pruebas realizadas sobre la VF

De cara a probar el funcionamiento de la VF desarrollada, tal y como se menciona en la Iteración 6.3, se realizaron un conjunto de pruebas qué sirvieron, además, para establecer la configuración más idónea para el experimento a resolver. Dichas pruebas se realizaron utilizando la validación cruzada (cross-validation), la cual permite evaluar el comportamiento de diferentes modelos matemáticos y comparar sus resultados de manera fiable.

7.2.1 Validación cruzada

La validación cruzada permite conocer la capacidad de generalización de un modelo empleando un conjunto de datos. En cada iteración, los datos disponibles se dividen normalmente en dos conjuntos: entrenamiento y test. Los primeros se utilizan para capacitar el modelo mientras que los segundos permiten conocer su precisión. Para reducir la varianza y garantizar la independencia entre el modelo y los datos, se realizan varias rondas de validación cruzada utilizando diferentes conjuntos de entrenamiento y test. Los resultados de todas las rondas se promedian para estimar la precisión del modelo [30]. En este trabajo, la validación cruzada permitirá escoger una serie de métricas y utilizarlas para comparar las diferentes configuraciones de la VF con el objetivo de tener una de idea de las mejores topologías y parámetros a emplear en el experimento.

Existen diferentes variaciones de esta técnica en función de las particiones de datos empleadas, sin embargo, solo se comentará la validación cruzada en K iteraciones o *K-fold cross validation*[30], ya que es la que se ha empleado en este trabajo. En este método, el conjunto de datos inicial se divide en K subconjuntos del mismo tamaño y se realizan K iteraciones empleando $K - 1$ subconjuntos como datos de entrenamiento y un subconjunto para validar el modelo. La estimación del error E se realiza de la siguiente forma:

$$E = \frac{1}{K} \sum_{i=1}^K E_i$$

donde E_i es el error sobre el conjunto de test en la iteración i . Esto reduce significativamente

la varianza ya que todos los datos serán utilizados. El valor de K puede ser el que se desee, aunque generalmente se suele emplear $K = 5$ o $K = 10$.

7.2.2 Parámetros evaluados

Se empleó el método *K-fold cross validation* para evaluar la VF implementada con $K = 10$ y se realizó la validación de la ANN analizando la importancia de los siguientes parámetros:

- **Topología de la red:** se probaron diferentes números de capas y neuronas ocultas para determinar la topología que mejor se ajusta a este tipo de experimento.
- **Épocas de entrenamiento:** número de iteraciones que la red empleará para aprender los datos del entrenamiento. Se estudió el error de entrenamiento empleando valores comprendidos entre 20 y 100 épocas en el caso del Adam.
- **Algoritmo de aprendizaje:** se emplearon los algoritmos de descenso del gradiente y el Adam, explicados en el Apartado 2.2.

La razón por la que se comenzó empleando el descenso del gradiente, y tras analizar sus resultados se utilizó el algoritmo Adam, se debe a que el descenso del gradiente proporciona una medida fiable del error mínimo ideal que se puede conseguir con un modelo concreto. Por lo tanto, se realizaron una serie de pruebas iniciales para determinar qué configuraciones se ajustaban mejor al problema a resolver y así agilizar las pruebas posteriores con el Adam.

Métricas y gráficas empleadas

Para evaluar el comportamiento y el impacto de los parámetros comentados se emplearon las siguientes métricas y gráficas:

- **Error más alto de entrenamiento:** se empleó dicho valor, además del promedio del error de test entre las K iteraciones, ya que al ser el valor más pesimista del entrenamiento puede ser más representativo a la hora de seleccionar un modelo u otro. Es un número real.
- **Error de test de la peor iteración:** error sobre el conjunto de test en la iteración con el peor error de entrenamiento. Es un número real.
- **Valor medio del error de test en las K iteraciones:** este valor proporcionará una idea de la capacidad de predicción de la red, es decir, permite evaluar la precisión del modelo.
- **Desviación estándar en el error de test:** muestra la dispersión del error en la validación cruzada. Es un número real.

- **Tiempo de computación medio en las K iteraciones:** se muestra el tiempo medio de cómputo que necesita el modelo para aprender el conjunto de entrenamiento. Este valor dependerá de la topología, número de épocas empleadas y velocidad de aprendizaje utilizada, por lo que será determinante a la hora de seleccionar la configuración de la red, ya que en MotivEn, el aprendizaje de la VF debería ser lo más rápido posible de forma que esta pueda ser utilizada y actualizada en tiempo real. Se medirá en segundos.
- **Gráfica con los errores de la validación cruzada:** se muestra el error de entrenamiento y el error sobre el conjunto de test en cada iteración.
- **Gráfica con el tiempo de cómputo:** se muestra el tiempo que necesita la red para aprender los datos de entrenamiento en cada una de las iteraciones.
- **Gráfica de la iteración con el peor error en el entrenamiento:** muestra el peor error de entrenamiento.
- **Gráfica de las predicciones realizadas empleando la red con el peor error en el entrenamiento:** se muestran los valores de utilidad reales frente los predichos por la red.

Datos empleados

Los datos que se emplearon estaban compuestos por un conjunto trazas recopiladas previamente en el experimento realizado por el GII y comentado en el Apartado 7.1. Los estados de dichas trazas estaban compuestos por los valores de los sensores (dCR , dCB , dCX). En la Figura 7.3 se muestran algunos de los datos empleados y su respectiva utilidad. Como se puede apreciar, la utilidad fue asignada de manera descendente en función de la antigüedad del estado sensorial. Los datos de las columnas Sensor1, Sensor2 y Sensor3 se corresponden con los valores de los sensores dCR , dCB y dCX respectivamente. De esta forma, un valor de distancia igual a cero en alguna de las dos primeras columnas significa que uno de los respectivos robots ha recogido el bloque, mientras que en la última columna, significa que bloque está sobre el *goal*. Cada traza (secuencia de n estados) representa un ejemplo con el que se puede entrenar o evaluar la red. El número de trazas total eran 42, por lo que en cada iteración se emplearon 37 trazas para entrenar el modelo y 5 trazas para evaluar su precisión, todas seleccionadas de forma aleatoria.

7.2.3 10-fold cross validation empleando el algoritmo de descenso del gradiente

El descenso del gradiente permitirá acotar las configuraciones de redes a analizar con el algoritmo Adam, y que servirán para determinar el modelo de la VF a emplear en el experi-

Sensor1	Sensor2	Sensor3	Utility
991,7303928	201,053653001	185,951606608	0,015625
967,11354292	175,875344622	185,951606608	0,03125
941,57953212	152,81015132	185,951606608	0,046875
916,37308003	126,938307426	185,951606608	0,0625
893,78906683	102,187499143	185,951606608	0,078125
867,78429459	76,6798590132	185,951606608	0,09375
844,37987333	56,9424048673	185,951606608	0,109375
812,52352829	0	219,252780395	0,125
762,68642569	0	183,019803808	0,140625
698,46709713	0	203,196596262	0,15625
619,16311229	0	249,685026766	0,171875
570,00600211	0	271,031717613	0,1875
526,68121363	0	304,942154852	0,203125
480,58733212	0	327,98288498	0,21875
424,60861155	0	338,102119196	0,234375
376,68681012	0	363,224115919	0,25

Figura 7.3: Datos empleados en la validación cruzada.

mento. Además, el descenso del gradiente proporcionará una idea más clara del nivel de error ideal al que se podrá llegar y que resultará útil para fijar el número de épocas en las pruebas con el algoritmo Adam.

Aunque los conjuntos de entrenamiento y test fueron generados de manera aleatoria, las trazas empleadas en el entrenamiento fueron presentadas a la red ordenadas en función de la antigüedad de las mismas, empleando en primer lugar aquellas que se recopilaron al inicio de las SURs, ya que serán más ambiguas que las recopiladas cuando las regiones tenían cierta certidumbre. Al realizar la validación cruzada de esta forma el aprendizaje resulta más similar al que se realizaría durante la simulación, lo que puede proporcionar una mayor certidumbre sobre la configuración a emplear en el robot real. Sin embargo, hay que señalar que en este caso, los $K-1$ subconjuntos serán presentados a la red como un único conjunto de datos, pues el algoritmo del descenso del gradiente emplea toda la información disponible para entrenar y actualizar el modelo. En el caso del Adam, cada traza será aprendida de forma individual, simulando la recopilación de información en tiempo real.

Los modelos probados, en este caso y en las pruebas realizadas con el Adam, estaban formados por una capa de entrada compuesta por 3 neuronas, una por cada sensor empleado, y una capa de salida formada por una neurona, la utilidad esperada. En las pruebas realizadas se analizaron varias topologías de red compuestas por una, dos o tres capas ocultas, donde cada una de las cuales contaba con un número determinado de neuronas (3, 10 o 20). Aunque se podrían haber probado múltiples combinaciones, lo que interesaba en esta parte del trabajo era obtener una serie de modelos iniciales que establecieran la base de cara a evaluar el comportamiento del Adam.

Las pruebas se realizaron empleando una tasa de aprendizaje $\eta = 0.001$. La velocidad de aprendizaje no era un factor tan representativo, ya que el Adam lo ajusta automáticamente en función de los parámetros de la red. Por lo tanto, no se realizó una evaluación muy exhaustiva

sobre su importancia en las configuraciones de las VFs.

En cuanto a las épocas, no se empleó ningún valor fijo sino que se decidió detener el entrenamiento una vez el error se estabilizara. De esta forma, se evitaría el sobreajuste de la red. Para ello, se empleó el *callback EarlyStopping*¹ de Keras.

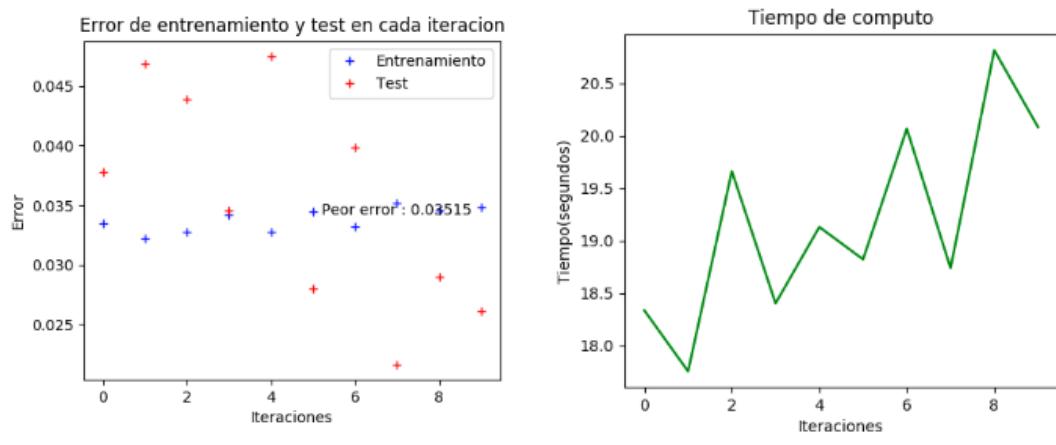
Validación cruzada						
Topología	Épocas	Error de entrena- miento en la peor iteración	Error de test en la peor ite- ración	Error medio de test	Desviación estándar	Tiempo medio de entrena- miento
3-10-1	4050	0.03540	0.02333	0.03625	0.00741	6.5363
3-20-1	3426	0.03530	0.02395	0.03609	0.00753	6.8301
3-10-3-1	5535	0.03570	0.02114	0.03597	0.00854	8.7175
3-10-10-1	4479	0.03552	0.02172	0.03571	0.00813	7.2805
3-10-3-3-1	12988	0.03515	0.02163	0.03552	0.00865	19.1812
3-10-10-3-1	9970	0.03512	0.02175	0.03554	0.00877	16.4057
3-10-10-10-1	8177	0.03512	0.02176	0.03561	0.00872	14.1494
3-10-10-20-1	31	0.34342	0.34342	0.34342	6E-05	0.5010
3-10-20-3-1	7919	0.03514	0.02170	0.03559	0.00862	13.8817
3-20-10-3-1	9424	0.03513	0.02190	0.03578	0.00844	19.4632
3-3-10-20-1	21	0.34342	0.34342	0.34342	6E-05	0.5220
3-20-10-10-1	8378	0.03511	0.02180	0.03567	0.00868	17.1867

Cuadro 7.1: Tabla de las configuraciones y pruebas realizadas con el descenso del gradiente.

En la Tabla 7.1 se muestran los resultados obtenidos. Se han resaltado en amarillo los mejores valores de error en las respectivas columnas. Los resultados de la octava y undécima configuración muestran que dichas topologías no son capaces de aprender los datos de entrenamiento, ya que sus valores de error son altos y no varían en todas las iteraciones, por lo tanto, fueron descartadas en las pruebas posteriores. Por otro lado, parece que la gran mayoría de configuraciones, con excepción de la octava y undécima fila, ofrecen resultados similares con un promedio del error de test bastante bajo independientemente de que sean la mejor configuración o no. Esto parece indicar que casi todos los modelos ofrecen una respuesta similar al problema en cuestión, lo que no ofrece más información que el hecho de que el aprendizaje clásico podría ser una opción válida para obtener VFs a partir de las SURs, algo que los *investigadores* ya conocían. Por otro lado, los valores obtenidos deberían permitir analizar

¹https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

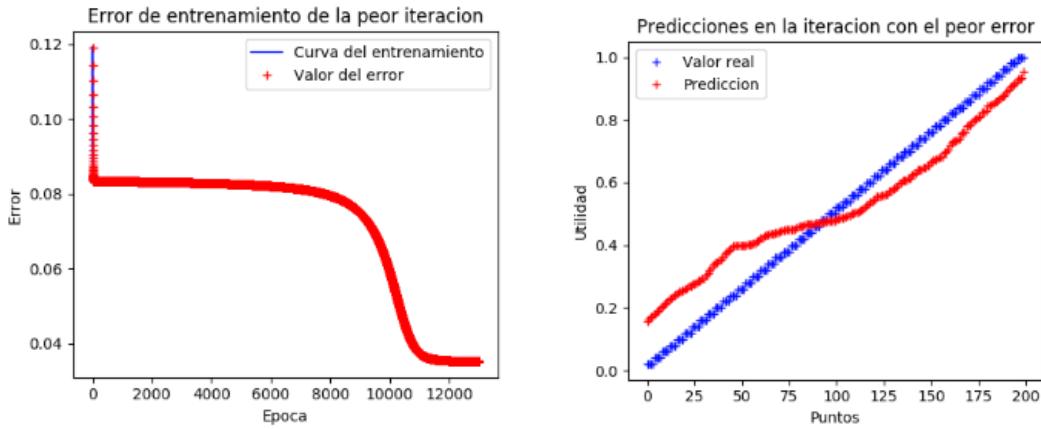
el comportamiento del Adam, ya que las configuraciones a examinar deberían conseguir un valor de error medio bastante similar. Para finalizar, se puede observar que el principal inconveniente de aplicar este paradigma de aprendizaje es el tiempo de entrenamiento requerido para actualizar el modelo, lo que es un inconveniente en un sistema como MotivEn. En las Figuras 7.4 y 7.5 se muestran las gráficas para la configuración con el mejor error medio en la validación cruzada. Esta configuración presenta un error en la validación cruzada ligeramente mejor que las demás configuraciones al mismo tiempo que posee un error de entrenamiento próximo al mejor valor obtenido. Por lo tanto, podría ser un buen candidato para la VF dependiendo de los resultados obtenidos con el Adam.



(a) Errores en cada iteración.

(b) Tiempo de entrenamiento en cada iteración.

Figura 7.4: Gráficas de los errores y del tiempo de entrenamiento.



(a) Error de entrenamiento en la peor iteración.

(b) Predicciones de la peor iteración.

Figura 7.5: Gráficas de la red con el peor error de entrenamiento y sus predicciones.

7.2.4 10-fold cross validation empleando el algoritmo Adam

Los resultados obtenidos empleando el descenso del gradiente sirvieron para establecer las topologías de red a evaluar empleando el algoritmo Adam. Los resultados de estas pruebas permitirán establecer la VF que se empleará en la tarea a realizar en el experimento.

Además de las diferentes topologías a analizar, también se probaron una serie de valores para las épocas de entrenamiento. Esto se debe a que emplear un método que detenga el entrenamiento en este tipo de algoritmos no resulta trivial, ya que el error oscila bruscamente, lo que puede provocar que no consiga converger a un mínimo. En este caso, el número de épocas serán las iteraciones que emplea la red para entrenar cada traza de forma individual, por lo que este valor influirá bastante sobre el tiempo que necesitará el modelo para actualizar la VF. Se intentará buscar el número de iteraciones con las que la VF proporciona una predicción precisa y es entrenada con mayor rapidez. Inicialmente se emplearon los siguientes valores: 20, 50 y 100 épocas. De esta forma, se puede evaluar el comportamiento de cada modelo empleando diferentes tiempos de entrenamiento y analizar su influencia en el rendimiento.

En este caso, no se ha modificado la tasa de aprendizaje y se han empleado los valores establecidos por los autores del algoritmo, siendo estos $\eta = 0.001$, $\beta_1 = 0.9$ y $\beta_2 = 0.999$.

Las topologías y épocas empleadas junto con sus respectivos resultados se muestran en la Tabla 7.2. Como se puede apreciar, el error de test medio en la validación cruzada disminuye, de manera general, en las redes con 3 capas ocultas. En los valores de error resaltados en amarillo en la Tabla 7.2, se puede ver que la configuración con mejor error medio es la misma que en las pruebas realizadas con el descenso del gradiente. Sin embargo, los modelos no suelen mejorar el error de test cuando emplean 100 épocas de entrenamiento, de hecho, incluso lo empeoran. En lo que respecta al peor error del entrenamiento, con 100 épocas si se logra re-

ducir dicho error, pero tampoco significativamente si se compara con el error obtenido tras 50 épocas. Esto puede deberse al comportamiento del Adam, el cual ajusta individualmente cada parámetro de la red basándose en la varianza y el promedio de los gradientes anteriores de la función error. Por lo tanto, al aumentar el número de épocas, dichos parámetros presentan diferencias menos significativas, lo que provoca que el ajuste de los parámetros sea menor[9]. Sin embargo, cuando se le presenta a la red una traza con estados sensoriales desconocidos, esta sufre variaciones bruscas en la función de error lo que lleva al Adam a realizar ajustes mayores en los parámetros. Esto se puede observar en la Figura 2.6. Como se puede observar, la función de error oscila bruscamente cada vez que se le presenta una nuevo ejemplo a la red. Por lo tanto, parece que los mejores valores de error se logran empleando alrededor de 50 épocas de entrenamiento. Si comparamos el error obtenido con el Adam y los valores obtenidos con el descenso del gradiente clásico, se puede apreciar que tanto el error de test como el error medio en la validación cruzada son bastante parejos, mientras que el error de entrenamiento es mucho más bajo en el caso de Adam. Esto se debe a que este algoritmo realiza las actualizaciones de forma individual sobre cada traza lo que hace que el error se ajuste de forma más rápida y brusca, mientras que el descenso del gradiente hace que la actualización sea más lenta y progresiva. La red con el mejor error de entrenamiento es una de las que emplea 100 épocas, siendo la única en la que se produce una variación significativa en dicho ajuste respecto a la configuración que emplea 50, lo que se seguramente está condicionado por la topología de la red. El factor diferenciador aquí es el tiempo de cómputo. Como se puede ver, el Adam logra mejorar la capacidad de predicción obtenida con el descenso del gradiente empleando mucho menos tiempo, de ahí que se pensase que el aprendizaje online podría representar una alternativa válida dentro de los modelos de utilidad de MotivEn.

7.2. Pruebas realizadas sobre la VF

Validación cruzada						
Topología	Épocas	Error de entrena- miento en la peor iteración	Error de test en la peor ite- ración	Error medio de test	Desviación estándar	Tiempo medio de entrena- miento
3-10-3-1	20	0.0121	0.0200	0.0346	0.0120	2.1490
3-10-3-1	50	0.0052	0.0219	0.0342	0.0113	4.3960
3-10-3-1	100	0.0029	0.0372	0.0396	0.0074	6.5674
3-10-10-1	20	0.0073	0.0284	0.0476	0.0293	2.0130
3-10-10-1	50	0.0050	0.0213	0.0340	0.0010	4.8275
3-10-10-1	100	0.0043	0.0408	0.0462	0.0158	6,9044
3-10-3-3-1	20	0.0082	0.0233	0.0315	0.0112	2.7857
3-10-3-3-1	50	0.0048	0.0208	0.0305	0.0147	5.7652
3-10-3-3-1	100	0.0042	0.0214	0.0309	0.0086	9.0405
3-10-10-3-1	20	0.0079	0.0277	0.0328	0.0112	2.4134
3-10-10-3-1	50	0.0048	0.0205	0.0309	0.0093	5.7013
3-10-10-3-1	100	0.0043	0.0209	0.0334	0.0102	10.4595
3-10-10-10-1	20	0.0064	0.0227	0.0383	0.0188	2.5118
3-10-10-10-1	50	0.0042	0.0278	0.0392	0.0165	5.5761
3-10-10-10-1	100	0.0045	0.0204	0.0399	0.0209	8.8642
3-10-20-3-1	20	0.0138	0.0192	0.0368	0.0101	3.0329
3-10-20-3-1	50	0.0048	0.0210	0.0327	0.0130	6.4107
3-10-20-3-1	100	0.0044	0.0207	0.0323	0.0129	9.3359
3-20-10-3-1	20	0.0854	0.0854	0.0851	0.0001	2.3126
3-20-10-3-1	50	0.0833	0.0833	0.0833	3.76E-5	5.0121
3-20-10-3-1	100	0.0833	0.0833	0.0833	3.76E-5	8.3354
3-20-10-10-1	20	0.0068	0.0217	0.0404	0.0206	2.1830
3-20-10-10-1	50	0.0042	0.0256	0.0327	0.0148	5.2117
3-20-10-10-1	100	0.0046	0.0216	0.0331	0.0107	8.4198

Cuadro 7.2: Tabla de las configuraciones y pruebas realizadas empleando el Adam.

Validación cruzada						
Topología	Épocas	Error de entrena- miento en la peor iteración	Error de test en la peor ite- ración	Error medio de test	Desviación estándar	Tiempo medio de entrena- miento
3-10-3-3-1	50	0.0048	0.0208	0.0305	0.0147	5.7652
3-10-10-3-1	50	0.0048	0.0205	0.0309	0.0093	5.7013
3-10-3-3-1	100	0.0042	0.0214	0.0309	0.0086	9.0405
3-10-3-3-1	20	0.0082	0.0233	0.0315	0.0112	2.7857
3-10-20-3-1	50	0.0048	0.0210	0.0327	0.0130	6.4107
3-10-10-3-1	20	0.0079	0.0277	0.0328	0.0112	2.4134
3-10-10-3-1	100	0.0043	0.0209	0.0334	0.0102	10.4595
3-10-3-1	50	0.0052	0.0219	0.0342	0.0113	4.3960

Cuadro 7.3: Tabla de las mejores configuraciones ordenadas a partir de su error medio.

Entre los modelos empleados y sus respectivos resultados, se destacan las configuraciones de la Tabla 7.3. Se puede observar que, en la mayor parte de los casos, se obtiene un error medio inferior empleando 50 épocas. El valor de error en entrenamiento se muestra estable en la casi todas las redes, con las excepciones de los modelos entrenados con 20 épocas, lo cual es normal ya que la red no fue capaz de ajustar tan bien los parámetros al emplear menos tiempo de entrenamiento. Por otro lado, el peor error con 50 y 100 épocas se estabiliza en valores próximos, lo que demuestra que los parámetros de la red sufren modificaciones menos relevantes partir de un número determinado de épocas. Además, los valores del error medio y de la dispersión son bastantes similares lo que indica que inicialmente estos modelos podrían tener un comportamiento similar durante el experimento. En este caso, se utilizó el tiempo medio de entrenamiento como un valor diferenciador para seleccionar un modelo u otro. De esta forma, se seleccionaron las redes que mejor predicciones realizaron empleando el menor tiempo de entrenamiento.

Finalmente, para seleccionar la configuración que se emplearía en el experimento final, se evaluaron las topologías resaltadas en amarillo de la Tabla 7.3, empleando un número de épocas comprendido entre 30 y 60. Se seleccionaron estos valores para evaluar si los modelos lograban mejores predicciones sin tener que incrementar el tiempo de entrenamiento. Los resultados obtenidos están en la Tabla 7.4.

Validación cruzada						
Topología	Épocas	Error de entrena- miento en la peor iteración	Error de test en la peor ite- ración	Error medio de test	Desviación estándar	Tiempo medio de entrena- miento
3-10-3-1	30	0.0059	0.0254	0.0307	0.0098	2.8925
3-10-3-1	40	0.0049	0.0207	0.0332	0.0114	4.3204
3-10-3-1	50	0.0052	0.0219	0.0342	0.0113	4.3960
3-10-3-1	60	0.0046	0.0208	0.0355	0.0105	4.7763
3-10-3-3-1	30	0.0050	0.0219	0.0272	0.0109	3.1600
3-10-3-3-1	40	0.0056	0.0225	0.0290	0.0110	5.2762
3-10-3-3-1	50	0.0048	0.0208	0.0305	0.0147	5.7652
3-10-3-3-1	60	0.0044	0.0206	0.0282	0.0092	6.2342
3-10-10-3-1	30	0.0052	0.0220	0.0315	0.0098	3.8771
3-10-10-3-1	40	0.0088	0.0231	0.0431	0.0253	5.0068
3-10-10-3-1	50	0.0048	0.0205	0.0309	0.0093	5.7013
3-10-10-3-1	60	0.0048	0.0219	0.0288	0.0086	6.0962

Cuadro 7.4: Configuraciones candidatas para establecer la VF.

Las Figuras 7.6 y 7.7 muestran las gráficas correspondientes a la red con 3 capas ocultas de 10, 3 y 3 neuronas respectivamente, en concreto, el modelo que emplea 30 épocas para entrenar la red . El motivo de emplear esta configuración en las pruebas finales, se debe a que los valores de sus métricas son un poco mejores con respecto al resto de configuraciones con una topología distinta. Respecto a las configuraciones con su misma topología, sus valores son próximos a la red que emplea 60 épocas, la cual consigue el mejor error de entrenamiento. Sin embargo, el hecho de que fuese la configuración que emplea menos tiempo de entrenamiento ha sido determinante para su elección, ya que permitirá actualizar el modelo de forma más rápida. Aunque las demás configuraciones se podrían haber empleado como VFs del sistema, se decidió no analizar su comportamiento ya que esta decisión podía alargar demasiado la recopilación y análisis de los resultados. Además, existía la posibilidad de obtener resultados similares, un hecho que ya se había demostrado con los resultados de la validación cruzada.

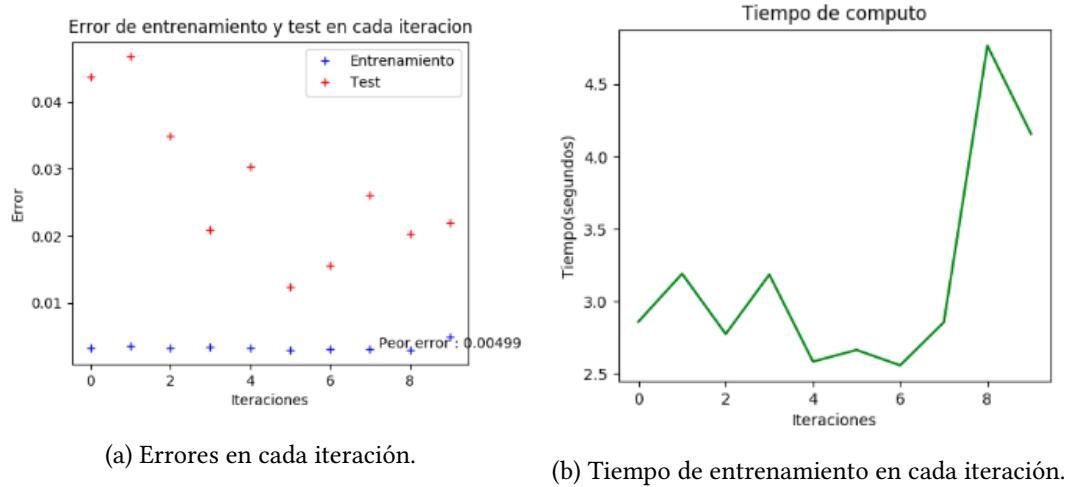


Figura 7.6: Gráficas del error y tiempo de entrenamiento empleando el Adam.

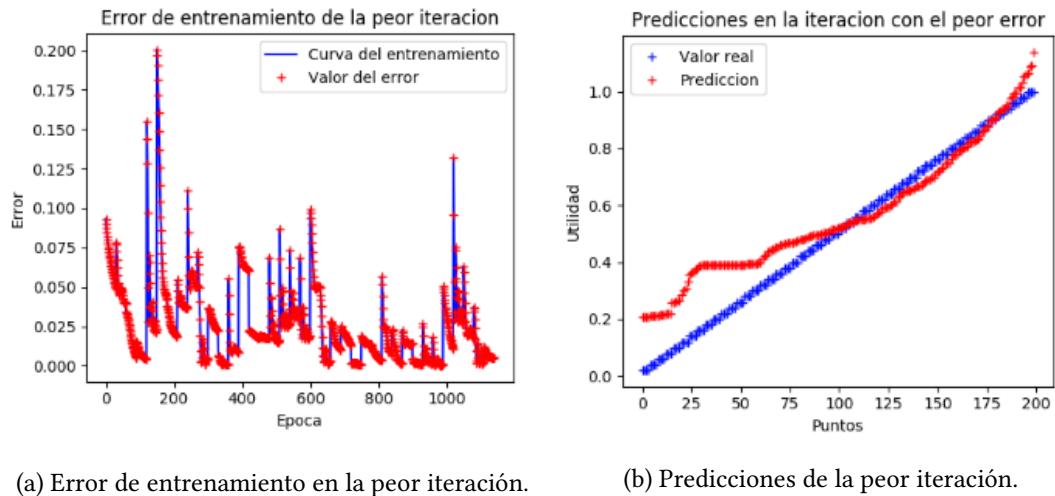


Figura 7.7: Gráficas de la red con el peor error de entrenamiento y sus predicciones empleando el Adam.

En conclusión, la red que se empleó para realizar el experimento final fue la que poseía una topología formada por 3 capas ocultas de 10, 3 y 3 neuronas respectivamente y empleaba 30 épocas de entrenamiento (configuración resaltada en amarillo en la Tabla 7.4).

7.3 Experimento realizado en el simulador Gazebo

Finalizada la integración de los respectivos módulos (módulo *Simulator Module* y módulo *Value Functions Module*) se ejecutó el experimento explicado en la Sección 7.1 para evaluar el funcionamiento de MotivEn en el entorno de simulación de Gazebo empleando la VF desarrollada en este trabajo.

El análisis del experimento permitiría evaluar la mejora del “reality gap” respecto al simulador 2D original, el cual era uno de los objetivos establecidos en el trabajo. El módulo *Simulator Module* y el paquete de ROS desarrollado permitirán plantear y analizar experimentos futuros en un escenario más cercano a la realidad. Además, posibilitará observar el comportamiento de la función de valor implementada y emplear los resultados obtenidos para plantear mejoras o nuevas tareas a resolver con MotivEn. Para ello, se ejecutó el experimento empleando las siguientes configuraciones:

- En el primer caso, MotivEn inició su ejecución usando la VF desarrollada y entrenada con los datos de las SURs.
- En el segundo caso, la VF online fue entrenada a medida que el robot exploraba su entorno y descubría el objetivo del experimento.

7.3.1 Experimento entrenando la VF a partir de las SURs

Las primeras pruebas del experimento se realizaron empleando la VF desarrollada pero entrenada de forma offline a partir de los datos de las SURs. Como ya se comentó, cada SUR emplea la tendencia de su respectivo sensor para proporcionar la dirección en la que la distancia entre el robot y el objetivo disminuye y por lo tanto, aumenta la utilidad a largo plazo. Una vez se realizan un número suficiente de iteraciones, la SUR es capaz de proporcionar un valor de utilidad en el área en la que se activa la tendencia del respectivo sensor, por lo tanto, cuantas más iteraciones se realicen menos ambigua será el área de la SUR y en consecuencia puede emplearse en el aprendizaje de una VF. Para no alargar el tiempo de simulación, se entrenó la VF desarrollada de forma offline con los datos de las SURs utilizados en la validación cruzada. Las trazas que conformaban las SURs habían sido recopiladas por el GII a partir de la simulación del experimento en un entorno real. De esta forma, se simulaba la generación previa de las SURs, el cual es un paso anterior al aprendizaje de VFs. Al realizar el aprendizaje de esta forma, se simulaba el comportamiento real que debería tener el sistema motivacional pero, en este caso, saltándose la etapa de modelado de SURs previa, lo que acortaba la simulación en gran medida. Esto permitió evaluar el sistema con una VF entrenada de forma “online” a partir de las SURs.

El MotivEn comenzó utilizando la motivación extrínseca. Esto provocó que los estados

sensoriales candidatos del robot fuesen evaluados empleando el modelo de utilidad disponible, concretamente, la *DeepAnnVF*, ya que no existía ningún otro. En cada iteración, la VF predijo la utilidad esperada para cada estado candidato del robot. Dicha utilidad fue empleada por el sistema para seleccionar la acción que aumentara la utilidad a largo plazo. Una vez el robot lograba el objetivo, se reinició el experimento situando el bloque y el *goal* (coordenadas en las que el brazo del Baxter debía colocar el bloque) en una posición aleatoria sobre la mesa. De este modo, se analizó el comportamiento de la VF en un entorno en el que la posición de los elementos era dinámica y desconocida por los robots. Además, se configuró la zona en la que los robots podían realizar sus acciones. El área por la que se desplazó el brazo del Baxter era una región menor que el tamaño de la mesa, mientras que robot móvil podía desplazarse por toda la superficie de la misma. Esta configuración también se empleó en las pruebas realizadas con la VF sin entrenar. En la Figura 7.8 se muestran las áreas de la mesa por las que se desplazarán los robots. Esto fue útil para comprobar si la VF lograba mantener el comportamiento del experimento, es decir, si lograba que el robot móvil le llevase la pelota al Baxter cuando este no pudiese cogerla. En las Figuras 7.9 y 7.10 se muestra una de las pruebas en las que sucedió este caso. La primera imagen muestra una posible disposición de los elementos del entorno. Posteriormente, ambos robots, motivados extrínsecamente, se desplazarían hasta la posición del bloque. Dado que el Baxter no alcanzaba a cogerlo, el robot móvil lo recogió y se lo acercó. Finalmente el Baxter recogió el bloque y lo llevó hasta el *goal*.

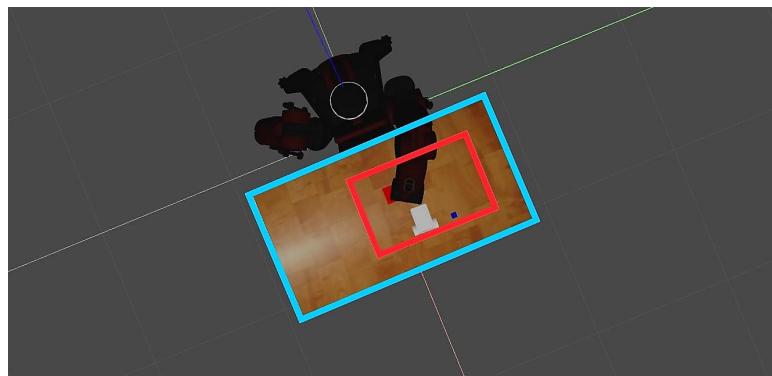
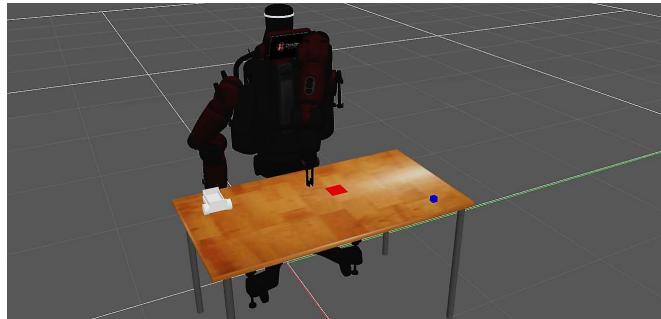


Figura 7.8: Áreas de la mesa en la que el Baxter (región roja) y robot móvil (región azul) desempeñan su actividad.

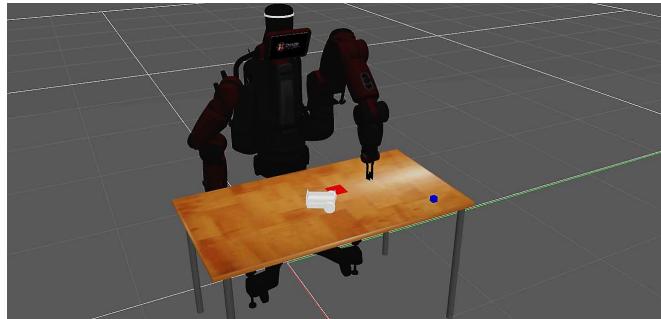
Para evaluar el comportamiento de la VF se realizaron 5 pruebas. En cada una de ellas, se almacenó el número de iteraciones necesarias para resolver la tarea 10 veces seguidas. De este modo, se obtendría la certeza de que la VF era capaz de predecir la mejor acción para el sistema, y, al mismo tiempo, se recopilaba la información necesaria para evaluar el comportamiento del sistema motivacional empleando dicha configuración.

La VF empleada en ambas configuraciones (a partir de las SURs y entrenada desde cero),

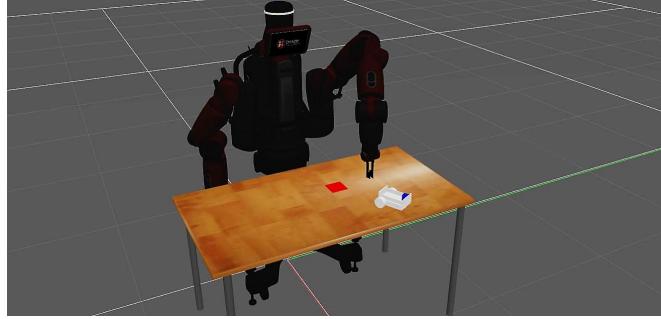
fue la red seleccionada al final de las pruebas realizadas con el Adam en el Apartado 7.2.4. La red fue entrenada con las trazas que conformaban las SURs, realizando 30 épocas de aprendizaje con cada una de las anteriores. El error de entrenamiento y las iteraciones empleadas en cada una de las pruebas se muestran en la Figura 7.11. En la gráfica que muestra la función de error, se puede ver cómo este fluctúa continuamente. Esto demuestra que el aprendizaje se realizó simulando una situación real en la que los datos se iban recopilando a medida que el robot desempeñaba su actividad con la diferencia que, en este caso, se emplearon unas SURs ya definidas para reducir el tiempo de simulación.



(a) Inicio del experimento.

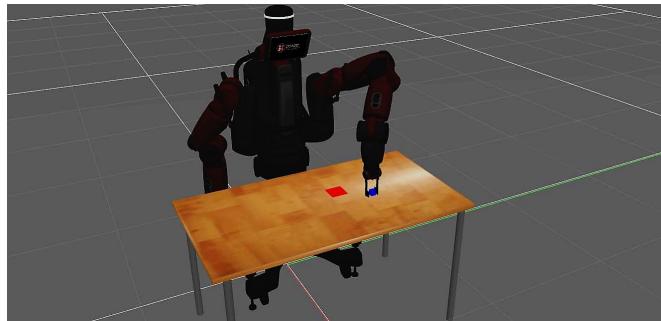


(b) Robots desplazándose hacia el bloque.

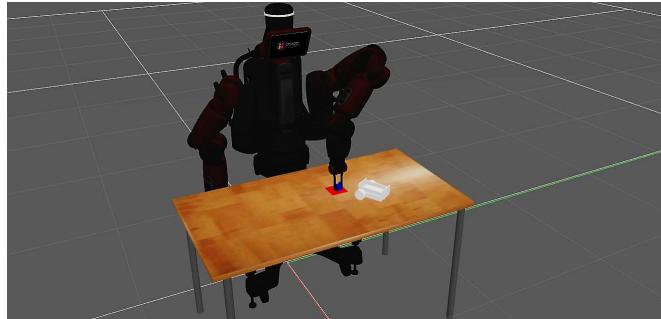


(c) Robot móvil le acerca el bloque al Baxter.

Figura 7.9: Secuencia de acciones para que el robot móvil recoga el bloque y se lo acerque al brazo del Baxter.



(a) El Baxter recoge el bloque.



(b) El Baxter coloca el bloque en la posición objetivo.

Figura 7.10: El Baxter recoge el bloque y lo coloca en la posición objetivo (*goal*)

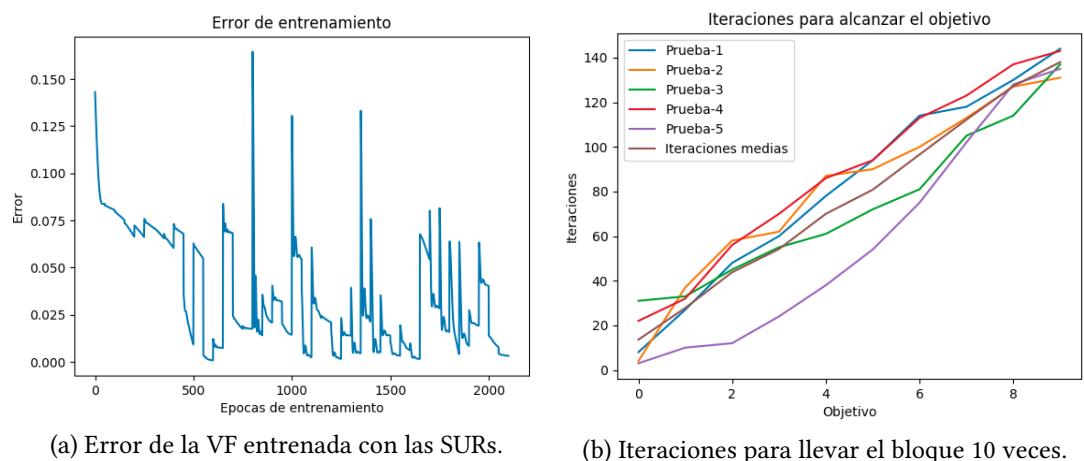


Figura 7.11: Gráficas del error e iteraciones empleadas empleando la VF entrenada a partir de las SURs.

It. mínimas	It. máximas	It. medias	It. mínimas (10)	It. medias (10)	It. máximas (10)
2	31	13.82	131	138	144

Cuadro 7.5: Información de las pruebas realizadas.

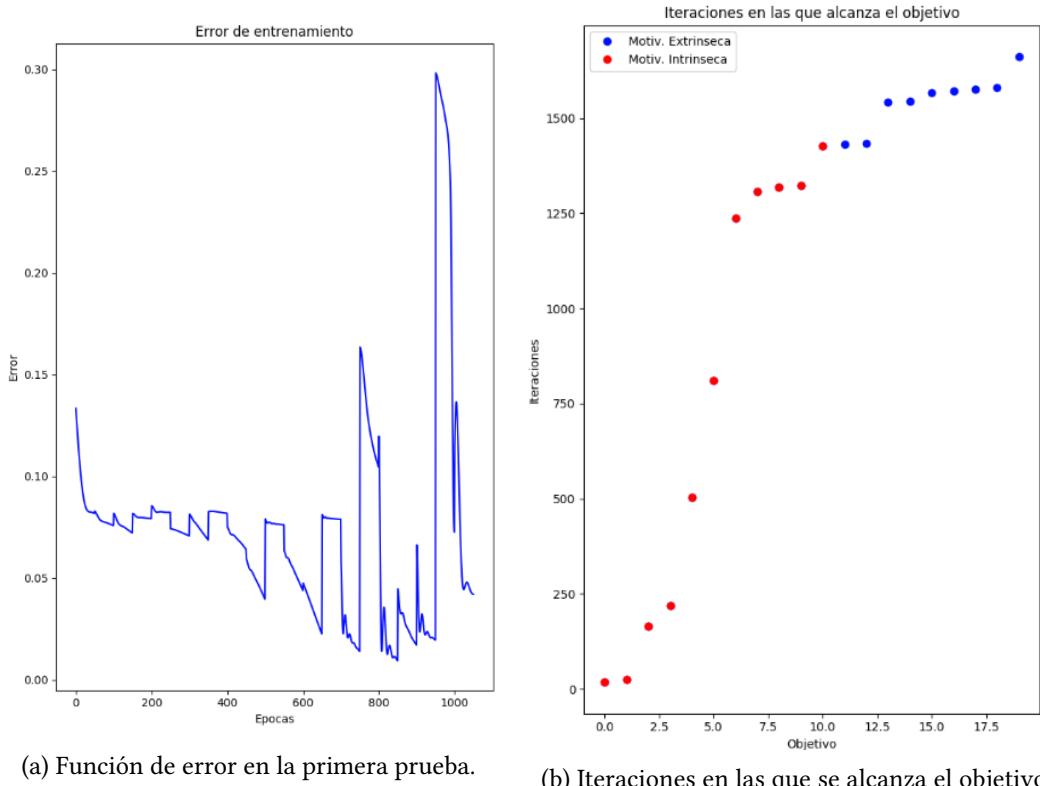
Los resultados obtenidos muestran que la VF entrenada con las SURs logra resolver las pruebas utilizando de media 138 iteraciones. Las tres primeras columnas de la Tabla 7.5 indican el número de iteraciones necesarias para situar el bloque en el *goal* una vez. Los valores de las dos primeras columnas pueden hacer referencia a casos en los que el bloque pudo haberse situado próximo al Baxter o lejos de él respectivamente, una vez se reinició el experimento. El hecho de que solo se empleasen 2 iteraciones para realizar la tarea indica que el brazo del Baxter, bloque y el *goal* se encontraban muy próximos entre sí, lo cual es posible ya que las posiciones eran aleatorias. El valor de la segunda columna podría relacionarse con algún caso en el que el bloque fue colocado en una esquina de la mesa y el robot móvil se encontraba situado en el lado opuesto de la misma. Por otro lado, las iteraciones medias parecen indicar que en la mayoría de los casos, los robots se desplazaron siempre en la dirección en la que se situaba el objeto en cuestión, lo que demuestra que el modelo de utilidad empleado realiza correctamente su labor. Las últimas columnas hacen referencia a las iteraciones totales para resolver la tarea 10 veces seguidas. El valor de ambas columnas muestra que, aunque en cada prueba la disposición de los objetos fue aleatoria y generada de manera dinámica, el sistema motivacional realizó el comportamiento esperado. Por lo tanto, se puede confirmar que una vez se poseen regiones de utilidad bien definidas, obtener una VF precisa no debería ser un problema.

7.3.2 Experimento entrenando la VF desde cero

En estas pruebas, el MotivEn fue configurado para utilizar la motivación intrínseca, por lo que el robot móvil y el brazo del Baxter debían explorar el entorno de forma aleatoria en busca de estados novedosos con los que alcanzar el objetivo. Una vez se alcanzara el objetivo, se crearía la motivación extrínseca asociada. En este caso, se configuró MotivEn para no modelar las SURs, por lo que la *DeepAnnVF* emplearía directamente las trazas recopiladas para entrenar la red. Este factor implicaba que los estados sensoriales tenían una alta probabilidad de ser ambiguos debido a las diferentes posiciones que los objetos podían tener sobre el entorno. Para determinar en qué momento se podía utilizar la VF se empleó el mapa de certeza asociado. De esta forma, el robot utilizaría la motivación extrínseca si el estado sensorial estaba dentro del área de certeza de la VF. Sin embargo, se estableció un número veces que se debía cumplir el objetivo antes de comenzar a emplear el modelo de utilidad. Esto se decidió debido a que en las etapas iniciales del aprendizaje, las áreas de certeza parten de regiones mucho mayores que presentan un valor de certeza muy similar, lo que podría hacer que el sistema utilizase la VF de forma errónea. Entonces, una vez se alcanzara el objetivo, se determinó que sería necesario llegar 10 veces más antes de poder utilizar la VF.

En este caso, se repitió el experimento tres veces. Para determinar la finalización de cada prueba, se fijó un número total de veces que se debía alcanzar el objetivo. En la primera prueba

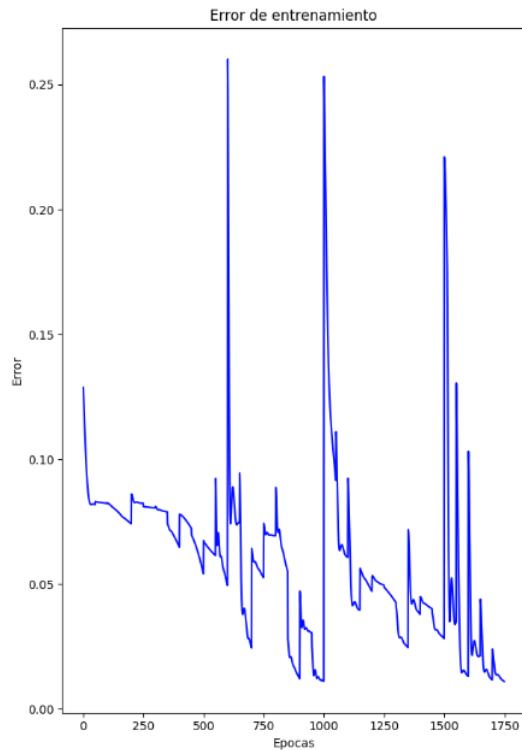
se emplearon 20 veces, en la segunda 35 y en la última 50. De esta forma, se podría analizar si el número de iteraciones para llevar el bloque al *goal* disminuye a medida que se emplea la VF.



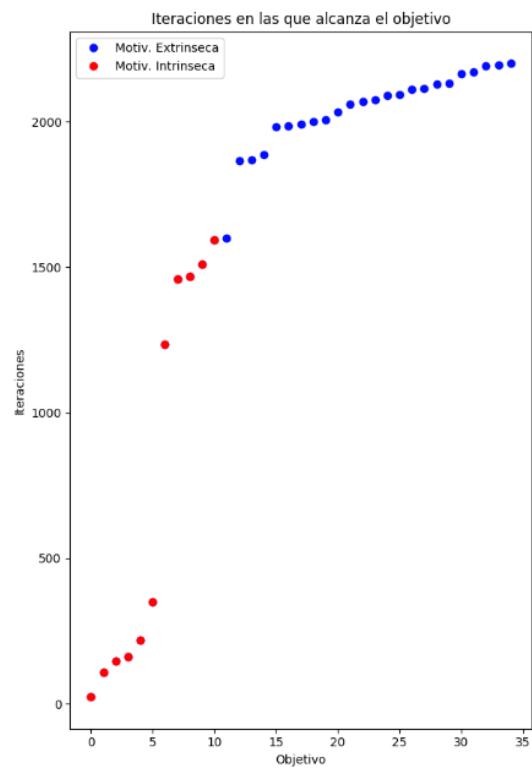
(a) Función de error en la primera prueba.

(b) Iteraciones en las que se alcanza el objetivo.

Figura 7.12: Gráficas del error e iteraciones empleadas en la primera prueba.



(a) Función de error en la primera prueba.



(b) Iteraciones en las que se alcanza el objetivo.

Figura 7.13: Gráficas del error e iteraciones empleadas en la segunda prueba.

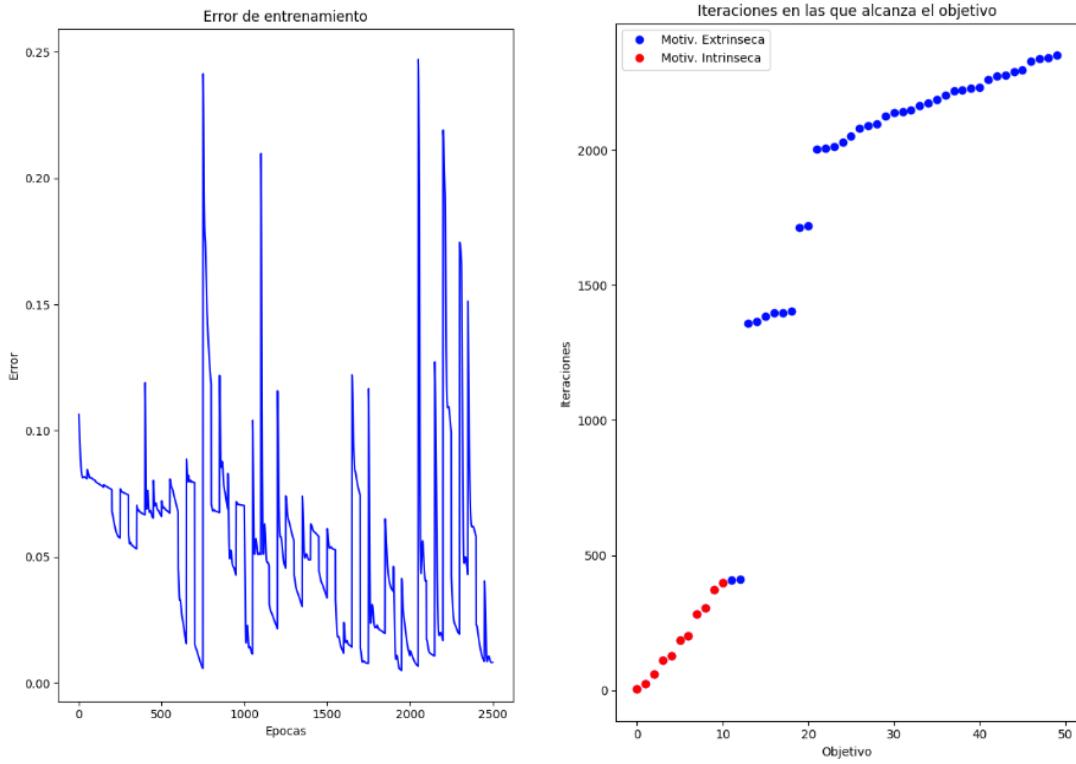


Figura 7.14: Gráficas del error e iteraciones empleadas en la tercera prueba.

Las Figuras 7.12, 7.13 y 7.14 muestran el error de entrenamiento y las iteraciones en las que se alcanzó el objetivo en cada prueba. Cuando la motivación intrínseca estaba activa, se puede apreciar que era necesario un mayor número de iteraciones para realizar la tarea. También se puede ver que existe una gran variabilidad en el número de pasos empleados para alcanzar la meta, lo que demuestra la aleatoriedad en las posiciones y acciones de los robots. Hay que señalar que los objetivos logrados intrínsecamente con un gran número de iteraciones permitieron ajustar mejor el área de certeza del modelo de utilidad. Estas diferencias se pueden identificar comparando las iteraciones de las Figuras 7.13 y 7.14. En el primer caso, fueron necesarias alrededor 1600 iteraciones para poder emplear la VF, mientras que en el segundo caso, solo se emplearon 400. Por esta razón, existe una diferencia significativa cuando se comenzó a emplear el modelo de utilidad en el segundo caso. Posiblemente se debiera a que el mapa de certeza de la VF se encontraba definida para un área concreta de la mesa, algo que pudo haber ocurrido ya que como muestra Figura 7.14, existía poca diferencia entre el número de iteraciones empleado. El reinicio del experimento y la modificación de la posición de los elementos pudo provocar la aparición de un estado sensorial desconocido hasta el momento, obligando al sistema a volver a emplear la motivación intrínseca hasta alcanzar un estado de

certeza. En cuanto a los objetivos alcanzados empleando la motivación extrínseca, se observa en ambas pruebas un descenso general en el número de pasos. Esto demuestra que, a medida que se utilizaban ambos tipos de motivaciones, el área de certeza y la red fueron ajustándose gradualmente mediante los distintos impulsos del sistema y la información recopilada.

Objetivos	It. totales (int)	It. medias (int)	It. medias (ext)	It. medias últimos 10 obj. (ext)
20	1426	140.8	28.0	28.0
35	1594	157.0	26.05	11.8
50	400	39.5	51.16	13.23

Cuadro 7.6: Información recopilada en las pruebas realizadas.

Los datos de Tabla 7.6 reflejan lo explicado anteriormente. La segunda y tercera columna contienen las iteraciones realizadas de manera intrínseca, mientras que las últimas, son la información relacionada con la motivación extrínseca. Realizar un número elevado de iteraciones de manera intrínseca permite definir una región de certeza con mayor fiabilidad. Esto se puede identificar relacionando las iteraciones de la segunda y cuarta columna, lo que concuerda con lo comentado en el párrafo anterior. Por otro lado, estos datos también muestran que, a medida que se utilizan ambos tipos de motivaciones, la red se va ajustando a la resolución de la tarea. Para determinar que la VF está obteniendo unos resultados aceptables, se obtuvo el valor medio de las iteraciones empleadas en los últimos 10 objetivos (última columna de la Tabla 7.6). Si comparamos dicho valor con el número de pasos medio que utilizaba la VF entrenada a partir de las SURs (tercera columna de la Tabla 7.5), se puede observar un número de iteraciones medio similar. Esto indica que es posible realizar el aprendizaje de una VF desde cero y sin necesidad de tener que modelar previamente las SURs. También hay que destacar que la aplicación de esta alternativa es posible que presente una mayor dificultad para tareas y/o entornos más complejos, en los que sea necesario emplear ambos tipos de funciones de utilidad. Sin embargo, puede representar una buena alternativa para definir comportamientos sencillos y flexibles de una manera más rápida y eficaz.

Finalmente, los resultados obtenidos parecen indicar que la aplicación del aprendizaje online podría ser una opción viable de cara a obtener políticas que optimicen la resolución de tareas de forma autónoma en sistemas cognitivos de aprendizaje abierto.

Capítulo 8

Conclusiones y trabajo futuro

El haber participado e intentado contribuir al desarrollo y mejora de un proyecto de tal índole como es el caso de MotivEn, ha representado para mí la principal motivación para esforzarme y lograr los objetivos planificados. De esta forma, se lograron los siguientes resultados:

- Se integraron satisfactoriamente los módulos software solicitados, compatibilizando su funcionamiento con los componentes previos a su desarrollo.
- En las pruebas realizadas, se demostró que el empleo del paradigma de aprendizaje online puede ser una alternativa viable para el desarrollo de funciones de utilidad en MotivEn.
- El entorno de simulación desarrollado permitirá incluir y analizar la respuesta del sistema motivacional en futuros experimentos más próximos a la realidad.

El trabajo desarrollado me ha permitido analizar las aptitudes y conocimientos técnicos obtenidos en los últimos cuatro años. Además, me ha mostrado en primera persona la importancia que poseen determinados factores para la consecución de un proyecto.

El principal *handicap* ha sido la falta de experiencia. Aunque la metodología seguida ha permitido tener un conocimiento global sobre las distintas fases del desarrollo, la escasa experiencia en la estimación unido al desconocimiento de las tecnologías empleadas provocaron desviaciones significativas en la evolución del proyecto. Por otro lado, dicha inexperiencia me obligó obtener un cierto nivel de conocimiento en ROS, Gazebo y TensorFlow, tecnologías populares en sus respectivos campos. El desarrollo de la Value Function empleando Deep Learning me permitió aprender más acerca de los paradigmas de aprendizaje máquina, en concreto, de los algoritmos online y de su impacto en el futuro.

Aunque el conocimiento que poseía inicialmente en robótica era escaso, y más aún, en sistemas basados en motivaciones, gracias a la experiencia y apoyo de mis directores llegué

a comprender el funcionamiento de MotivEn, y en consecuencia, a desarrollar e integrar los componentes necesarios.

El desarrollo del entorno de simulación y las pruebas realizadas, me mostraron la dificultad que exige comprobar el funcionamiento y la mejora de un sistema de este tipo, ya que fue necesario emplear una gran cantidad de tiempo para recopilar información que proporcionase una medida objetiva del correcto funcionamiento del sistema.

Realizar este proyecto me ha formado en el campo de la robótica y en el ámbito del Deep Learning llegando a saber utilizar algunas de las tecnologías más populares en dichos campos.

8.1 Trabajo futuro

Los procedimientos realizados han permitido la consecución de los objetivos planteados inicialmente. Sin embargo, al tratarse de un proyecto de robótica cognitiva centrado en el campo del aprendizaje abierto, todavía se podrían realizar diferentes mejoras y experimentos que podrían contribuir a mejorar el sistema motivacional:

- Comportamiento de los robots. Mejorar el desenvolvimiento de determinadas acciones de bajo nivel realizadas por los robots, ya que había ocasiones en las que el agarre del objeto o su desplazamiento no eran realizados con exactitud.
- Utilizar un modelo 3D del Robobo. Utilizar un modelo virtual del robot Robobo reduciría aún más las diferencias con la realidad, lo que podría ser de utilidad para evitar futuros problemas en un experimento con robots reales.
- Experimento en un entorno real. Una vez analizado el experimento en un entorno virtual, se podría evaluar el comportamiento del sistema en robots reales. Esto permitiría comprobar que los cambios realizados en la arquitectura han proporcionado un sistema más fácil de extender y mantener.
- Aprendizaje en paralelo de los modelos de utilidad. El hecho de que se pudiese aprender una SUR y una Value Function simultáneamente, proporcionaría más información sobre en qué etapas de la actividad del robot sería aconsejable emplear uno u otro modelo. Además, con este tipo configuración sería posible definir una jerarquía de modelos de utilidad que persigan la consecución óptima de distintos objetivos locales.
- Evaluación del sistema motivacional en un entorno con diferentes objetivos. Realizar un experimento en el que sistema autónomo tuviese que aprender a realizar dos tareas diferentes permitiría estudiar una forma de reutilizar el conocimiento del sistema. Eso podría facilitar el establecimiento de un conjunto de acciones comunes y mejorar la adaptabilidad a largo plazo.

Apéndices

Apéndice A

Glosario de acrónimos

AdaGrad *Adaptive Gradient*

Adam *Adaptive Moment Estimation*

ANN *Artificial Neural Network*

API *Application Programming Interface*

CBIM *Category-Based Intrinsic Motivation*

DREAM *Deferred Restructuring of Experience in Autonomous Machines*

GII *Grupo Integrado de Ingeniería*

IAC *Intelligent Adaptive Curiosity*

IMRL *Intrinsically Motivated Reinforcement Learning*

MotivEn *Motivational Engine*

ODE *Open Dynamics Engine*

RMSProp *Root Mean Square Propagation*

ROS *Robot Operating System*

RPC *Remote Procedure Call*

SGD *Stochastic Gradient Descent*

SUR *Separable Utility Region*

UML *Unified Modeling Language*

VF Value Function

XML eXtensible Markup Language

Apéndice B

Manual de integración

En este apéndice se explican los pasos que deberán seguir los investigadores del GII para emplear la versión de MotivEn con los módulos y el entorno de simulación desarrollado en este trabajo. Para importar la versión actual de MotivEn, bastará con copiar el proyecto en uno de los directorios del sistema de archivos. En cuanto a la integración del paquete ROS desarrollado, los pasos serán distintos.

B.1 Integración del paquete ROS

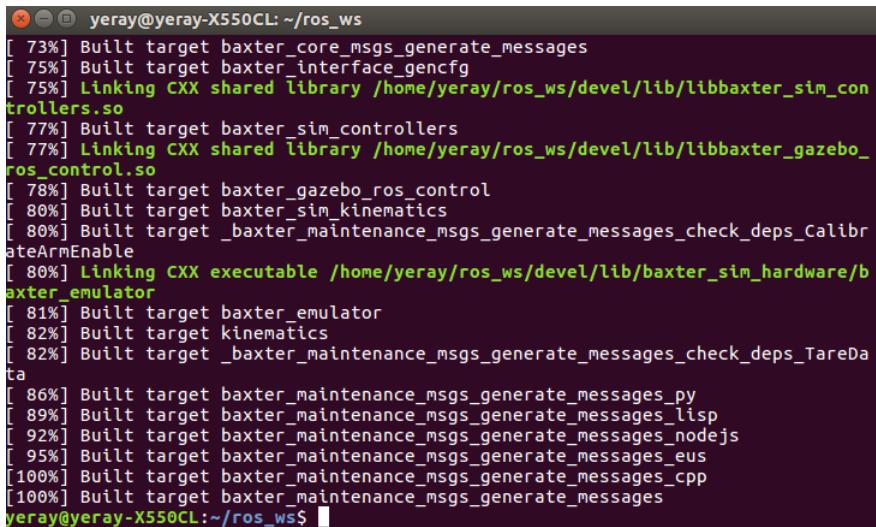
En las instrucciones que se explican a continuación se considera que el sistema operativo en el que se integrará el simulador es una distribución de Ubuntu que cuenta con una distribución de ROS Kinetic instalada, ya posee un *workspace* de ROS y ha instalado el API y el modelo virtual del robot Baxter.

El primer paso se basa en mover o copiar el paquete ROS *simulator* en el directorio *src* del espacio de trabajo. El *workspace* ROS se divide en tres espacios o directorios[24]:

- Directorio *src*. Contiene los paquetes de ROS con el código y configuración de los experimentos.
- Directorio *build*. En este directorio se encuentran los paquetes del directorio *src* compilados.
- Directorio *devel*. Contiene los *targets*, es decir, los archivos de ejecución de los paquetes ROS compilados. Estos podrán ser ejecutados cuando se cargue el *workspace* en la variable de entorno de ROS, mediante el script *setup.bash*.

Una vez copiado el paquete, para lanzar el simulador es necesario construir el paquete *simulator*. Para ello hay que lanzar una consola o terminal y acceder al directorio en el que se

creara el *workspace*. Una vez dentro, se ejecutará el comando *catkin_make*, el cual construye todos los paquetes del *workspace*. En la Figura B.1 se muestra el resultado de dicha acción.



```
yeray@yeray-X550CL: ~/ros_ws
[ 73%] Built target baxter_core_msgs_generate_messages
[ 75%] Built target baxter_interface_gencfg
[ 75%] Linking CXX shared library /home/yeray/ros_ws/devel/lib/libbaxter_sim_controllers.so
[ 77%] Built target baxter_sim_controllers
[ 77%] Linking CXX shared library /home/yeray/ros_ws/devel/lib/libbaxter_gazebo_ros_control.so
[ 78%] Built target baxter_gazebo_ros_control
[ 80%] Built target baxter_sim_kinematics
[ 80%] Built target _baxter_maintenance_msgs_generate_messages_check_deps_CalibrateArmEnable
[ 80%] Linking CXX executable /home/yeray/ros_ws/devel/lib/baxter_sim_hardware/baxter_emulator
[ 81%] Built target baxter_emulator
[ 82%] Built target kinematics
[ 82%] Built target _baxter_maintenance_msgs_generate_messages_check_deps_TareData
[ 86%] Built target baxter_maintenance_msgs_generate_messages_py
[ 89%] Built target baxter_maintenance_msgs_generate_messages_lisp
[ 92%] Built target baxter_maintenance_msgs_generate_messages_nodejs
[ 95%] Built target baxter_maintenance_msgs_generate_messages_eus
[100%] Built target baxter_maintenance_msgs_generate_messages_cpp
[100%] Built target baxter_maintenance_msgs_generate_messages
yeray@yeray-X550CL:~/ros_ws$
```

Figura B.1: Resultado de ejecutar del comando *catkin_make*.

Tras esto, es necesario configurar el script que se encargará de la comunicación con el modelo virtual del Baxter. Estando en el *workspace*, se busca el script *baxter.sh* y se abre empleando un editor de texto. Con el fichero abierto, se debe buscar la línea en la que se indique la distribución de ROS que empleará el script y en ella se introduce *kinetic*. Por último se debe identificar la dirección IP que se empleará para comunicarse con el Baxter. Esto se debe a que la API del Baxter utiliza este script para poder conectarse con el robot real a través de su dirección IP, pero dado que en este caso emplearemos un modelo virtual, es necesario indicarle la dirección de nuestro equipo. En la Figura B.2 se muestra la configuración que empleará el script.

APÉNDICE B. MANUAL DE INTEGRACIÓN

```
#!/bin/bash
# Copyright (c) 2013-2015, Rethink Robotics
# All rights reserved.

# This file is to be used in the *root* of your Catkin workspace.

# This is a convenient script which will set up your ROS environment and
# should be executed with every new instance of a shell in which you plan on
# working with Baxter.

# Clear any previously set your_ip/your_hostname
unset your_ip
unset your_hostname
#-----
#-----          USER CONFIGURABLE ROS ENVIRONMENT VARIABLES          #
#-----#
# Note: If ROS_MASTER_URI, ROS_IP, or ROS_HOSTNAME environment variables were
# previously set (typically in your .bashrc or .bash_profile), those settings
# will be overwritten by any variables set here.

# Specify Baxter's hostname
#baxter_hostname="baxter_hostname.local"
baxter_hostname="yeray-X550CL"
# Set *Either* your computers ip address or hostname. Please note if using
# your hostname that this must be resolvable to Baxter.
#your_ip="192.168.XXX.XXX"
#your_ip="10.236.76.108"
your_ip="192.168.1.42"
#your_ip="192.168.0.19"
#your_hostname="my_computer.local"

# Specify ROS distribution (e.g. indigo, hydro, etc.)
#ros_version="indigo"
ros_version="kinetic"
#-----#
```

Figura B.2: Configuración necesaria para comunicarse con el modelo virtual del Baxter.

Una vez realizada la configuración, se cerrará el fichero. Para finalizar y comprobar que se ha integrado y configurado el modelo del Baxter de forma correcta, se ejecutará en la terminal script *baxter.sh* seguido del comando *roslaunch simulator simulator.launch* para lanzar el entorno virtual de Gazebo, tal y como se muestra en la Figura B.3. En la Figura B.4 se puede ver el escenario de simulación tras ejecutar *roslaunch*.

```
yeray@yeray-X550CL:~/ros_ws$ source baxter.sh
[baxter - http://yeray-X550CL:11311] yeray@yeray-X550CL:~/ros_ws$ roslaunch simulator simulator.launch
```

Figura B.3: Ejecución del script *baxter.sh* y del entorno de simulación del experimento.

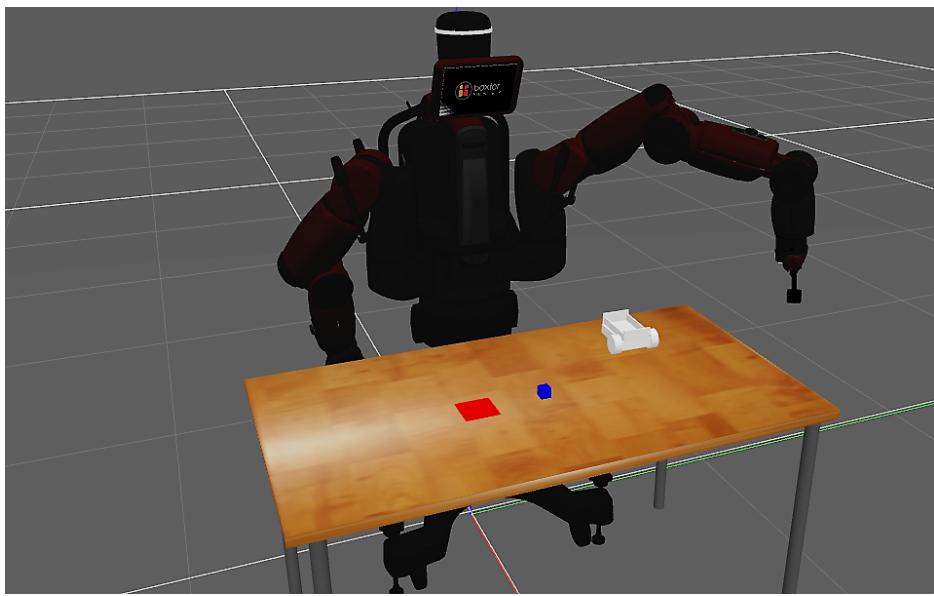


Figura B.4: Entorno de simulación cargado.

Apéndice C

Manual de configuración

Como se comentó en la Iteración 6.7, se emplearon una serie de ficheros de configuración para definir el comportamiento inicial de MotivEn y el tipo de simulador que se emplearía para lanzar el entorno robótico virtual. En este apéndice se explicará el tipo de parámetros que se pueden emplear para determinar la configuración inicial del sistema motivacional.

C.1 Configuración de MotivEn

El fichero *Configuration.xml* permitirá configurar el comportamiento inicial que se desee para el sistema motivacional e indicarle el entorno de simulación a partir del cual recopilará la información necesaria.

El tag *initial_motivation* le indicará el tipo de motivación que se utilizará para guiar la conducta del robot. También se han definido una serie de tags para configurar el uso de los modelos de utilidad. Los tags *use_value_function*, *use_surs* y el tag *initial_utility_model* establecerán que modelo de utilidad se empleará inicialmente para la motivación extrínseca. Por último, el tag *simulator_type* servirá para determinar el simulador en el que se lanzará el entorno virtual. En la Figura C.1 se muestra una posible configuración.

```
<?xml version="1.0"?>
<MDBCoreConfiguration>

    <simulator_type>gazebo</simulator_type>
    <initial_motivation>Int</initial_motivation>
    <use_value_function>True</use_value_function>
    <use_surs>False</use_surs>
    <initial_utility_model>VF</initial_utility_model>

</MDBCoreConfiguration>
```

Figura C.1: Posible configuración de MotivEn.

También será necesario configurar el script *baxter_start.sh*. Dicho script cargará la configuración del fichero *baxter.sh* comentado en el Apéndice B. Por lo tanto, es necesario indicarle la ruta en la que se encuentra la configuración del Baxter. En la Figura C.2, se muestra la configuración definida en este trabajo.

```
#!/usr/bin/env bash
cd /home/yeray/ros_ws/
./baxter.sh
```

Figura C.2: Script que permite cargar la configuración para comunicarse con el robot Baxter.

Por último, para poder ejecutar MotivEn con éxito será necesario configurar el script *start_motivEN.sh*. En dicho fichero, se especificará la ruta donde se encuentra el fichero que permite cargar los paquetes de simulación en la variable de entorno de ROS. Esto se debe a que el componente responsable de la comunicación con el entorno de simulación emplea diferentes métodos definidos en la API del Baxter, la cual se compone de varios paquetes ROS. El script responsable de esta tarea es el *setup.bash* que se encuentra en directorio *devel* del área de trabajo de ROS. Finalmente se ejecutará con *python* el fichero principal del sistema, el *MDBCore.py*. En la Figura C.3, se muestra la configuración del script *start_motivEN.sh*.

```
#!/bin/bash
source /home/yeray/ros_ws/devel/setup.bash
python MDBCore.py
```

Figura C.3: Configuración del script *start_motivEN.sh*.

C.2 Configuración de los simuladores

Dentro de MotivEn se definió el archivo *SimulatorConfiguration.xml* el cual contendrá una serie de parámetros que serán empleados por el respectivo simulador para cargar diferentes configuraciones.

```
<?xml version="1.0"?>
<SimulatorConfiguration>

<vel_robobo>0.05</vel_robobo>
<despl_robobo>0.1</despl_robobo>
<despl_baxter>0.2</despl_baxter>
<!-- Límites -->
<robobo_min_x>0.35</robobo_min_x>
<robobo_max_x>0.9</robobo_max_x>
<robobo_min_y>-0.6</robobo_min_y>
<robobo_max_y>0.6</robobo_max_y>
<baxter_min_x>0.5</baxter_min_x>
<baxter_max_x>0.7</baxter_max_x>
<baxter_min_y>-0.2</baxter_min_y>
<baxter_max_y>0.3</baxter_max_y>
</SimulatorConfiguration>
```

Figura C.4: Posible configuración a emplear en alguno de los simuladores desarrollados.

En la Figura C.4 se muestra el contenido de dicho fichero de configuración. El tag *vel_robobo* permite establecer la potencia que se aplicará a las ruedas del robot móvil. Los tags *despl_robobo* y *despl_baxter* permitirán configurar el rango de desplazamiento del robot móvil y del brazo del Baxter. El resto de tags permiten establecer los límites entre los que los robots pueden moverse, esto es especialmente útil para controlar el área en la que queremos colocar el *goal* o el bloque a recoger por el Baxter.

Apéndice D

Manual de ejecución

Para poder ejecutar el MotivEn y el simulador desarrollado con Gazebo y ROS se deben seguir los siguientes pasos:

- En primer lugar, se debe lanzar un terminal y acceder al directorio en el que se encuentre el módulo MotivEn. Una vez allí, se ejecutará el script *baxter_start.sh* para cargar la configuración del modelo virtual del Baxter. El siguiente paso será ejecutar por terminal el comando *roslaunch simulator simulator.launch* y esperar a que el entorno de simulación se cargue. En la Figura D.1 se muestran los pasos realizados.
- El último paso será lanzar un nuevo terminal de consola y ejecutar el script *start_motivEN.sh*.

```
yeray@yeray-X550CL:~/ros_ws  
yeray@yeray-X550CL:~$ cd Escritorio/MDBCore/Experimento1_4/  
yeray@yeray-X550CL:~/Escritorio/MDBCore/Experimento1_4$ source baxter_start.sh  
[baxter - http://yeray-X550CL:11311] yeray@yeray-X550CL:~/ros_ws$ roslaunch simulator simulator.launch
```

Figura D.1: Ejecución del script *baxter_start.sh* y del entorno de simulación del experimento.

```
yeray@yeray-X550CL:~/Escritorio/MDBCore/Experimento1_4  
yeray@yeray-X550CL:~/Escritorio/MDBCore/Experimento1_4$ ./start_motivEN.sh
```

Figura D.2: Ejecución de MotivEn.

Apéndice E

Contenido del CD

El CD entregado junto con la copia física de este documento contiene los siguientes archivos:

- **MendezRomero_Yeray_TFG_2019.pdf**: versión electrónica de esta memoria.
- **MendezRomero_Yeray_TFG_2019_resumo.pdf**: contiene el resumen de este documento.
- **MendezRomero_Yeray_TFG_2019_code.zip**: contiene el código fuente desarrollado en este trabajo.
- **MendezRomero_Yeray_TFG_2019_images.zip**: contiene algunas de las imágenes(arquitectura, gráficas de error, etc) incluidas en este documento.

Bibliografía

- [1] Gerhard Lakemeyer and Hector Levesque, “Cognitive robotics,” *Foundations of Artificial Intelligence*, vol. 3, no. 23, pp. 869–866”, 12 2008.
- [2] DREAM, “Deferred restructuring of experience in autonomous machines,” <http://www.robotsthatdream.eu/>, 2015.
- [3] G. I. de Ingeniería, “Grupo integrado de ingeniería,” <http://www.gii.udc.es/>, 2014.
- [4] Alejandro Romero, Abraham Prieto, Francisco Bellas, Rodrigo Salgado, and Richard J. Duro, “Simplifying the creation and management of utility models in continuous domains for cognitive robotics,” *Neurocomputing*, pp. 1–12, 2018.
- [5] ——, “Introducing separable utility regions in a motivational engine for cognitive developmental robotics,” *Integrated Computer-Aided Engineering*, pp. 3–20, 2018.
- [6] M. Korein, “Interactions between intrinsic and extrinsic motivation,” *Carnegie Mellon University*, pp. 1–11, 5 2010.
- [7] R. Robotics, “Baxter research robot,” <http://sdk.rethinkrobotics.com/wiki/Home>, 2015.
- [8] Robobo, “Robobo,” <https://theroboboproject.com/?lang=es>, 2017.
- [9] Doyen Sahoo, Jing Lu, and Peilin Zhao, “Online learning: A comprehensive survey,” *SMU Technical Report*, no. 1, pp. 1–100, 10 2018.
- [10] N. Designer, “Five algorithms to train a neural network,” https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network, 2015.
- [11] S. Ruder, “An overview of gradient descent optimization algorithms,” *ArXiv*, pp. 1–14, 1 2016.
- [12] Frederic Kaplan and Pierre-Yves Oudeyer, “Intelligent adaptive curiosity: a source of self-development,” vol. 117, 2004.

- [13] James Marshall, Lisa Meeden, Rachel Lee, and Ryan Walker, “Category-based intrinsic motivation,” 2009.
- [14] Andrew G. Barto, Nuttapong Chentanez, and Satinder P. Singh, “Intrinsically motivated reinforcement learning,” in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. MIT Press. [Online]. Available: <http://papers.nips.cc/paper/2552-intrinsically-motivated-reinforcement-learning.pdf>
- [15] Andrew Bagnell, Kober Jens, and Peters Jan, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, 09 2013.
- [16] Adam Laud and Gerald DeJong, “The influence of reward on the speed of reinforcement learning: An analysis of shaping.” 01 2003, pp. 440–447.
- [17] F. Patrawala, “Implement reinforcement learning using markov decision process,” <https://medium.com/coinmonks/implement-reinforcement-learning-using-markov-decision-process-tutorial-272012fdae51>, 2018.
- [18] A. Zelinsky, C. Gaskett, and L. Fletcher, “Reinforcement learning for a vision based mobile robot,” in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 2000, pp. 403–409.
- [19] Baoxia Cui, Huaiqing Yang, and Yong Duan, “Robot navigation based on fuzzy rl algorithm,” in *Advances in Neural Networks - ISNN 2008*, Fuchun Sun, Jianwei Zhang, Jinde Cao, Ying Tan, and Wen Yu, Eds., 2008, pp. 391–399.
- [20] Leslie Pack Kaelbling and William D. Smart, “A framework for reinforcement learning on real robots,” *Computer Science Department Brown University*, 1998.
- [21] G. Atkeson, C. Bentivegna, and m. . . p. . . t. . L. Gordon Cheng, , year = 2019.
- [22] Dan Wu, Dezfoulian Hamid, and Imran Shafiq Ahmad, “A generalized neural network approach to mobile robot navigation and obstacle avoidance,” *Advances in Intelligent Systems and Computing*, vol. 193, pp. 25–42, 01 2013.
- [23] Bahram SADEGHIBIGHAM and Farhad SHAMSFAKHR, “A neural network approach to navigation of a mobile robot and obstacle avoidance in dynamic and unknown environments,” 2017.
- [24] ROS, “What is ros?” <http://wiki.ros.org/ROS/Introduction>, 2018.

BIBLIOGRAFÍA

- [25] J. Buhigas, “Todo lo que necesitas saber sobre tensorflow, la plataforma para inteligencia artificial de google,” <https://puentesdigitales.com/2018/02/14/todo-lo-que-necesitas-saber-sobre-tensorflow-la-plataforma-para-inteligencia-artificial-de-google/>, 2018.
- [26] J. Torres, *Deep Learning – Introducción práctica con Keras*, 2018, no. 6.
- [27] Gazebo, “Gazebo,” <http://gazebosim.org/>, 2014.
- [28] ROBOLOGS, “Gazebo simulator: simular un robot nunca fue tan fácil,” <https://robologs.net/2016/06/25/gazebo-simulator-simular-un-robot-nunca-fue-tan-facil/>, 2016.
- [29] V. R. Villán, “Las metodologías ágiles más utilizadas y sus ventajas dentro de la empresa,” <https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/>, 2018.
- [30] P. Gupta, “Cross-validation in machine learning,” <https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f>, 2017.

