Universidad Carlos III de Madrid

Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Integración de Planificación Automática y ROS para el control autónomo de dos robots en el juego del *Sokoban*

Autor: Raúl García Jerez

Tutor: Ángel García Olaya

Agradecimientos

En primer lugar, por supuesto a mi familia, sobre todo a mis padres, ya que sin ellos no podría estar aquí. Pero tampoco me olvido de mi hermano, ni de mis abuelos, tíos, y todos los demás.

También a todas las personas que me han acompañado a lo largo de mi vida académica, en el colegio, en el instituto y en la Universidad, sobre todo a los que han pasado de ser simples compañeros a ser amigos, con los que espero no perder nunca el contacto.

Por supuesto, de estos últimos tengo que nombrar a Alejandro, por todos los días perdidos para ayudarme a terminar de poner a funcionar los robots, hacer las grabaciones, y en general por aguantarme durante la realización de todo el trabajo, su ayuda es impagable.

Cómo no, a mi tutor Ángel García Olaya y su infinita paciencia con mis dudas y mis idas de olla los días previos a la entrega, sin su ayuda habría sido imposible llegar a tiempo para este trabajo.

Por último pero no menos importante, a mi pareja, Rosi, ya que es la que me ha dado más fuerzas para lograr esto, la que me ayudaba a desconectar y tener días o ratos de disfrute a pesar de lo duros que han sido los dos últimos meses, y es la que más ha aguantado mi “monotema”, este TFG.

Y a todos los que hayan contribuido lo más mínimo y no haya nombrado, gracias.

Resumen

El presente trabajo pretende unir dos áreas de la informática: la Robótica y la Planificación Automática. La Planificación Automática proporciona una secuencia de acciones que, partiendo de un estado inicial, permiten llegar a un estado meta deseado. Aplicado a la robótica, esto permite resolver problemas en un entorno real.

El objetivo de este trabajo es desarrollar un sistema que utilice Planificación Automática para el control de robots autónomos en problemas multi-agente. Como prueba de ello, el sistema permitirá a dos robots de tipo P3DX resolver problemas del dominio del juego *Sokoban*.

Palabras clave: Inteligencia Artificial, Robótica, Planificación Automática, ROS, PELEA, P3DX, Java, rosjava, P2OS, multi-agente, Sokoban.

Abstract

This project intends to connect two areas of Computer Science: Robotics and Automated Planning. Automated Planning provides a sequence of actions that allows reaching a desired goal state from an initial state. Applied to robotics, this makes possible to solve problems in a real environment.

The goal of this project is to develop a system that uses Automated Planning to control independent robots in multi-agent problems. To illustrate this, the system will make able for two P3DX robots to solve problems from the *Sokoban* domain.

Keywords: Artificial Intelligence, Robotics, Automated Planning, ROS, PELEA, P3DX, Java, rosjava, P2OS, multi-agent, Sokoban.

Índice General

Agradecimientos ... 1

Resumen ... 2

Abstract ... 2

Capítulo 1: Introducción ... 11

1.1 Descripción del problema ... 12

1.2 Motivación ... 13

1.3 Objetivos del trabajo ... 14

1.4 Estructura del documento ... 14

Capítulo 2: Estado del Arte ... 17

2.1 Robótica ... 17

2.2 Planificación Automática ... 25

2.3 PELEA ... 28

2.4 ROS ... 31

2.4.1 Programación en rosjava ... 33

Capítulo 3: Descripción del sistema ... 35

3.1 Análisis del sistema ... 36

3.1.1 Descripción de las características funcionales ... 36

3.1.2 Restricciones del sistema ... 37

3.1.3 Entorno operacional ... 38

3.1.4 Especificación de casos de uso ... 39

3.1.5 Especificación de requisitos ... 44

3.2 Diseño del sistema ... 56

3.2.1 Descripción general del sistema ... 56

3.2.2 Descripción de componentes ... 57

3.2.3 Funcionamiento del sistema ... 69

Capítulo 4: Experimentación ... 71

4.1 Elementos del entorno de pruebas ... 71

4.2 Pruebas unitarias ... 72

4.2.1 Descripción de los atributos de las pruebas unitarias ... 72

4.2.2 Descripción textual de las pruebas unitarias ... 73

4.3 Pruebas del sistema ... 78

4.3.1 Preparación de las pruebas del sistema ... 78

4.3.2 Pruebas del sistema realizadas ... 79

Capítulo 5: Gestión del trabajo ... 83

5.1 Descripción de las fases del trabajo ... 83

5.2 Planificación ... 84

5.3 Presupuesto ... 85

Capítulo 6: Conclusiones y trabajos futuros ... 89

6.1 Conclusiones generales ... 89

6.2 Conclusiones referentes a los objetivos ... 89

6.3 Trabajos futuros ... 90

Glosario ... 92

Bibliografía ... 93

Anexo A: Instalación y configuración del entorno ... 95

A.1 Instalación y configuración de ROS Hydro ... 95

A.2 Instalación y configuración de PELEA ... 98

Anexo B: Manual de uso del sistema ... 101

Anexo C: Resumen en inglés ... 103

C.1 Introduction 103

C.1.1 Description of the problem 104

C.1.2 Motivation 105

C.1.3 Goals of the work 106

C.1.4 Summary structure 107

C.2 System description 107

C.2.1 System analysis 107

C.2.2 System design 108

C.3 Conclusions and future work 110

C.3.1 General conclusions 110

C.3.2 Conclusions concerning the objectives 111

C.3.3 Future work 112

Anexo D: Documentos de interés 113

D.1 Mensajes 113

D.1.1 geometry_msgs/Twist 113

D.1.2 myjava_msgs/SensorsData 113

D.2 Dominios 115

D.2.1 *Sokoban* original 115

D.2.2 *Sokoban* modificado 117

D.3 Scripts de ejecución 121

D.3.1 Script ROS.sh 121

D.3.2 Script PELEA5.sh 121

D.3.3 Script PELEA5B.sh (segundo PC) 122

Índice de figuras

Figura 1 - Ejemplo de un problema en *Sokoban* 12
Figura 2 - Robots P3DX 13
Figura 3 - Autómata teatro de Herón de Alejandría 18
Figura 4 - Reloj elefante de Al-Jazari 18
Figura 5 - Gallo de Estrasburgo 19
Figura 6 - Posible diseño del león de Da Vinci 19
Figura 7 - Telar de Jacquard 20
Figura 8 - Exoesqueleto del *Handyman* 21
Figura 9 - Brazos del *Handyman* 21
Figura 10 - Robot *Shakey* 21
Figura 11 - Evolución de los robots Honda 22
Figura 12 - Robots Aibo y Nao 23
Figura 13 - Prototipo de *Super Ball Bot* 24
Figura 14 - Esquema de PELEA de cinco módulos 31
Figura 15 - Grafo funcionamiento de ROS 32
Figura 16 - Esquema general de la estructura del sistema 35
Figura 17 - Diagrama del entorno operacional del sistema 38
Figura 18 - Diagrama de casos de uso 40
Figura 19 - Diagrama general de la arquitectura del sistema 57
Figura 20 - Diagrama del módulo de control 58
Figura 21 - Diagrama de flujo de *Controller* 61
Figura 22 - Diagrama del módulo de planificación 63
Figura 23 - Diagrama de flujo de *Monitoring* 68
Figura 24 - Diagrama de secuencia del sistema 70
Figura 25 - Simulador P2OS 72
Figura 26 - Leyenda mapas *Sokoban* 79
Figura 27 - Problema *Sokoban* 1 79
Figura 28 - Problema *Sokoban* 2 80

Figura 29 - Problema *Sokoban* 3 82
Figura 30 - Ciclo de vida en cascada 83
Figura 31 - Diagrama de Gantt 86
Figura 32 - Instalación de PELEA (I) 98
Figura 33 - Instalación de PELEA (II) 99
Figura 34 - Instalación de PELEA (III) 99
Figura 35 - Instalación de PELEA (IV) 100
Figure 1 - Example of a problem in *Sokoban* 104
Figure 2 - P3DX Robots 105
Figure 3 - General diagram of the system architecture 109

Índice de tablas

Tabla 1 - Caso de uso CU-001 41
Tabla 2 - Caso de uso CU-002 42
Tabla 3 - Caso de uso CU-003 42
Tabla 4 - Caso de uso CU-004 43
Tabla 5 - Caso de uso CU-005 43
Tabla 6 - Requisito del sistema RF-001 46
Tabla 7 - Requisito del sistema RF-002 46
Tabla 8 - Requisito del sistema RF-003 46
Tabla 9 - Requisito del sistema RF-004 47
Tabla 10 - Requisito del sistema RF-005 47
Tabla 11 - Requisito del sistema RF-006 47
Tabla 12 - Requisito del sistema RF-007 48
Tabla 13 - Requisito del sistema RF-008 48
Tabla 14 - Requisito del sistema RF-009 48
Tabla 15 - Requisito del sistema RF-010 49
Tabla 16 - Requisito del sistema RF-011 49
Tabla 17 - Requisito del sistema RF-012 49
Tabla 18 - Requisito del sistema RF-013 50
Tabla 19 - Requisito del sistema RF-014 50
Tabla 20 - Requisito del sistema RF-015 50
Tabla 21 - Requisito del sistema RF-016 51
Tabla 22- Requisito del sistema RF-017 51
Tabla 23 - Requisito del sistema RNF-001 51
Tabla 24 - Requisito del sistema RNF-002 52
Tabla 25 - Requisito del sistema RNF-003 52
Tabla 26 - Requisito del sistema RNF-004 52
Tabla 27 - Requisito del sistema RNF-005 53
Tabla 28 - Requisito del sistema RNF-006 53

Tabla 29 - Requisito del sistema RNF-007 53
Tabla 30 - Requisito del sistema RNF-008 54
Tabla 31 - Requisito del sistema RNF-009 54
Tabla 32 - Requisito del sistema RNF-010 54
Tabla 33 - Requisito del sistema RNF-011 55
Tabla 34 - Requisito del sistema RNF-012 55
Tabla 35 - Requisito del sistema RNF-013 55
Tabla 36 - Prueba unitaria PU-001 73
Tabla 37 - Prueba unitaria PU-002 74
Tabla 38 - Prueba unitaria PU-003 74
Tabla 39 - Prueba unitaria PU-004 75
Tabla 40 - Prueba unitaria PU-005 75
Tabla 41 - Prueba unitaria PU-006 76
Tabla 42 - Prueba unitaria PU-007 76
Tabla 43 - Prueba unitaria PU-008 77
Tabla 44 - Prueba unitaria PU-009 77
Tabla 45 - Planificación 85
Tabla 46 - Datos del trabajo 85
Tabla 47 - Coste de personal 87
Tabla 48 - Coste de material 87
Tabla 49 - Resumen de costes 88

Capítulo 1: Introducción

Durante los últimos años, la robótica está cambiando. Antiguamente, los robots únicamente se ocupaban de realizar tareas repetitivas y monótonas pero, en los últimos tiempos, han empezado a perseguir objetivos mucho más ambiciosos. Es lo que se llama robótica inteligente.

La robótica inteligente es una rama de la robótica que, en la actualidad, se encuentra en auge debido a los avances tanto en hardware como en software. La “inteligencia” de estos robots viene dada por técnicas que no son nuevas, pero que hasta hace muy poco no se utilizaban en este ámbito, como la Planificación Automática (PA).

A pesar de que inicialmente la PA surge como respuesta a un problema planteado por la robótica (el control autónomo del robot *Shakey*, que se mencionará en el siguiente capítulo del documento), durante muchos años han sido dos disciplinas totalmente alejadas, sobre todo por falta de madurez. Sin embargo, recientemente se ha llegado a un punto de desarrollo en el que intentar esta integración vuelve a tener sentido.

Es fácil ver la utilidad de este tipo de robots en problemas del mundo real cuando la NASA los utiliza en todas las misiones en las que quiere obtener información de entornos externos a la Tierra.

Por otro lado, un tipo de sistema que está cobrando fuerza últimamente es el multi-agente. El ejemplo más conocido de robótica inteligente multi-agente es la competición anual *Robocup*, donde un grupo de robots juegan al fútbol utilizando técnicas de IA.

Pero la robótica inteligente multi-agente no sólo tiene fines de investigación o lúdicos, sino que, resolviendo problemas como los del presente trabajo, podemos ver que el alcance de este tipo de sistemas está lejos de lo que, hasta hace muy poco, cabía imaginar.

1.1 Descripción del problema

El problema a resolver mediante este trabajo consiste en la resolución de problemas del famoso rompecabezas *Sokoban* en un entorno real con dos jugadores, donde cada uno de ellos será un robot de tipo P3DX controlado mediante Planificación Automática.

En la Figura 1 se muestra un ejemplo de una pantalla del *Sokoban* (en japonés "encargado de almacén") con dos jugadores. Este rompecabezas consiste en empujar todas las cajas hacia las casillas marcadas como meta, pudiendo sólo empujar las cajas y no tirar de ellas. El jugador únicamente puede moverse en horizontal o vertical, sólo cuando la casilla a la que quiere moverse está vacía o tiene una caja, pero la casilla siguiente a la caja está vacía, resultando el movimiento en el cambio de casilla tanto del jugador como de la caja. Adicionalmente, se debe intentar resolver con el menor número posible de pasos y empujes.

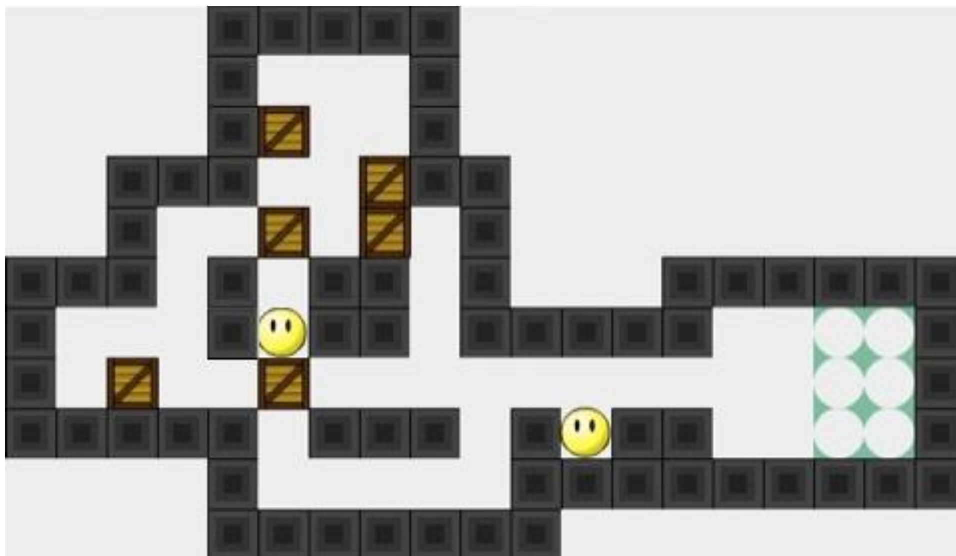


Figura 1 - Ejemplo de un problema en *Sokoban*

Como se describió anteriormente, este trabajo se aplicará sobre un entorno real. Es decir, cada uno de los jugadores será un robot y tendrá que empujar un conjunto de cajas. Para ello se modificará la estructura del dominio clásico del *Sokoban*, permitiendo que dos jugadores puedan resolver el problema de forma simultánea ejecutando algunas acciones a la vez.

Los robots utilizados en este proyecto serán del tipo Pioneer P3DX. El P3DX es uno de los robots más populares para investigación y docencia. Es una pequeña caja de unos 45x38x24cm con dos ruedas, que puede alcanzar los 1,2m/s de velocidad lineal y 300 grados/s de velocidad de rotación. Consta de sónares y detectores de colisiones (*bumpers*) que proporcionan información de su entorno [1]. En la Figura 2 se muestra una pareja de ellos.

Figura 2 - Robots P3DX

1.2 Motivación

Como se mencionó en la introducción, la robótica inteligente se encuentra, en estos momentos, en auge. Cada vez son más frecuentes las tareas que antiguamente requerían la inteligencia del ser humano pero actualmente empiezan a ser desarrolladas por robots.

La Planificación Automática es una de las técnicas más aplicadas a la implantación de esta “inteligencia” en los robots, ya que sus características se adaptan fácilmente a algo entendible por un robot, posiblemente debido a que en su origen la PA estaba diseñada para esta labor.

Pero a la robótica inteligente se le puede añadir una característica que incrementa enormemente su potencial, el hecho de ser multi-agente. Como ejemplo, el tema concreto del trabajo, el *Sokoban*, es bastante interesante en la actualidad ya que empresas realmente grandes e importantes (por ejemplo, Amazon) utilizan sistemas de este tipo para organizar sus almacenes, por lo que está totalmente a la vanguardia de la tecnología actual.

1.3 Objetivos del trabajo

El objetivo principal de este trabajo consiste en el desarrollo de un sistema que integra Planificación Automática con ROS (*Robot Operating System*) para el control de robots autónomos. Como prueba de su correcto funcionamiento, este sistema debe permitir a dos robots de tipo P3DX resolver problemas del dominio del *Sokoban*. Como objetivos específicos, se pueden contemplar los siguientes:

- Familiarización con el framework ROS: para poder llevar a cabo cualquier acción con los robots, en primer lugar se debe saber cómo se utiliza algún tipo de sistema que se conecte con ellos y tenga acceso a sus funcionalidades. El sistema elegido es ROS.
- Familiarización con el robot P3DX: aparte de ROS se deben conocer las singularidades del robot a utilizar en este trabajo, así como las posibilidades que ofrece.
- Reformulación del dominio del *Sokoban*: el dominio clásico del *Sokoban*, tal y como se contempla en la Competición Internacional de Planificación (IPC) [2] no es adecuado para este trabajo, ya que no contempla más que un único jugador, por lo que habrá que realizar pequeñas modificaciones sobre él.
- Desarrollo del sistema de control: debe existir un sistema capaz de hacer que el robot realice los movimientos que se deseen y que devuelva la información de sus sensores cuando sea requerido.
- Integración con los sistemas de PA: el problema de este trabajo requiere la PA para determinar qué acciones debe realizar el robot utilizando la información de los sensores.
- Experimentación: una vez desarrollado el sistema, se debe realizar la experimentación que determinará si el resultado ha sido satisfactorio.
- Desarrollo de la documentación: todo lo realizado durante el desarrollo del trabajo debe verse correctamente plasmado tanto en la presente memoria como en la presentación realizada ante el tribunal.

1.4 Estructura del documento

Este documento está dividido en 6 capítulos y 4 anexos. A continuación se muestra una breve descripción de cada uno de ellos:

- Capítulo 1

  En este capítulo se presenta una introducción al trabajo, así como al problema que ha sido resuelto, mostrando la motivación del trabajo, enumerando los objetivos propuestos para la realización de éste junto a una breve descripción de cada uno y, por último, realizando esta descripción de los capítulos del documento.

- Capítulo 2

  En este capítulo se presenta el estado de la cuestión en la cual puede ser enmarcado este trabajo. En primer lugar se presenta una descripción de los diferentes modelos que han sido desarrollados a los largo de los últimos siglos y que han producido la aparición de la robótica moderna. A continuación se presenta una descripción detallada de qué es la planificación automática. Para finalizar se presenta una descripción de la arquitectura PELEA que ha sido utilizada para conectar el modelo de razonamiento y el modelo de actuación, así como del framework ROS, utilizado para el control de los robots.

- Capítulo 3

  En este capítulo se presenta la descripción del sistema, separada en las descripciones del análisis y el diseño, que explicarán, respectivamente, las características que tiene que cumplir el sistema y cómo éste está formado.

- Capítulo 4

  En este capítulo se presenta la descripción de la experimentación realizada, mediante la cual podremos comprobar que se ha llegado a los objetivos fijados.

- Capítulo 5

  En este apartado se presenta la gestión del trabajo, compuesta por la planificación y el presupuesto, para dar una idea del coste del trabajo realizado.

- Capítulo 6

  En este capítulo se presentan las conclusiones obtenidas tras la realización del trabajo, así como la enumeración de posibles trabajos futuros sobre el sistema.

- Anexo A

  Este anexo contiene los pasos necesarios para instalar y configurar el entorno necesario para ejecutar el sistema.

- Anexo B

  Este anexo contiene los pasos necesarios para utilizar el sistema desarrollado.

- Anexo C

  Este anexo contiene un resumen del presente documento en inglés.

- Anexo D

  Este anexo contiene algunos de los documentos que se han utilizado a lo largo del desarrollo del trabajo (mensajes, dominios y scripts de ejecución).

Capítulo 2: Estado del Arte

En este capítulo se describe el marco teórico en el que se enmarca el trabajo que ha sido desarrollado. En primer lugar se presenta una introducción a la robótica a través de los diferentes trabajos que han sido desarrollados en los últimos 300 años. A continuación se presenta una descripción detallada de la Planificación Automática, que será utilizada como modelo de razonamiento para controlar a los diferentes robots. Para finalizar, se presenta una descripción detallada de la arquitectura PELEA, que será utilizada para conectar el sistema de control con el sistema de razonamiento, así como una descripción del framework ROS, que servirá para desarrollar el mencionado sistema de control.

2.1 Robótica

En 1921, el escritor Karel Capek estrenó su obra *Rossum's Universal Robot* (R.U.R.), donde estos robots eran máquinas fabricadas para servir a sus jefes humanos, hasta que se rebelaron y destruyeron la vida humana. Ésta fue la primera aparición de la palabra *robot*, con origen en la palabra eslava *robota*, que denomina el trabajo realizado de forma forzada [3].

A pesar de que el término *robot* es relativamente nuevo, lo que nosotros conocemos por robot lleva existiendo desde mucho antes. Los griegos acuñaron una palabra para designar las máquinas y dispositivos capaces de imitar las funciones y movimientos de los seres vivos: *automatos*, es decir, 'movimiento propio'. En castellano hemos heredado la palabra autómata, y la utilizamos para designar las máquinas que imitan los movimientos de un ser animado.

El primer libro que documenta conocimientos sobre autómatas fue escrito por Herón de Alejandría (siglo I d.C.), que diseñó varios autómatas distintos que servían básicamente como entretenimiento, como el pequeño teatro que se presenta en la Figura 3.

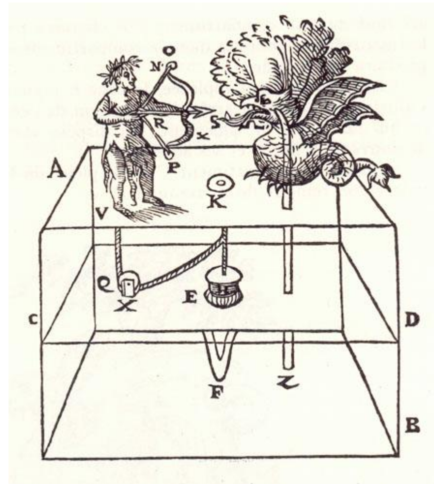


Figura 3 - Autómata teatro de Herón de Alejandría

Tras él, fueron varios quienes diseñaron sus propios autómatas. Como ejemplos más notorios, podemos mencionar el Hombre de Hierro de Alberto Magno (siglo XIII d.C.) que, según las crónicas de la época, le servía como mayordomo, siendo capaz de andar y abrir la puerta, entre otras cosas; así como los inventos de Al-Jazari (del mismo siglo, Figura 4), entre los que destaca su reloj elefante, que marcaba las horas mediante seres humanos y animales mecánicos. Cabe mencionar el autómata más antiguo que se conserva en la actualidad, el Gallo de Estrasburgo (1350), mostrado en la Figura 5. Este autómata formaba parte del reloj de la catedral de la ciudad y podía mover las alas y el pico cuando daba las horas.

Figura 4 - Reloj elefante de Al-Jazari

Figura 5 - Gallo de Estrasburgo

Uno de los hombres más influyentes de la historia de la humanidad, Leonardo Da Vinci (1452-1519), también diseñó sus propios autómatas, de los cuales sólo tenemos constancia de dos. Uno de ellos era antropomorfo, vestido con armadura y, aunque no se tiene constancia de que fuera construido en su momento, hace unos años se llevó a cabo siguiendo los diseños originales y podía realizar movimientos tales como sentarse, mover los brazos y girar la cabeza. El segundo se trataba de un león mecánico, regalo para el rey de Francia, que era capaz de moverse y abrir su pecho soltando las flores que eran símbolo de la monarquía francesa. Podemos ver en la Figura 6 un diseño actual de lo que podría haber sido este autómata.

Figura 6 - Posible diseño del león de Da Vinci

Durante los siguientes siglos se continuó con esta mecánica de construir autómatas como atracción y entretenimiento hasta que, a finales del siglo XVIII y principios del XIX, comenzaron a darse grandes avances en la industria textil (que, posteriormente, se extendieron a toda la industria) mediante los autómatas. La hiladora giratoria de Hargreaves (1770), la hiladora mecánica de Crompton (1779), el telar mecánico de Cartwright (1785) y el telar de Jacquard (1801, Figura 7) son ejemplos de ello. Este último utilizaba tarjetas perforadas para programar las acciones de la máquina; el mismo tipo de tarjetas que posteriormente se utilizarían para los computadores de mediados del siglo XX.

Figura 7 - Telar de Jacquard

A partir de este momento se comenzaron a utilizar dispositivos automáticos en la producción, comenzando así la automatización industrial. A finales de los años 50, un ingeniero de General Electric fabricó *Handyman*, que consistía en dos brazos robóticos con una precisión y fuerza increíbles. El operador manejaba una especie de exoesqueleto que transmitía su movimiento a los brazos del *Handyman*. En las Figura 8 y 9 podemos ver el exoesqueleto y los brazos, respectivamente.

Figura 8 - Exoesqueleto del *Handyman*

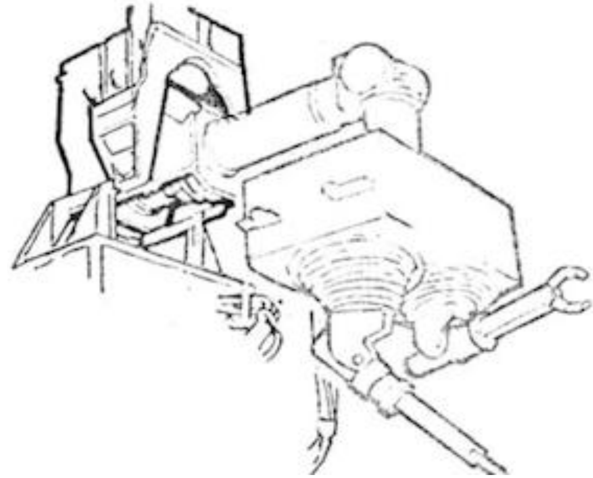


Figura 9 - Brazos del *Handyman*

Poco después, a principios de los 60, se instaló en una cadena de General Motors el primer robot industrial, *Unimate*, que realizaba tareas en la cadena de producción de los vehículos que podían ser peligrosas para los trabajadores. En la misma década se creó el robot *Shakey*, el primero que combinó razonamiento lógico con acción física [4]. Al igual que en el caso que trata este trabajo, *Shakey* utilizaba Planificación Automática para determinar sus acciones. De hecho, su aspecto es, si tenemos en cuenta la lejanía en el tiempo (casi 50 años), bastante parecido al del P3DX usado en este trabajo, como podemos ver en la Figura 10.

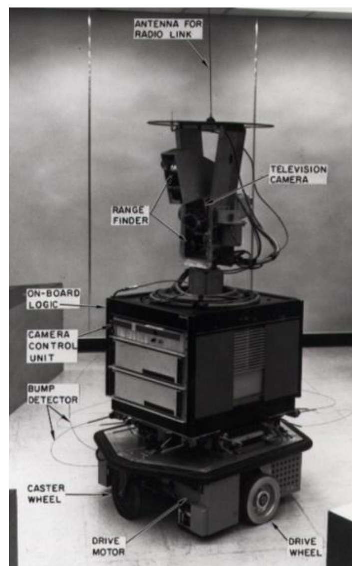


Figura 10 - Robot *Shakey*

Fue a partir de la siguiente década (1970) cuando empezó el gran "boom" de la robótica, debido a la evolución tanto en hardware como en software que ocurrió desde aquella época. Desde entonces, los robots industriales pueden encontrarse en prácticamente cualquier fábrica, pero este apartado se centrará en los robots móviles. En 1977, las sondas *Voyager* 1 y 2 fueron enviadas a los planetas exteriores para recoger información gracias a los sensores y cámaras con las que estaban equipadas. Recientemente, en el año 2013, la *Voyager* 1 se declaró como el primer objeto creado por el hombre que consigue alcanzar el espacio interestelar [5].

Avanzando dos décadas más, en 1996 la empresa japonesa Honda presentó el robot *P2*, robot bípedo con forma humana (Figura 11, tercero por la derecha), que era capaz de caminar, empujar objetos, subir o bajar escaleras y, además, lo hacía sin cables [6]. Dos años más tarde, en 1998, ve la luz un proyecto conjunto de Lego y el Instituto de Tecnología de Massachusetts (MIT, por sus siglas en inglés). Lego *Mindstorms*, destinado a niños, suele utilizarse para la docencia, ya que posee algunos de los elementos básicos de la robótica, como la programación de acciones [7]. Sólo 4 años más tarde que *P2*, Honda presentó un prototipo de robot humanoide llamado ASIMO (Figura 11, primero por la derecha), más pequeño y más avanzado, con capacidad para correr, comunicarse e interactuar con el medio [8].

Figura 11 - Evolución de los robots Honda

En 2007, el robot humanoide Nao sustituye al perro Aibo de Sony (en el mercado desde 1999) [9] como plataforma para la *Robocup* (Figura 12) [10]. Un año después, se lanzó a las instituciones con fines de investigación y docencia [11].

Figura 12 - Robots Aibo y Nao

Como último gran hito de la robótica, en 2011 la NASA lanzó al espacio el robot tipo vehículo explorador (*rover*, en inglés) *Curiosity*, que aterrizó en Marte al año siguiente. Su principal cometido es investigar la capacidad pasada y presente del planeta para alojar vida [12].

Actualmente, siguiendo en el campo de la exploración espacial, la NASA está investigando un nuevo tipo de robot que sería capaz de aterrizar en el astro destino recibiendo el mínimo daño gracias a su revolucionario diseño. Como podemos ver en la Figura 13, el *Super Ball Bot* se trata de un conjunto de componentes rígidos unidos por materiales flexibles (basado en el principio de tensegridad) en una forma similar a una esfera [13]. El objetivo de la NASA con este proyecto es llegar a Titán, una de las lunas de Saturno.

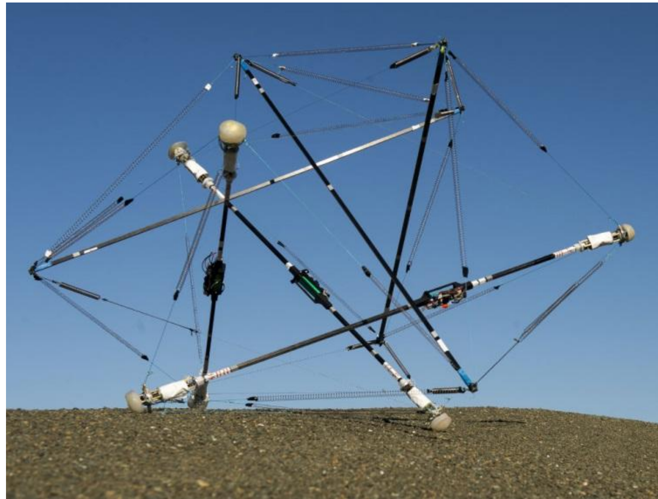


Figura 13 - Prototipo de *Super Ball Bot*

Teniendo en cuenta las diferentes características y capacidades que tienen los distintos robots presentados anteriormente, es posible agruparlos en 4 generaciones.

- Primera generación: los robots pertenecientes a esta generación son denominados robots manipuladores. Se trata de robots que no reciben ninguna información del entorno, simplemente ejecutan una secuencia de acciones preprogramada.
- Segunda generación: los robots pertenecientes a esta generación son denominados robots de aprendizaje. Los robots de esta generación obtienen cierta información de su entorno, que les permite adaptar su actuación al mismo. Estos robots pueden aprender a realizar una secuencia de movimientos mediante el seguimiento de los movimientos de un operador humano.
- Tercera generación: los robots pertenecientes a esta generación son denominados robots con control sensorizado. Los robots ahora tienen un computador que ejerce las veces de controlador y que, mediante un programa que utiliza la información obtenida por los sensores, determina las acciones a ejecutar.
- Cuarta generación: los robots pertenecientes a esta generación son denominados robots inteligentes. A lo anterior se añade el concepto de "estado del mundo" que representa lo que rodea al robot. Poseen sensores mucho más sofisticados y las estrategias para utilizar la información recibida son mucho más complejas.

A primeros del año 2014, la Organización Internacional para la Estandarización (ISO) hizo pública la norma ISO-13482 [14], que pretende definir los parámetros por los que se deben regir los robots personales, que cubren el amplio espectro de todos los robots cuya finalidad implique el cuidado y la asistencia de personas. La esperanza de que se produzca una evolución espectacular en los próximos años ha causado que aparezca esta norma. De hecho, varios hitos han tenido lugar en los últimos meses que anuncian el surgimiento de la era de los robots, como la reciente adquisición de varias empresas de robótica por parte de Google.

2.2 Planificación Automática

La Planificación Automática es un área de la IA cuya meta es obtener una secuencia de acciones para resolver un problema dado [15]. Típicamente dicha secuencia es ejecutada por un robot u otro agente. La PA utiliza dos elementos, un lenguaje de representación y un algoritmo de ejecución. El lenguaje de representación más extendido y utilizado es el denominado Planning Domain Definition Language (PDDL). Con él, podemos representar: el estado inicial del problema mediante predicados, las posibles acciones que se pueden realizar y las metas que se quieren alcanzar.

El estado inicial y las metas a alcanzar forman lo que se denomina “problema”, mientras que las posibles acciones y los predicados genéricos que pueden instanciarse para describir un determinado estado forman lo denominado “dominio”. El dominio representa lo que se puede hacer en el contexto de ese problema (existe el dominio del *Sokoban*, el dominio del mundo de los bloques, etc.), y cada uno de estos dominios puede tener infinitos problemas, que son combinaciones distintas de estados iniciales y metas.

El estado inicial y las metas se describen de la misma manera, mediante un conjunto de predicados. Un ejemplo de predicado puede ser *(en jugador1 casilla4-4)*, lo que puede querer decir que en el estado inicial de un determinado problema el jugador 1 está en la casilla 4,4; o bien que una de las metas del problema es que el jugador 1 esté en dicha casilla.

Una acción está formada por parámetros, precondiciones y efectos. Para poder ejecutar la acción, que viene determinada en gran parte por los parámetros, en el estado actual del problema

tienen que cumplirse las precondiciones y, cuando se termine de ejecutar la acción, en el estado del problema se encontrarán los efectos. Un ejemplo de acción en el propio *Sokoban* puede ser la acción *mover*. Esta acción *mover* tiene como parámetros la casilla de origen, la de destino y la dirección en la que se mueve el jugador. Las precondiciones son que el jugador esté en la casilla de origen y que la casilla de destino esté vacía y sea adyacente a la de origen por la dirección indicada. Los efectos son que la casilla de origen está vacía y el robot está en la de destino, además de las que se pueden deducir a partir de éstas, que el robot ya no está en la casilla de origen y la de destino ya no está vacía.

El objetivo del proceso de planificación será encontrar la secuencia de acciones que, partiendo del estado inicial, lleven a un estado en el que se cumplan las metas.

Repasando brevemente la historia de la PA, cabe comenzar por el primer gran sistema de planificación, STRIPS (*Stanford Research Institute Problem Solver*, 1971), que usaba un sistema de búsqueda en el espacio de estados, y que fue diseñado como el componente planificador de acciones del software del robot *Shakey*, mostrado en el apartado anterior.

Pero fue el lenguaje utilizado en STRIPS lo que realmente influyó en la planificación. En 1986 se introdujo ADL (*Action Description Language*), que tomó STRIPS como base e intentó tratar problemas más realistas. No fue hasta 1998 cuando apareció PDDL, que fue introducido como estándar para la representación de problemas en planificación y que incorpora elementos de STRIPS, ADL y otros lenguajes (de hecho STRIPS o ADL son subconjuntos de PDDL). Desde ese mismo año ha sido utilizado como estándar en las competiciones de planificación *(International Planning Competition* (IPC)), primero en el marco de la conferencia AIPS (*Artificial Intelligence Planning Systems*) y posteriormente en su nueva denominación de ICAPS *(International Conference on Automated Planning and Scheduling*) [16].

A principios de los 70, los planificadores sólo consideraban secuencias de acciones totalmente ordenadas. Descomponían el problema en sub-problemas y, tras obtener una secuencia de acciones (sub-plan) para cada sub-meta, simplemente las realizaban de forma secuencial. Pronto se descubrió que esta planificación lineal era incompleta, ya que no podía resolver problemas muy simples que necesitaban de la intercalación de las acciones de los distintos sub-planes.

Para solucionar este problema, Waldinger introdujo en 1975 la planificación de regresión de objetivos, mediante la cual se reordenan las fases de un plan totalmente ordenado para evitar conflictos entre los sub-objetivos.

Para llevar a cabo la planificación de orden parcial se necesita lograr la detección de conflictos. La construcción de planes parcialmente ordenados fue antecesora del planificador NOAH y el sistema NONLIN. Este tipo de planificación dominó los 20 años siguientes de investigación.

En 1992, Kautz y Selman propusieron el algoritmo SATPLAN, que entiende la planificación como satisfabilidad. Estos autores se inspiraron en el éxito de las búsquedas locales para satisfacer problemas.

Tres años después, en 1995, Blum y Furst introdujeron su sistema GRAPHPLAN, mucho más rápido que los planificadores de orden parcial. Pronto aparecerían otros sistemas de planificación de grafos, como IPP, STAN y SGP.

Drew McDermott publicó su UNPOP en 1996, y consiguió el resurgimiento de la planificación en espacio de estados. HSP, el planificador de búsqueda heurística de Bonet y Geffner fue el primero en intentar solucionar problemas de planificación complejos mediante búsquedas en el espacio de estados.

FASTFORWARD (FF), publicado en el año 2000 por Hoffman, fue durante mucho tiempo el planificador de búsqueda en el espacio de estados más utilizado. Ganó la competición de planificación AIPS del mismo año. FF se sirve de un algoritmo de búsqueda muy rápido combinando búsquedas locales y hacia delante, utilizando una heurística de planificación de grafos simplificada.

El reinado en las competiciones de FF terminó en el año 2008 con la llegada de Lama, de Helmert, que también ganó la competición de 2011 (desde ese año las competiciones son cada tres años en lugar de bianuales). Lama es una implementación optimizada para la competición del planificador Fast-Downward, un proyecto de código libre en el que probar distintas técnicas de planificación [17].

El planificador utilizado en este proyecto, Metric-FF, es una variante de FF que parte de la misma base pero con la posibilidad de soportar funciones numéricas, aspecto que ni Lama ni Fast-Downward soportan adecuadamente [18].

La PA es uno de los pilares de la IA y, en la actualidad, es uno de los temas más recurrentes en artículos y conferencias del sector debido a su amplia utilidad.

2.3 PELEA

PELEA (*Planning, Execution and Learning Architecture*, esto es, arquitectura de planificación, ejecución y aprendizaje) es una arquitectura genérica de planificación de código libre escrita en Java [19]. Ha sido desarrollada por tres universidades españolas, entre ellas la Universidad Carlos III de Madrid. Está especialmente enfocada al control de robots, siendo independiente del robot concreto que se utilice. También puede utilizarse con una gran variedad de tipos de planificadores (en el caso de este trabajo, como acabamos de mencionar, se utiliza Metric-FF).

Esta arquitectura permite planificar acciones y ejecutarlas, pero también monitorizar la ejecución del plan, replanificar si es necesario y aprender nuevo conocimiento para mejorar los resultados de la planificación.

Es una arquitectura multi-plataforma, con la salvedad de los módulos que deben encapsular un planificador automático y que por lo tanto funcionan en el sistema operativo en el que funcione dicho planificador.

El esquema genérico de PELEA está compuesto por ocho módulos que se intercambian información. La información intercambiada es:

- estadoL: estado de bajo nivel, generalmente obtenido a través de los sensores.
- estadoH: estado de alto nivel, se crea generalizando o agregando la información contenida en estadoL.
- Metas: metas de alto nivel que el sistema debe alcanzar.
- Métricas: medidas de rendimiento que se deben tener en cuenta al crear el plan.
- planH: acciones que forman el plan a alto nivel que consigue alcanzar las metas.

- planL: acciones a realizar a bajo nivel para ejecutar las acciones de planH. En general cada acción del planH conllevará la ejecución de varias acciones de bajo nivel.
- dominioH: definición (el modelo) de las acciones que se pueden realizar a alto nivel.
- dominioL: definición (el modelo) de las acciones que se pueden realizar a bajo nivel.
- Ejemplos de aprendizaje: compuestos por problemas, planes generados y resultado de los planes. Constituyen la entrada del módulo de aprendizaje.
- Heurísticas: conocimiento que permite al planificador encontrar planes más rápido o con mejor calidad.
- Información de monitorización: información que se utiliza para realizar la monitorización de la ejecución del plan. En general está compuesta por un conjunto de variables de bajo nivel y sus rangos de valores permitidos.

Aunque el esquema genérico tiene ocho módulos, en este trabajo se va a usar la versión de cinco. Estos módulos son:

- *Execution*. Es el módulo que se encarga de la interfaz con el robot. Recibe las acciones a bajo nivel que debe realizar el robot y se encarga de su ejecución. También recibe del robot los datos de los sensores y los coloca en el formato adecuado para cuando otro módulo (el de monitorización) se los solicita. Es el único módulo del que puede haber más de uno; de hecho, debe haber uno para cada robot.
- *Monitoring*. Se encarga principalmente de monitorizar que el estado de bajo nivel se corresponda con el esperado. Para evitar tener que monitorizar todas las variables que conforman el bajo nivel, el módulo *Decision Support* le informa de las que son relevantes para el problema. De esta forma, el *Monitoring* recibe información sobre qué variables debe monitorizar, variables que pueden pertenecer al ambiente o al plan, sus rangos válidos y los momentos temporales en que deben comprobarse. En concreto, para cada variable a monitorizar se guarda el momento en que se genera, el primer y último instante temporal en que se prevé que una acción la necesitará como precondición y su rango de valores válido. Este módulo también se encarga de enviar el plan de bajo nivel a *Execution*. Este plan puede estar formado por varias acciones a realizar en paralelo, por

lo que el *Monitoring* debe encargarse de calcular cuándo terminan de ejecutarse y pedir el estado correspondiente.

- *Decision Support*. Las dos principales actividades de este módulo son, por un lado, decidir qué variables se deben monitorizar y cuáles son sus valores válidos y, por otro, llamar al *High-level replanner* cuando el estado real no se corresponde con el esperado. En este último caso, el módulo debe decidir si la discrepancia hace el plan inválido o no. En el primer caso, debe también decidir si es mejor crear un plan desde cero (utilizando como estado inicial el observado), replanificando, o si el plan inicial se puede reparar.
- *Low-level planner*. Transforma el plan de acciones de alto nivel en un plan de acciones de bajo nivel. En muchas ocasiones esta transformación se puede hacer de manera directa, mediante, por ejemplo, una tabla de equivalencias, sin necesidad de planificar. En otros casos será necesario usar un planificador independiente del dominio para realizar la conversión.
- *LowToHigh*. Se encarga de realizar la conversión del estado de bajo nivel al de alto nivel. Recoge los datos de los sensores y los convierte en conocimiento de alto nivel.

Aunque para la definición de los dominios, los problemas y los planes PELEA utiliza lenguajes estándar de planificación, como PDDL, el intercambio de información entre todos sus módulos se realiza en XML, en un lenguaje que se ha denominado XPDDL. La transformación desde el lenguaje de planificación a XML la realiza el módulo *Decision* Support.

A continuación se va a describir el ciclo de ejecución de PELEA para la versión de cinco módulos (Figura 14). Al empezar, todos los nodos se registran en *Monitoring*. Después *Monitoring* les manda un mensaje de activación y la red empieza a funcionar. Se leen los ficheros de dominio y problema en PDDL para traducirlos, convirtiéndolos a XML y de ahí a un objeto fácil de tratar. Tras esto, *Monitoring* pide al nodo *Execution* el estado de bajo nivel actual del mundo y, tras enviarlo a *LowToHigh* y recibirlo de nuevo pero en forma de alto nivel, se lo envía junto con las metas al *Decision Support* en XPDDL. *Decision Support* busca un planificador que pueda resolver el problema. Tras ejecutar el planificador elegido, le manda un plan a *Monitoring*. Este plan está compuesto por acciones que pueden estar divididas en subacciones paralelas. Dicho plan se envía a *Low-level planner* para traducirlo a bajo nivel, y se devuelve a *Monitoring*. Una variable cursor de

infoMonitoring indica cuál es la acción que se está ejecutando actualmente. *Monitoring* la envía a *Execution* y éste la ejecuta.

Después *Monitoring* pide de nuevo a *Execution* el estado actual del mundo. A continuación compara el estado que el planificador se esperaba con el estado actual del mundo. Si son similares (la acción ha tenido el efecto esperado en el mundo) se continúa con la siguiente acción del plan y se actualiza el cursor. Si la acción no ha tenido el efecto esperado se inicia la replanificación. *Monitoring* pide al *Decision Support* un plan reparado enviándole el trozo de plan que ya lleva ejecutado y el estado del mundo recibido. Este nodo usará un replanificador de su lista y en caso de no encontrarlo usará un planificador normal.

Este proceso continúa hasta que se hayan cumplido todas las metas.

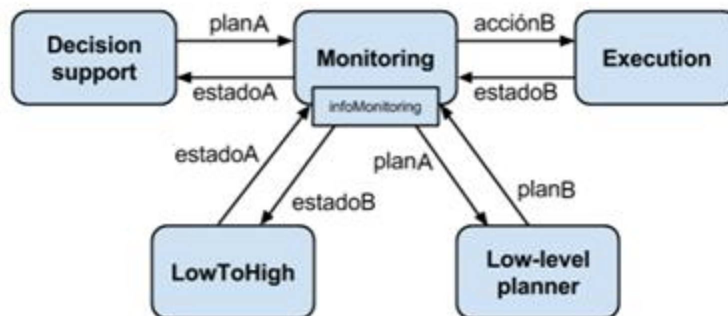


Figura 14 - Esquema de PELEA de cinco módulos

2.4 ROS

Los sistemas operativos robóticos nacen para poder controlar de una manera más avanzada y sencilla los movimientos y funcionalidades de los robots. El más usado y conocido es ROS, pero existen otros como *Robocomp* [20] o *Player* [21].

Aunque en un principio se pretendía utilizar *Robocomp*, la idea fue descartada debido a la gran cantidad de fallos que tenía y la falta de documentación. Por ello, finalmente se eligió ROS, globalmente conocido y utilizado.

ROS, del inglés *Robot Operating System* (sistema operativo robótico), es un conjunto de librerías que permiten escribir software que pueda ser utilizado por un robot [22]. ROS tiene soporte para gran variedad de robots, entre ellos el P3DX de este trabajo.

ROS utiliza un sistema basado en nodos, mensajes y *topics*. Los nodos son unidades de computación que ejecutan una tarea simple. Un nodo puede enviar o recibir mensajes, que pueden ser tipos primitivos (*integer*, *float*, etc.) o estructuras de datos formadas por una combinación de los anteriores. Para ello, un nodo debe suscribirse o publicar en un *topic*. El *topic* hace las veces de canal de comunicación. Todo mensaje publicado en él será recibido por todo nodo que este suscrito a él. En la Figura 15 podemos ver un diagrama muy simple de esta comunicación.

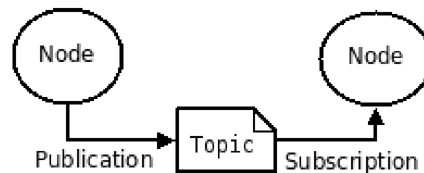


Figura 15 - Grafo funcionamiento de ROS

Por poner un ejemplo, el *topic* puede ser '*velocidadrobot*', y el mensaje puede publicarlo un nodo controlador con el contenido de la velocidad y dirección en la que se quiere que se mueva el robot para que el propio robot, que tendrá un nodo suscrito a dicho *topic*, lo reciba y cambie su velocidad a la indicada.

Para que esto pueda llevarse a cabo es necesario un nodo maestro, que es quien se ocupa de registrar qué nodo está registrado/publica en qué *topic*.

Los mensajes también pueden ser definidos por el usuario, formando lo que se llama *mensajes personalizados* (del inglés *custom messages*), que pueden agrupar uno o varios de cualquiera de los demás tipos de mensaje.

Aunque ROS tiene más elementos, estos son los más básicos y los únicos que se utilizarán en el trabajo. Mediante ROS se lleva a cabo toda la parte de control del robot, esto es, el movimiento y la obtención de información de los sensores.

ROS se organiza mediante paquetes (*packages*) y pilas (*stacks*). Los paquetes son conjuntos de nodos que, juntos, llevan a cabo pequeñas funcionalidades; mientras que las pilas son conjuntos de paquetes que abarcan una funcionalidad completa.

A pesar de que ROS tiene soporte nativo para C++ y Python, existe la posibilidad de utilizar una implementación en Java llamada rosjava, que ha sido elegida debido a que, como se ha comentado en el apartado anterior, PELEA también está escrito en Java y la comunicación entre ambos sistemas se verá facilitada si ambos están codificados en el mismo lenguaje.

2.4.1 Programación en rosjava

Como se verá en el siguiente capítulo del documento, el módulo de control del sistema se ha desarrollado utilizando rosjava, una implementación de ROS en Java.

Se eligió utilizar rosjava debido a la experiencia del programador con el lenguaje y el hecho de que la arquitectura de control Pelea también está escrita en Java y se pensó que sería conveniente trabajar con el mismo lenguaje. A pesar de ello, finalmente la conexión entre módulos se ha hecho por TCP/IP, por lo que la coincidencia en lenguaje no ha sido especialmente útil en ese sentido, pero sí por la comentada experiencia del programador y por conocer y utilizar una implementación “nueva” de ROS.

Al principio, el hecho de que rosjava sea relativamente nuevo provocó algunas dificultades ya que no hay mucho soporte, especialmente en comparación con ROS en C++ o Python, sus lenguajes nativos. Pero, a pesar de estas complicaciones iniciales, finalmente se descubrió que, aparte del hecho de que para ejecutar y compilar módulos en Java hay que utilizar comandos distintos que para los módulos en lenguajes nativos, el contenido de las clases es muy parecido, obviando las diferencias entre distintos lenguajes, por supuesto.

Capítulo 3: Descripción del sistema

En este capítulo se muestra la descripción del sistema que se ha desarrollado para este trabajo. Es un sistema que permite la resolución de problemas multi-agente del *Sokoban* en el mundo real con robots P3DX, utilizando el sistema operativo ROS y la arquitectura de planificación PELEA. La Figura 16 muestra el esquema más general posible de dicho sistema.

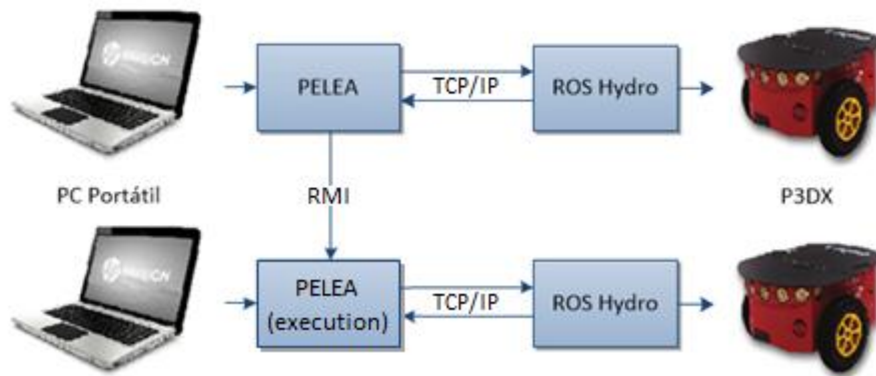


Figura 16 - Esquema general de la estructura del sistema

En el esquema se muestran tan solo dos robots, cada uno controlado por una instancia del nodo *Execution* de PELEA. Uno de ellos se encuentra corriendo en el mismo ordenador que el resto de PELEA, mientras que el otro se encuentra en un ordenador distinto y se comunica con los otros módulos de PELEA mediante RMI (*Remote Method Invocation*). Debido a que los robots P3DX no tienen procesador interno es necesario un ordenador conectado mediante serie-USB para controlar cada uno.

El esquema anterior se puede extender para el control de cualquier número de robots P3DX, sin necesidad de cambiar más que el archivo de configuración que especifica el número de robots (nodos *Execution*) presentes. En este TFG se han realizado pruebas reales con solo 2 robots dado que es el número del que dispone el grupo de investigación donde se ha desarrollado.

A continuación, se muestra el proceso del análisis y el diseño que se ha seguido para el desarrollo del sistema del presente trabajo.

3.1 Análisis del sistema

En este apartado se presenta la descripción de las características funcionales, seguida de la descripción de las restricciones del sistema y del entorno operacional y, por último, la especificación de los casos de uso y de los requisitos del sistema.

3.1.1 Descripción de las características funcionales

El sistema desarrollado en este TFG debe ser capaz de solucionar problemas multi-agente del dominio del *Sokoban* con robots físicos P3DX. Para ello, debe tener dos funcionalidades principales.

La primera es el control de los robots. El sistema debe ser capaz de controlar los movimientos de los robots (avanzar y girar) y recuperar la información de sus sensores (lecturas de los sónar, información de odometría (posición y velocidad), etc.).

La segunda es la obtención de las acciones a realizar para solucionar el problema. El sistema debe ser capaz de leer los archivos PDDL de dominio y problemas, generar una secuencia de acciones que los resuelva y, tras la ejecución de cada acción por el módulo de control, comprobar que el resultado de la acción es el esperado.

Además, al dominio clásico del *Sokoban* se debe sumar la posibilidad de que varios jugadores solucionen el problema juntos. Esto se ha realizado añadiendo el objeto “caja doble” o “caja grande”, que tiene las particularidades de ocupar dos casillas adyacentes y necesitar dos robots para moverla. Este objeto se ha añadido debido a que el dominio del *Sokoban* con la simple inclusión de varios jugadores es muy parecido al *Sokoban* de un jugador, ya que en general no requiere de la colaboración de los jugadores para resolver los problemas; de esta forma se muestra realmente el potencial de un sistema multi-agente en el que los robots tienen que realizar acciones simultáneas.

3.1.2 Restricciones del sistema

En este apartado se presentan las restricciones del sistema. Éstas pueden ser de dos tipos, las impuestas por el hardware utilizado y las impuestas por el software empleado para el desarrollo del sistema.

3.1.2.1 Restricciones hardware

Producidas por los dispositivos hardware utilizados en el trabajo, son las siguientes:

- El robot P3DX necesita tener batería cargada para encender el motor que le permite realizar los movimientos necesarios.
- Son necesarios dos robots P3DX para este trabajo, debido a que el sistema es un sistema multi-agente.
- Son necesarios dos ordenadores, uno para controlar cada robot, debido a que cada ordenador sólo puede controlar un robot a la vez porque deben estar conectados por cable.
- La conexión entre ordenadores y robots se realiza mediante cable, por lo que los ordenadores tienen que estar encima de los robots durante la ejecución del sistema para que no se produzca ninguna desconexión u otro accidente.

3.1.2.2 Restricciones software

Producidas por el software utilizado en el trabajo, es decir, sistema operativo, lenguajes y entornos de programación, y frameworks, sistemas y arquitecturas utilizados.

- El framework utilizado para el desarrollo del módulo de control es ROS.
- Se utiliza el sistema de repositorios git para obtener algunas partes de ROS.
- La pila de ROS utilizada para el control del robot es P2OS, que proporciona un sencillo control sobre todos sus movimientos y sensores, excepto los detectores de colisiones o *bumpers*, por lo que no han podido utilizarse para este trabajo.
- Se utiliza la implementación de ROS en Java, rosjava, que permite programar para ROS en dicho lenguaje.

- La arquitectura de planificación utilizada es PELEA, debido a su facilidad de integración con robots y el lenguaje en el que está escrita.
- El lenguaje de programación utilizado para el módulo de control es Java, por motivos de preferencia del programador y por compatibilidad con la arquitectura PELEA, escrita en Java.
- El entorno de programación utilizado para trabajar con PELEA y el módulo de control es NetBeans.
- El kit de desarrollo de Java (JDK, del inglés *Java Development Kit*) debe utilizarse, debido a que todo el sistema está escrito en Java.
- El sistema debe ejecutarse en un sistema operativo Ubuntu por requerimientos de ROS. La versión utilizada ha sido la última versión LTS (*Long Term Support*, soporte a largo plazo; suelen ser las más estables) en la fecha de inicio del trabajo, Ubuntu 12.04.
- También el planificador elegido (Metric-FF) requiere Linux para su ejecución, pero se considera una restricción menor ya que si se hubiera tenido que trabajar con Windows existen planificadores que corren en este Sistema Operativo.

3.1.3 Entorno operacional

En este apartado se describen todos los elementos necesarios para el desarrollo del sistema. Dicho entorno se muestra, para un único robot, en la Figura 17. Para el sistema completo (con dos robots), todo el entorno debería duplicarse, a excepción de lo necesario para el desarrollo del sistema: NetBeans y git.

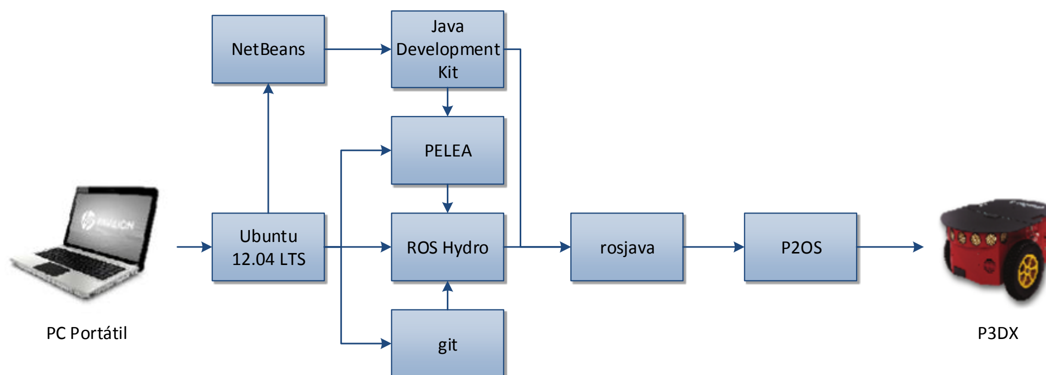


Figura 17 - Diagrama del entorno operacional del sistema

3.1.3.1 Entorno operacional hardware

Los dispositivos hardware necesarios para el correcto funcionamiento del sistema son los siguientes:

- Dos robots P3DX: este robot, descrito en el apartado 1.1 del presente documento, por sus características únicamente permite realizar movimientos simples (avanzar y girar) y dispone de sensores que proporcionan su odometría, lecturas de los sónares, información de batería, estado del motor y activaciones de los detectores de colisiones. Necesitan un puerto serie para ser conectados a un PC.
- Dos PCs portátiles: uno para controlar cada robot, no necesitan ninguna característica especial, más allá del software. Los robots se conectan por USB, por lo que deben disponer de este puerto.

3.1.3.2 Entorno operacional software

Los sistemas software necesarios para el desarrollo y utilización del sistema son los siguientes:

- Sistema Operativo Ubuntu 12.04 LTS.
- *Framework* ROS, versión Hydro.
- P2OS, pila de ROS para el control de robots compatibles, para Hydro.
- Implementación en java de ROS (rosjava) para Hydro.
- Arquitectura de control PELEA, versión de 5 nodos.
- Sistema de repositorios git para Ubuntu versión 1.8.1.2 o superior.
- Entorno de desarrollo (IDE) NetBeans versión 8.0 o superior.
- Kit de desarrollo de Java (JDK) versión 7 o superior.

3.1.4 Especificación de casos de uso

En este apartado se presentan los casos de uso del sistema, en primer lugar, mediante un diagrama, en la Figura 18. A continuación, se describen los actores que interactúan en ellos y,

finalmente, se detallan los casos de uso mediante tablas, habiendo definido previamente el formato de dichas tablas.

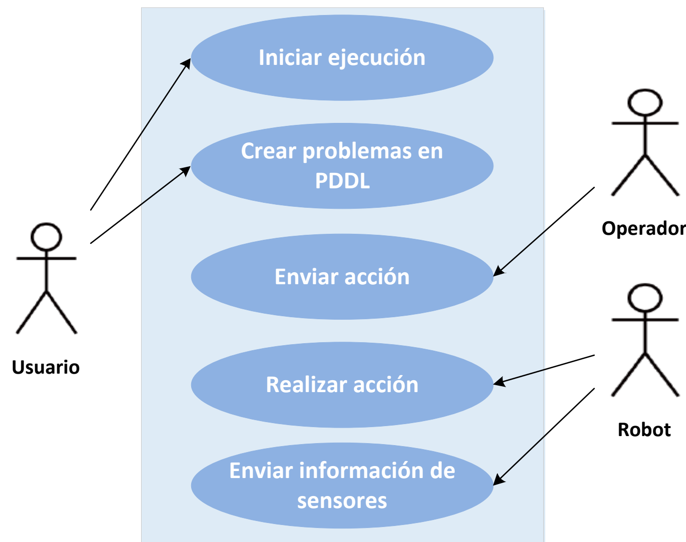


Figura 18 - Diagrama de casos de uso

3.1.4.1 Descripción de los actores

Los actores que participan en los casos de uso del sistema son los siguientes:

- Operador: Se trata del módulo de planificación (PELEA), que es quien decide las acciones que realiza el robot.
- Robot: Se trata del conjunto que forman el módulo de control y el propio robot; son los que realizan las acciones necesarias. En el caso del presente trabajo, normalmente habrá dos actores de este tipo.
- Usuario: Es la persona que inicia la ejecución del sistema para que puedan llevarse a cabo las demás tareas, y la encargada de crear los problemas en PDDL.

3.1.4.2 Descripción de los atributos de los casos de uso

Para la realización de la descripción textual de los distintos casos de uso, se han seleccionado una serie de atributos que describen cada uno de los casos de uso. A continuación se realiza una descripción del significado de cada uno de los atributos utilizados:

- Código: Identificación unívoca abreviada del caso de uso, se construye mediante CU seguido de un - y de tres dígitos. Por ejemplo CU-001.
- Nombre: Identificación extendida del caso de uso.
- Actores: Conjunto de entidades que interactúan con el caso de uso. El caso de uso representa una funcionalidad demandada por un actor.
- Descripción: Se realiza una descripción básica de la funcionalidad o funcionalidades del caso de uso.
- Precondiciones y efectos: Se realiza una descripción de las condiciones que deben cumplirse para poder realizar una operación, y el estado en el que queda el sistema tras realizar una operación.
- Escenario: Se realiza una descripción básica de las acciones que se ejecutarán paso a paso en el caso de uso.

3.1.4.3 Descripción textual de los casos de uso

En este apartado se describen, mediante las siguientes tablas, los casos de uso del sistema.

Código	**CU-001**
Nombre	Crear problemas en PDDL.
Actores	Usuario.
Descripción	El usuario recoge las características del problema y, siguiendo las restricciones del dominio, lo modela en PDDL.
Precondiciones	• Ninguna.
Efectos	• Problema en PDDL generado.
Escenario	• El usuario recoge las características que tiene el problema a resolver del dominio del *Sokoban*. • El usuario modela dichos problemas en PDDL teniendo en cuenta todas las restricciones existentes.

Tabla 1 - Caso de uso CU-001

Código	**CU-002**
Nombre	Iniciar ejecución.
Actores	Usuario.
Descripción	El usuario inicia la ejecución del robot y de los módulos de control y de planificación para que el sistema comience a funcionar.
Precondiciones	• El problema en PDDL debe haber sido creado.
Efectos	• La ejecución del sistema ha comenzado.
Escenario	• El usuario enciende el robot. • El usuario inicia el módulo de control. • El usuario inicia el módulo de planificación.

Tabla 2 - Caso de uso CU-002

Código	**CU-003**
Nombre	Enviar información de sensores.
Actores	Robot.
Descripción	El robot envía la información de sus sensores al operador.
Precondiciones	• El operador debe haber pedido la información de sensores (cosa que ocurre al iniciar la ejecución o tras enviar cada acción a realizar).
Efectos	• La información de los sensores ha sido recibida por el operador.
Escenario	• El robot recoge la información de sus sensores en un objeto. • El robot envía dicho objeto al operador.

Tabla 3 - Caso de uso CU-003

Código	**CU-004**
Nombre	Enviar acción.
Actores	Operador.
Descripción	El operador envía al robot la siguiente acción a realizar.
Precondiciones	• El operador debe haber generado la secuencia de acciones de bajo nivel. • El operador debe haber recibido del robot la información de los sensores.
Efectos	• La acción a realizar ha sido enviada.
Escenario	• El operador selecciona la primera acción de la secuencia que aún no ha sido enviada. • El operador envía dicha acción al robot.

Tabla 4 - Caso de uso CU-004

Código	**CU-005**
Nombre	Realizar acción.
Actores	Robot.
Descripción	El robot realiza la acción enviada por el operador.
Precondiciones	• El operador debe haber enviado la acción.
Efectos	• La acción ha sido realizada.
Escenario	• El robot recibe la acción. • El robot interpreta qué tipo de acción tiene que realizar (avanzar o girar). • El robot lleva a cabo la acción.

Tabla 5 - Caso de uso CU-005

3.1.5 Especificación de requisitos

En el presente apartado se muestran los requisitos del sistema. En primer lugar, se definen los atributos que componen dichos requisitos para, finalmente, presentar textualmente los requisitos, tanto funcionales como no funcionales.

3.1.5.1 Descripción de los atributos de los requisitos

Para la realización de la descripción textual de los distintos requisitos que han sido identificados, se han seleccionado una serie de atributos que describen cada uno de los requisitos. A continuación se realiza una descripción del significado de cada uno de los atributos utilizados para su descripción:

- Código: Identificación unívoca abreviada del requisito, se construye mediante el código del requisito seguido de un - y de tres dígitos. Los requisitos serán divididos en funcionales y no funcionales y sus códigos son RF para los requisitos funcionales y RNF para los requisitos no funcionales. Por ejemplo RF-001.
- Nombre: Identificación extendida del requisito.
- Descripción: Se realiza una descripción básica del requisito que ha sido identificado.
- Fuente: Indica a través de qué fuente ha sido identificado el requisito. Normalmente este valor se corresponderá con uno o varios códigos de los casos de uso.
- Necesidad: Determina el grado de implementación del requisito. Los valores que puede tomar este atributo son los siguientes:
    - Esencial: El requisito tiene que ser implementado.
    - Deseable: Es preferible implementar el requisito, pero no es obligatorio.
    - Opcional: El requisito se podrá implementar, pero no es importante ni obligatorio.
- Prioridad: Define la importancia del requisito, de forma que permita definir el orden en el cual será incluido en el proceso de diseño y el orden de implementación. Los valores que puede tomar este atributo son los siguientes:
    - Alta: El requisito debe ser implementado en las fases iniciales del desarrollo.

    - Media: El requisito debe ser implementado una vez que hayan sido implementados los requisitos de prioridad alta.
    - Baja: El requisito debe ser implementado en las fases finales del desarrollo. Estos requisitos no influirán en el correcto funcionamiento del sistema.
- Estabilidad: Define la estabilidad del requisito durante la vida útil del software. Esto implica si el requisito podrá ser o no modificado durante el ciclo del vida. Los valores que puede tomar este atributo son los siguientes:
    - Estable: El requisito no puede variar durante el ciclo de vida del sistema.
    - Inestable: El requisito puede variar a lo largo de la ciclo de vida del sistema.
- Verificabilidad: Define el grado de verificabilidad de un requisito, es decir indica en qué grado es posible comprobar que el requisito se ha incorporado en el sistema desarrollado. Los valores que puede tomar este atributo son los siguientes:
    - Alta: Se puede verificar que el requisito ha sido implementado en el sistema. Este tipo de requisitos se corresponden con las funcionalidades básicas del sistema.
    - Media: Se puede verificar que el requisito ha sido implementado en el sistema. Pero requiere de una comprobación compleja o del código fuente del sistema.
    - Baja: Es difícil verificar si el requisito ha sido implementado en el sistema o en algunos casos no es posible.

3.1.4.3 Descripción textual de los requisitos

En este apartado se presentan, mediante las siguientes tablas, los requisitos funcionales y no funcionales del sistema.

3.1.4.3.1 Requisitos Funcionales

A continuación, se describen los requisitos funcionales.

Código	RF-001	**Fuente**	Caso de uso CU-002.
Nombre	Scripts para el inicio de la ejecución.		
Descripción	Deben existir scripts que faciliten la ejecución de todos los elementos de los módulos de control y planificación.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 6 - Requisito del sistema RF-001

Código	RF-002	**Fuente**	Caso de uso CU-002.
Nombre	Ejecución sin interacción por el usuario.		
Descripción	Una vez comenzada la ejecución, el usuario no necesita realizar ninguna interacción más con el sistema para su correcto funcionamiento.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 7 - Requisito del sistema RF-002

Código	RF-003	**Fuente**	Caso de uso CU-002.
Nombre	Conexión continua entre módulos.		
Descripción	El módulo de control y el de planificación deben estar conectados durante toda la ejecución del sistema.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 8 - Requisito del sistema RF-003

Código	RF-004	**Fuente**	Caso de uso CU-003.
Nombre	Recoger información de los sensores.		
Descripción	El módulo de control debe ser capaz de recoger la información de los sensores del robot.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 9 - Requisito del sistema RF-004

Código	RF-005	**Fuente**	Caso de uso CU-003.
Nombre	Enviar información de los sensores.		
Descripción	El módulo de control debe ser capaz de enviar la información de los sensores al módulo de planificación cuando corresponda (al inicio de la ejecución o tras cada acción).		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 10 - Requisito del sistema RF-005

Código	RF-006	**Fuente**	Caso de uso CU-003.
Nombre	Traducción de información de los sensores en XML.		
Descripción	El módulo de planificación debe ser capaz de traducir la información de los sensores a XML.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 11 - Requisito del sistema RF-006

Código	RF-007	**Fuente**	Caso de uso CU-003.
Nombre	Traducción de información de los sensores en alto nivel.		
Descripción	El módulo de planificación debe ser capaz de traducir la información de los sensores a alto nivel.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 12 - Requisito del sistema RF-007

Código	RF-008	**Fuente**	Caso de uso CU-004.
Nombre	Traducción de acciones a bajo nivel.		
Descripción	El módulo de planificación debe poder traducir las acciones a bajo nivel, de forma que sean directamente entendibles por los robots.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 13 - Requisito del sistema RF-008

Código	RF-009	**Fuente**	Caso de uso CU-004.
Nombre	Envío de la acción.		
Descripción	El módulo de planificación debe poder enviar la acción a los robots a través de la conexión existente entre dicho módulo y ellos.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 14 - Requisito del sistema RF-009

Código	RF-010	Fuente	Caso de uso CU-005.
Nombre	Recepción de la acción.		
Descripción	El módulo de control debe ser capaz de recibir la acción enviada por el módulo de planificación.		
Necesidad	Esencial.	Prioridad	Alta.
Estabilidad	Estable.	Verificabilidad	Alta.

Tabla 15 - Requisito del sistema RF-010

Código	RF-011	Fuente	Caso de uso CU-005.
Nombre	Interpretación de la acción.		
Descripción	El módulo de control debe ser capaz de interpretar qué acción se le está pidiendo y si dicha acción es para su robot.		
Necesidad	Esencial.	Prioridad	Alta.
Estabilidad	Estable.	Verificabilidad	Alta.

Tabla 16 - Requisito del sistema RF-011

Código	RF-012	Fuente	Caso de uso CU-005.
Nombre	Avanzar.		
Descripción	El robot debe ser capaz de avanzar una distancia específica a una velocidad específica.		
Necesidad	Esencial.	Prioridad	Alta.
Estabilidad	Estable.	Verificabilidad	Alta.

Tabla 17 - Requisito del sistema RF-012

Código	RF-013	**Fuente**	Caso de uso CU-005.
Nombre	Girar.		
Descripción	El robot debe ser capaz de girar en ángulo recto a una velocidad específica.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 18 - Requisito del sistema RF-013

Código	RF-014	**Fuente**	Caso de uso CU-005.
Nombre	Segundo nodo *execution* en remoto.		
Descripción	El módulo de planificación debe ser capaz de ejecutar un segundo nodo *execution*, para el segundo robot, en un PC remoto.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 19 - Requisito del sistema RF-014

Código	RF-015	**Fuente**	Caso de uso CU-005.
Nombre	Movimientos simultáneos.		
Descripción	Los robots deben ser capaces de realizar ciertos movimientos simultáneamente (empuje de cajas dobles).		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 20 - Requisito del sistema RF-015

Código	RF-016	**Fuente**	Caso de uso CU-005.
Nombre	Cerrar conexión.		
Descripción	El módulo de control debe ser capaz de cerrar la conexión (el *socket*) con el módulo de planificación al terminar la ejecución de todas las acciones.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 21 - Requisito del sistema RF-016

Código	RF-017	**Fuente**	Cliente.
Nombre	Uso con más de dos robots.		
Descripción	El sistema debe ser fácilmente escalable para su uso con más de dos robots.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 22- Requisito del sistema RF-017

3.1.4.3.2 Requisitos No Funcionales

A continuación, se describen los requisitos no funcionales.

Código	RNF-001	**Fuente**	Analista.
Nombre	Sistema operativo Ubuntu 12.04 LTS.		
Descripción	El sistema debe funcionar en el sistema operativo Ubuntu 12.04 LTS.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 23 - Requisito del sistema RNF-001

Código	RNF-002	**Fuente**	Analista.
Nombre	Un PC por cada robot.		
Descripción	El sistema necesita un PC por cada robot que se utilice en la ejecución.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 24 - Requisito del sistema RNF-002

Código	RNF-003	**Fuente**	Analista.
Nombre	Lenguaje java en el módulo de control.		
Descripción	El módulo de control debe estar escrito en Java utilizando la implementación rosjava.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 25 - Requisito del sistema RNF-003

Código	RNF-004	**Fuente**	Cliente.
Nombre	Lenguaje java en el módulo de planificación.		
Descripción	El módulo de planificación debe estar escrito en Java.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 26 - Requisito del sistema RNF-004

Código	RNF-005	**Fuente**	Cliente.
Nombre	ROS para el módulo de control.		
Descripción	El módulo de control debe implementarse mediante el sistema operativo robótico ROS.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 27 - Requisito del sistema RNF-005

Código	RNF-006	**Fuente**	Analista.
Nombre	Comunicación en el módulo de control.		
Descripción	La comunicación interna del módulo de control se debe llevar a cabo mediante el sistema de publicación y suscripción a *topics* por los nodos de ROS.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 28 - Requisito del sistema RNF-006

Código	RNF-007	**Fuente**	Analista.
Nombre	Mensajes personalizados en ROS.		
Descripción	Para la transmisión de información de ROS se deben utilizar mensajes personalizados con la información que el analista vea más conveniente.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 29 - Requisito del sistema RNF-007

Código	RNF-008	**Fuente**	Cliente.
Nombre	PELEA para el módulo de planificación.		
Descripción	El módulo de planificación debe implementarse mediante la arquitectura de planificación PELEA (versión de 5 módulos).		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 30 - Requisito del sistema RNF-008

Código	RNF-009	**Fuente**	Analista.
Nombre	Metric-FF para la planificación en PELEA.		
Descripción	El planificador utilizado por PELEA debe ser Metric-FF.		
Necesidad	Opcional.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 31 - Requisito del sistema RNF-009

Código	RNF-010	**Fuente**	Analista.
Nombre	PDDL para los dominios y problemas.		
Descripción	Los dominios y problemas a resolver por el sistema deben estar escritos en lenguaje PDDL.		
Necesidad	Esencial.	**Prioridad**	Alta.
Estabilidad	Estable.	**Verificabilidad**	Alta.

Tabla 32 - Requisito del sistema RNF-010

Código	RNF-011	**Fuente**	Analista.
Nombre	Comunicación en el módulo de planificación.		
Descripción	Los nodos del módulo de planificación se conectarán mediante RMI, tanto los locales como los remotos.		
Necesidad	Opcional.	**Prioridad**	Opcional.
Estabilidad	Estable.	**Verificabilidad**	Estable.

Tabla 33 - Requisito del sistema RNF-011

Código	RNF-012	**Fuente**	Analista.
Nombre	Comunicación entre módulos de control y planificación.		
Descripción	Los módulos de control y planificación se conectarán mediante una conexión TCP/IP.		
Necesidad	Opcional.	**Prioridad**	Opcional.
Estabilidad	Estable.	**Verificabilidad**	Estable.

Tabla 34 - Requisito del sistema RNF-012

Código	RNF-013	**Fuente**	Analista.
Nombre	Conexión entre PC y robot.		
Descripción	Los robots deben conectarse a sus PCs correspondientes utilizando un cable de serie (robot) a USB (PC).		
Necesidad	Esencial.	**Prioridad**	Opcional.
Estabilidad	Estable.	**Verificabilidad**	Estable.

Tabla 35 - Requisito del sistema RNF-013

3.2 Diseño del sistema

En el presente apartado se presenta el diseño del sistema. Se comenzará realizando una descripción general del sistema, para posteriormente ahondar más en los componentes que lo forman y, finalmente, mostrar el funcionamiento de la ejecución del sistema completo.

3.2.1 Descripción general del sistema

El sistema está formado por dos módulos principales: el módulo de control y el módulo de planificación.

El módulo de control se encarga de tratar directamente con el robot, obteniendo los datos de sus sensores y haciéndole moverse de la forma que se requiera. Por su parte, el módulo de planificación se ocupa de proporcionar las acciones correspondientes al módulo de control. Para ello, obtiene una secuencia de acciones mediante la resolución del problema dado por el estado del mundo generado a partir de dicho problema en PDDL y de los datos de los sensores proporcionados por el módulo de control. Tras esto, envía cada acción al módulo de control para que la realice y, cuando el módulo de planificación recibe un aviso de que el módulo de control ya ha terminado, le manda la siguiente; y así sucesivamente hasta completar el plan. En caso de que una falle, el módulo de planificación crea un nuevo plan que resuelve el problema producido al encontrarse un imprevisto en el estado del mundo.

En la Figura 19 se puede ver el diagrama general de la arquitectura del sistema. En ella se muestran los dos módulos mencionados y sus componentes principales, así como los elementos que se intercambian entre ellos. Como se puede ver, hay 5 elementos principales que transitan entre los distintos módulos y sub-módulos del sistema. Estos elementos pueden tomar distintas formas tras las transformaciones que llevan a cabo cada uno de los módulos y sub-módulos sobre ellos, que se detallarán en el siguiente apartado del documento. Aquí sólo se describirán los elementos a grandes rasgos, de la siguiente forma:

- Estado: el estado actual del problema, esto es, los objetos existentes, la situación de todos ellos en el mapa, y otros datos relevantes para la resolución del problema.

- Plan: la secuencia de acciones generada, que llega al estado meta deseado partiendo del estado inicial.
- Acción: cada una de las acciones del plan.
- Movimiento: una acción de bajo nivel que consiste en un movimiento del robot, puede ser un avance o un giro, y consta de una distancia/ángulo y de un tiempo y distancia, o velocidad.
- Sensores: los datos que ofrece el robot de su entorno; odometría, lecturas de sónar, etc.

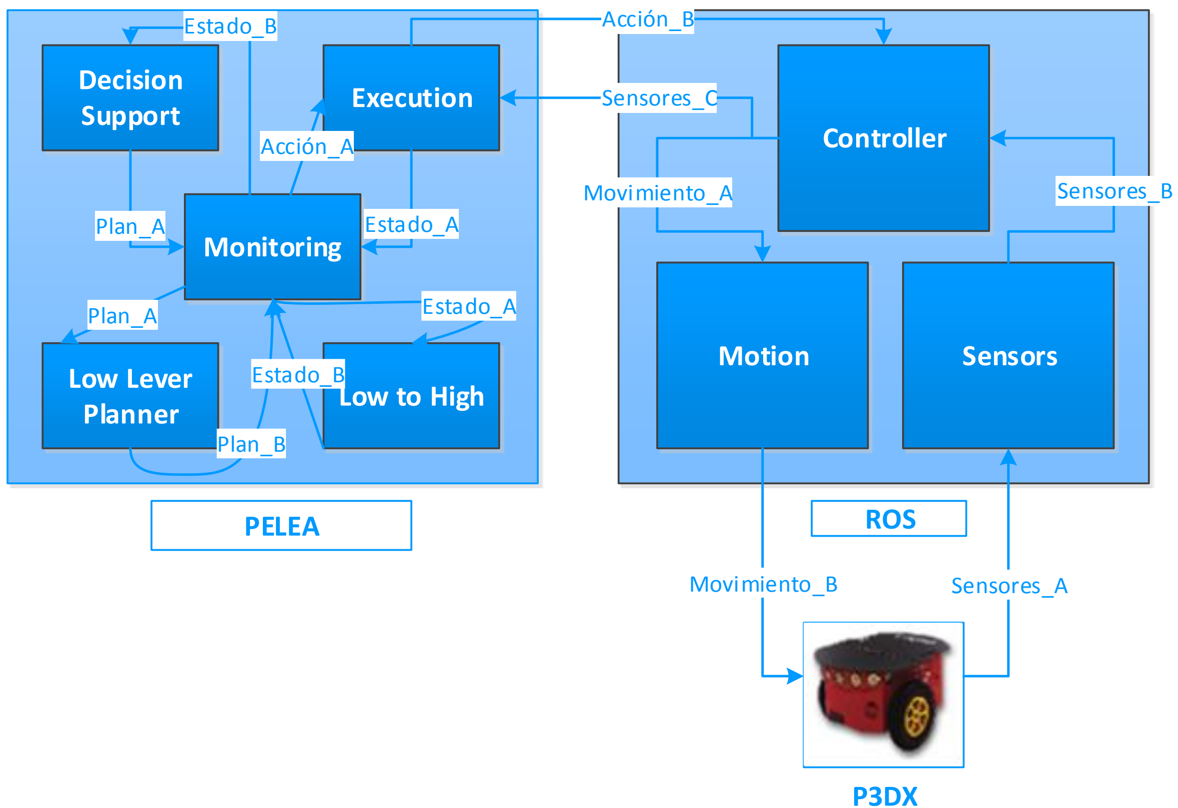


Figura 19 - Diagrama general de la arquitectura del sistema

3.2.2 Descripción de componentes

En este apartado se describen detalladamente los componentes del sistema que, como se ha mencionado anteriormente, son dos: el módulo de control y el de planificación.

3.2.2.1 Módulo de control

El módulo de control ha sido desarrollado en ROS ya que, como se comentó anteriormente en el documento, la primera alternativa contemplada (Robocomp) no cumplía los requisitos de documentación y soporte para el programador. Por tanto, el módulo de control hace uso del sistema de comunicación interna de ROS basado en *topics*. Este módulo consta de tres partes principales: el sub-módulo controlador, el sub-módulo que se ocupa de los movimientos y el sub-módulo que se ocupa de los sensores. Cada uno de ellos es una clase Java. Se puede ver esta arquitectura en la Figura 20.

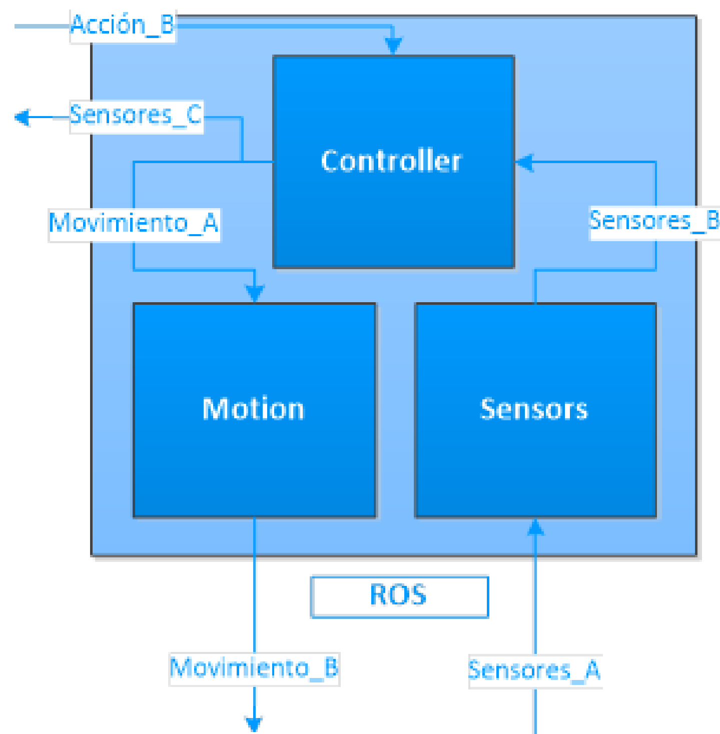


Figura 20 - Diagrama del módulo de control

A continuación se hace una descripción general de cada uno de los tres elementos que componen el módulo de control:

- *Motion*: es el sub-módulo que se ocupa de los movimientos del robot. Publica continuamente (cada 10 ms mediante un *sleep*, ya que las acciones que se publican

duran más que eso, y además no es demasiado tiempo como para que el retardo al recibir una nueva acción se note, pero el *sleep* es necesario porque sin él se "saturan" los canales de mensajes; en los tres sub-módulos se ha utilizado este *sleep* por los mismos motivos) en el *topic* que manda los movimientos al robot el movimiento que tiene que realizarse en cada momento (lo llamado en el diagrama Movimiento_B). Por defecto, este movimiento es quedarse quieto, pero variará cuando lo determine el controlador.

- *Sensors*: es el sub-módulo encargado de los sensores del robot. Constantemente recopila la información de cuatro datos distintos del robot (odometría, lecturas de sónares, nivel de batería y estado del motor; esto es, Sensores_A), los une en un único objeto y los publica (Sensores_B), también constantemente, para que el controlador pueda disponer de ellos.
- *Controller*: es el sub-módulo controlador, que se conecta al módulo de planificación mediante una conexión de tipo TCP/IP, ya que es la forma más sencilla de enviar objetos entre programas distintos teniendo en cuenta la experiencia del programador, tomando ventaja sobre otras alternativas contempladas como RMI. Por un lado, encapsula la información de los sensores que llega de *Sensors* (Sensores_C) y la manda a dicho módulo de planificación cuando sea necesario. Por el otro, recibe del mismo módulo de planificación la acción que se debe realizar (Acción_B). La acción es interpretada y, en caso de ser un movimiento, lo publica (Movimiento_A) en un *topic* al que *Motion* está suscrito, cambiando así el movimiento que está publicando continuamente dicho sub-módulo.

El funcionamiento del módulo de control es el siguiente:

1. Al iniciarse, los sub-módulos *Motion* y *Sensors* comienzan el funcionamiento que tendrán durante toda la ejecución. Por otro lado, *Controller* queda a la espera como servidor a que el módulo de planificación se conecte a él mediante TPC/IP.
2. Una vez la conexión ha sido realizada, lo primero que hace *Controller* es comprobar, mediante la información de la que dispone de los sensores, si el motor del robot está encendido o apagado. En caso de que sea necesario, lo enciende, y envía por primera vez los sensores del robot al módulo de planificación para que éste pueda obtener el

primer estado de alto nivel (esperando mediante un *sleep* a que el módulo de planificación esté preparado para recibirlos).

3. Tras esto, *Controller* entra en un bucle, en principio infinito, que sólo terminará cuando se reciba el mensaje del módulo de planificación de que la ejecución completa ha terminado. En dicho bucle sólo se actuará cuando se reciba algún mensaje del módulo de planificación. Éste mensaje puede ser un movimiento, un "quedarse quieto", o un, como se acaba de mencionar, "ejecución terminada".
4. En caso de ser un "quedarse quieto", *Controller* únicamente manda los sensores del robot al módulo de planificación.

   En caso de ser un movimiento, *Controller* lo transforma en un objeto que puede publicar en el *topic* al que *Motion* está suscrito y lo publica.

   Para determinar que el movimiento ha sido realizado, en caso de ser un avance, *Controller* determina la posición absoluta en la que debería pararse el robot independientemente del error cometido en el desplazamiento a la posición donde se encuentra. Por ejemplo, en caso de encontrarse el robot en la posición x=0,48, si se le manda avanzar 0,5 en el sentido positivo de las x se determina que donde debería pararse es en x=1,0 y no en x=0,98. De igual modo, si en lugar de un avance hay que realizar un giro, al existir dos valores de odometría que indican el ángulo girado, se puede saber cuándo esos valores indican que se ha girado en un ángulo recto (90º, que es el ángulo que se gira siempre, en un sentido o en otro), y es entonces cuando se determina que el movimiento ha terminado.

   Entonces cambia el movimiento para que sea "parar" y lo publica hasta que el robot lo hace. Es entonces cuando envía los sensores del robot al módulo de planificación.
5. Cuando se recibe el mensaje de "ejecución terminada", se sale del bucle, se envían los sensores al módulo de planificación una última vez y se cierra la conexión con él, terminando así la ejecución.

En la Figura 21 puede verse el diagrama de flujo de *Controller*, ya que los de *Motion* y *Sensors* es trivial y es *Controller* quien realmente maneja el módulo de control.

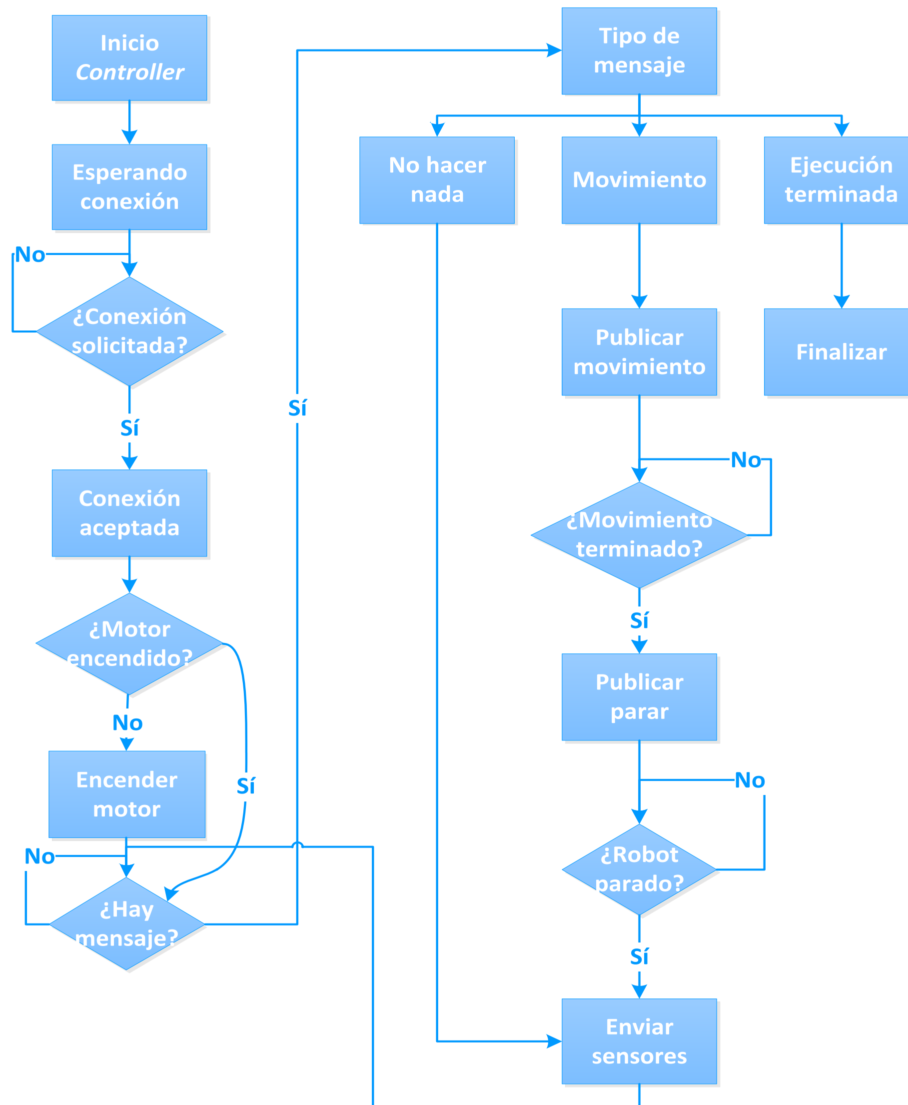


Figura 21 - Diagrama de flujo de *Controller*

Como existe un pequeño retardo entre que se indica al robot que pare tras un movimiento realizado y el momento en que este finalmente lo hace, no se le indica parar cuando ha realizado el movimiento completo, sino un poco antes, ya que es así como realizará el movimiento justo, tanto en avances como en giros.

Cabe destacar la siguiente limitación del módulo de control: a pesar de que *Sensors* obtiene 4 datos distintos, finalmente sólo se utilizan dos, que son la odometría y el estado del motor. De

hecho, el estado del motor se usa solamente para encender o no el motor al principio de la ejecución, el módulo de planificación sólo utiliza la odometría. Esto ha sido así porque, aunque en un principio se contemplaban las opciones de tener en cuenta las lecturas de los sónares o el nivel de batería en la resolución de los problemas, finalmente no se ha utilizado estos datos para la resolución de problemas a alto nivel.

3.2.2.1.1 Mensajes personalizados

Una de las características principales de ROS es el hecho de poder definir mensajes personalizados para publicar en los *topics*, que resulta muy útil cuando se quiere encapsular en un mismo objeto un conjunto de datos que no se adecuan a ninguno de los mensajes predefinidos de ROS.

En el caso de este trabajo se ha definido un tipo nuevo de mensaje: *SensorsData* (se muestra en el Anexo D.1.2). En un primer lugar se pensó en definir un mensaje personalizado para transmitir el movimiento del robot entre el controlador y el sub-módulo de movimiento, pero finalmente se utilizó directamente el mismo mensaje que comunica los movimientos directamente al robot (mensaje *geometry_msgs/Twist*, su contenido se muestra en el Anexo D.1.1) ya que, para la manera en que se ha implementado, no era necesario añadir más información a dicho mensaje.

Por su parte, el mensaje *SensorsData* combina en un mismo mensaje los mensajes que vienen dados por los cuatro sensores que se han tenido en cuenta, de forma que se pueden publicar en un *topic* como un mismo objeto para así comunicar la información de los sensores del sub-módulo de sensores al controlador.

Cabe comentar que la utilización de mensajes personalizados en rosjava, a pesar de que parecía conflictiva en un principio, simplemente consiste en crear un tipo de proyecto especial de rosjava para los mensajes que, por decirlo de algún modo, “importa” el proyecto creado por el método tradicional de creación de mensajes personalizados en ROS.

3.2.2.2 Módulo de planificación

El módulo de planificación utiliza la arquitectura PELEA. Dicha arquitectura ha sido explicada de forma general en anteriores capítulos del presente documento, pero las peculiaridades de su uso en este trabajo se describen en este apartado. La arquitectura utilizada consta de cinco nodos: *Monitoring*, *Decision Support*, *Execution*, *Low to High* y *Low Lever Planner*. Se puede ver esta arquitectura en la Figura 22.

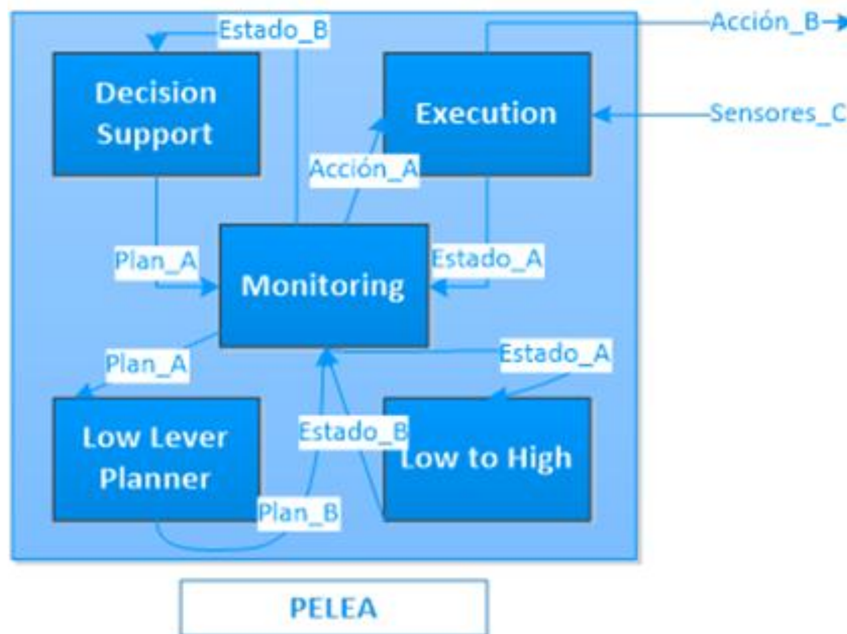


Figura 22 - Diagrama del módulo de planificación

A continuación se hace una descripción general de cada uno de los cinco nodos que componen el módulo de planificación:

- *Decision Support*: en este nodo se encuentra el planificador. Se recibe el estado de alto nivel (Estado_B en el diagrama) inicial del *Monitoring*, y a partir de ello se genera un plan de acciones de alto nivel (Plan_A) que se genera casi instantáneamente, en unos milisegundos, y se envía de vuelta a *Monitoring*. Este nodo no debería volver a participar a no ser que hiciera falta replanificar, cosa que en el presente sistema, al no haber tenido en cuenta las lecturas de los sónares o el nivel de batería, no debería hacer falta nunca.

- *Execution*: este nodo es el que se conecta por TCP/IP al módulo de control, le indica las acciones de bajo nivel que tiene que realizar (Acción_B) y obtiene de él el objeto con los datos de los sensores (Sensores_C). Debe haber uno de estos nodos por cada robot en el sistema, por tanto en el caso de este trabajo deberá haber dos, cada uno con un módulo de control para su robot, por supuesto. Ambos *Execution* podrían estar en el mismo PC, pero al tener que conectar los robots al PC por cable se ha tenido que elegir tener uno en un segundo PC. La conexión entre los nodos de PELEA se hace mediante RMI incluso en un mismo PC en local, por lo que simplemente el *Execution* que se ejecute en un PC remoto debe apuntar a la IP del PC principal en el fichero de configuración de PELEA. De hecho, podría utilizarse un PC externo para ejecutar todos los nodos salvo los *Execution* y tener dos PCs que únicamente tuvieran un *Execution* cada uno, pero por simplicidad el papel del PC externo lo ha tomado uno de los dos. La acción a realizar (Acción_A) le llega de *Monitoring*, y *Execution* simplemente le da un formato adecuado para enviarla al módulo de control, o envía un mensaje de "no hacer nada" si la acción que corresponde en ese momento no pertenece al robot de este *Execution*. Los sensores se envían al *Monitoring* tras expresarlos en formato XML (Estado_A).
- *Low to High*: es el nodo encargado de traducir a alto nivel la información sobre los estados de bajo nivel de cada uno de los robots que le llega del módulo de control del robot a través de *Monitoring* tras la ejecución de cada acción de bajo nivel. Para realizar esta traducción utiliza los datos de los sensores recibidos tras la realización de cada una de las acciones de bajo nivel que componen la acción de alto nivel actual. Esta información se combina con algunos datos que provienen del problema en PDDL, en concreto los predicados estáticos[1] a lo largo de todo el problema, las declaraciones de los objetos y las metas (Estado_A). El módulo contiene un objeto para la representación del mapa, incluyendo listas con objetos que representan las características de los robots y las posiciones de todos los objetos en el problema y que se puede expresar en forma

---

[1] En Planificación Automática un predicado estático es aquél que no es añadido ni borrado por ninguna acción y que por lo tanto se considera una característica inmutable del problema. En el caso del dominio del *Sokoban*, ejemplos de predicados estáticos son los que definen si una casilla (*location*) es meta o no, o el que describe la dirección en la que hay que moverse para pasar de una casilla a otra adyacente.

de estado de alto nivel (Estado_B) cuando se requiera. Este objeto se actualiza cada vez que se realiza una nueva acción de alto nivel y se obtiene el estado de bajo nivel a través de los sensores de un robot. Para realizar esta actualización, se adjudican unas coordenadas a cada casilla del mapa (la casilla 1,1 tiene las coordenadas x=0 e y=0) y se guarda la posición relativa inicial del robot a partir de la casilla en la que se encuentra según el problema PDDL. De esta forma se pueden calcular futuras posiciones del robot con los datos de sus sensores (aunque se transmiten los cuatro anteriormente indicados sólo se tiene en cuenta la odometría para realizar este cálculo). La posición de las cajas se actualiza cuando el robot pasa a ocupar una casilla previamente ocupada por una caja, asumiendo que la caja ha pasado a la casilla adyacente, empujada por el robot

- *Low level planner*: es el nodo que se ocupa de traducir el plan de acciones de alto nivel (Plan_A) a acciones de bajo nivel (Plan_B). Para ello, simplemente se comprueba la orientación del robot y cada movimiento se descompone en los giros de 90 grados necesarios, y en un avance de una cantidad de metros que viene dada por parámetro en el fichero de configuración de PELEA. Las acciones que involucran simultáneamente a dos robots igualmente se descomponen en los giros que necesite cada robot según su orientación y en un avance para cada uno. En un primer momento, la labor de este nodo se realizaba en el módulo de control, pero se descartó esa opción dada la posibilidad que ofrece la arquitectura utilizada, PELEA, para realizar esta tarea.
- *Monitoring*: es el nodo principal. En él se registran los demás nodos al empezar la ejecución, y casi todo lo que ocurre pasa por él. Aparte de lo mencionado en la descripción de lo demás nodos, es el encargado de seleccionar la acción del plan que se debe ejecutar en cada momento y enviarla a *Execution*. Además, compara el estado de alto nivel que le da *Low to High* con el estado esperado y, si no coinciden, debe mandarle a *Decision Support* el nuevo problema para que le envíe un nuevo plan. Como se comentó anteriormente, en este sistema no debería ocurrir.

A diferencia del módulo de control (que se ha desarrollado completo para este trabajo), en el de planificación sólo se ha trabajado sobre algunos nodos, dejando todo lo demás como viene por defecto con PELEA. Dichos nodos han sido el *Low Level Planner* y el *Low to High* en sus métodos de traducción, incluyendo los objetos desarrollados para el mapa, los robots y las

posiciones; y también el *Execution*, en el que se han debido modificar los métodos de ejecutar acción y obtener sensores para adecuarlos al problema del presente trabajo.

Además del trabajo sobre estos nodos, se ha debido modificar el dominio en PDDL del *Sokoban* para permitir que se realicen acciones conjuntas, ya que en el dominio utilizado la opción de utilizar varios robots sí aparecía, pero sin las acciones conjuntas. Simplemente se han tenido que añadir variaciones de las acciones de mover del dominio original en las que las precondiciones incluyen que los dos robots estén juntos y tengan una caja grande delante, y las dos casillas tras esa caja grande estén libres; y en los efectos, de forma análoga a las acciones tradicionales, que se vacíen las casillas de partida, los robots pasen a las de la caja y la caja a las dos que estaban libres. Para facilitar esto, se ha incluido un nuevo tipo de objeto en el dominio para representar las piedras grandes o dobles (objeto *big*), y predicados de *at* y *at-goal* específicos para ello, llamados *big-at* y *big-at-goal*. Se podrían haber adaptado los originales, pero se hizo así por simplicidad (ambos dominios se muestran en el Anexo D.2).

El funcionamiento del módulo de planificación es el siguiente:

1. El nodo *Monitoring* se inicia y se registra, y espera a que los demás hagan lo mismo.
2. Todos los nodos se inician y se van registrando en *Monitoring*. Los nodos *Execution*, además, se conectan mediante TCP/IP con sus respectivos módulos de control.
3. *Monitoring* comprueba que tiene registrados los nodos que correspondan (en el caso de este sistema, dos *Execution* y uno de cada uno de los demás, cosa que se indica en el archivo de configuración) y, en caso afirmativo, les manda un mensaje de activación a todos.
4. *Monitoring* pide a los *Execution* el estado de bajo nivel y estos, tras obtenerlo del módulo de control, se lo envían en XML.
5. *Monitoring* envía el estado de bajo nivel de todos los *Executions* del sistema a *Low to High*, que genera el mapa inicial y obtiene el estado de alto nivel inicial, mediante el mapa generado y el problema en PDDL. El estado de alto nivel es reenviado a *Monitoring*.

6. El estado de alto nivel es enviado por *Monitoring* a *Decision Support*, que utiliza el planificador (Metric-FF en el caso de este trabajo) para obtener un plan de acciones de alto nivel que resuelvan el problema. Dicho plan es reenviado a *Monitoring*.
7. El plan recién generado es enviado a *Low Level Planner*, que lo convierte en un plan de acciones de bajo nivel y lo devuelve a *Monitoring*.
8. *Monitoring* envía la primera acción del plan a ambos *Execution*. Estos preparan la acción y la envían al módulo de control (en caso de no ser una acción para el robot de ese *Execution*, cosa que saben comparando el nombre del "actor" de la acción con el nombre del *Execution*, la acción que envían es "no eres tú, no hagas nada"), que le devuelve los datos de sensores cuando termina (o inmediatamente si no tienen que hacer nada). *Execution* le da esta información en XML, como ya hizo antes, a *Monitoring*.
9. El nodo *Monitoring* le envía la siguiente acción del plan a los *Execution* si quedan más acciones de bajo nivel pertenecientes a la primera acción de alto nivel o, si era la última acción de bajo nivel de la acción de alto nivel, manda los estados de bajo nivel de la actual acción de alto nivel a *Low to High* para que actualice el mapa y devuelva el estado de alto nivel.
10. *Monitoring* compara este estado de alto nivel con el esperado. Si es el mismo (debería ser siempre) continúa con la siguiente acción, si no lo es envía el estado de alto nivel de nuevo a *Decision Support* para que replanifique y devuelva un nuevo plan de alto nivel.
11. La ejecución continúa hasta que todas las acciones del plan se han realizado, entonces se manda una última acción que se incluyó en *Low Level Planner* al traducir el plan, y que le indica al módulo de control que la ejecución ha terminado y puede cerrar la conexión.

En la Figura 23 se presenta el diagrama de flujo del nodo *Monitoring*, donde se ve la secuencia que se realiza con la ejecución del sistema en dicho nodo y, por tanto, en todo el módulo de planificación, ya que el *Monitoring* controla en buena medida a los demás.

Cabe destacar que, al estar el segundo *Execution* en un PC remoto, en el segundo robot al retardo comentado en el apartado que describe el módulo de control se añade el posible retardo que se produzca por esta conexión. Para solventarlo se ha añadido en el módulo de control del

primer robot un *sleep* de un cuarto de segundo que soluciona en gran medida este problema. Además, el adelanto también comentado anteriormente con el que se avisa en el módulo de control de que el movimiento ha finalizado se ha agrandado ligeramente en el segundo PC.

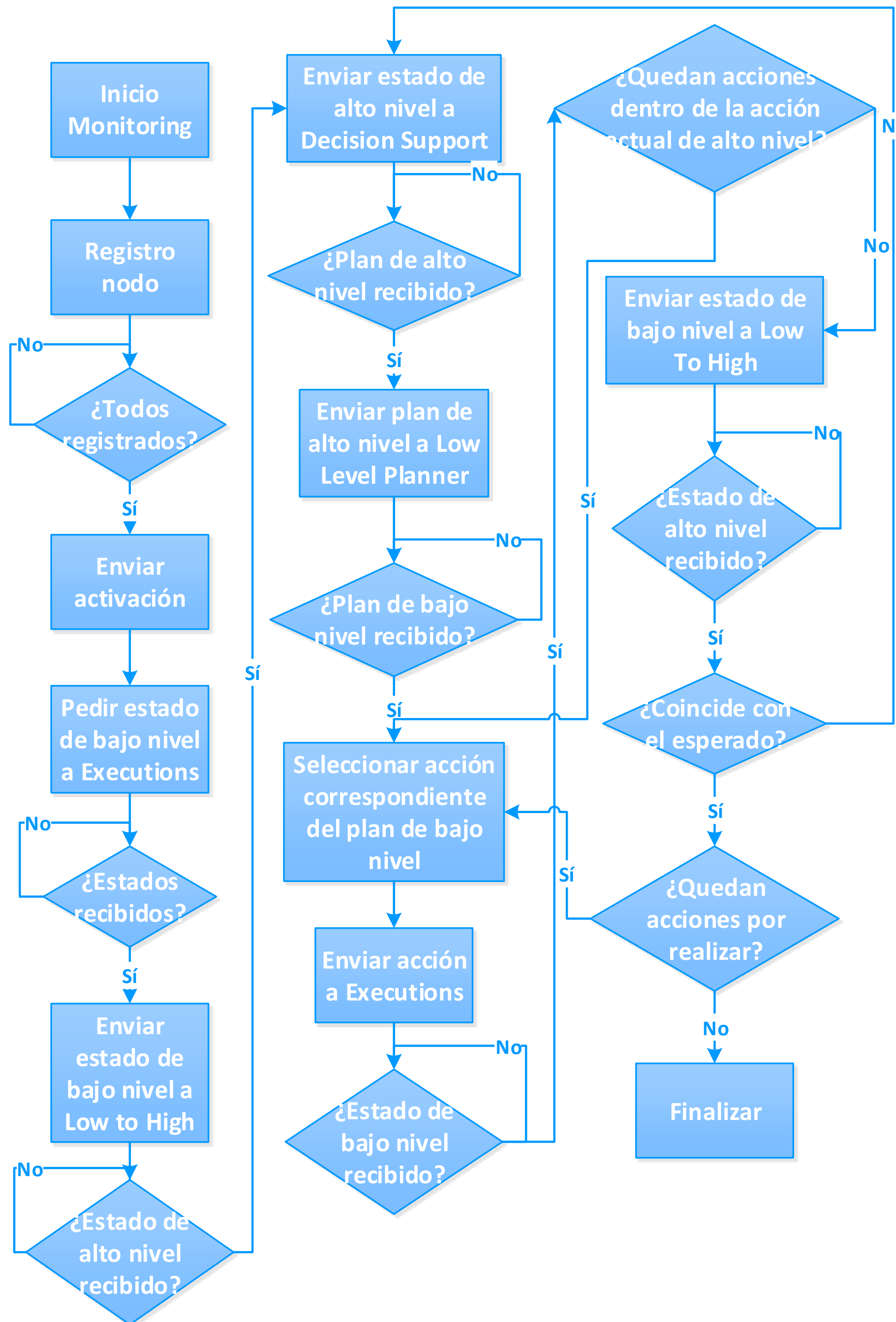


Figura 23 - Diagrama de flujo de *Monitoring*

Por lo demás, ambos módulos deben ser exactamente iguales en los dos PCs. La única diferencia en el módulo de planificación radica en que el *Execution* de cada PC debe recibir en el fichero de configuración el nombre de cada robot del problema en PDDL.

De nuevo como en el apartado del módulo de control, cabe comentar que no se han utilizado finalmente las lecturas de los sónares ni el nivel de batería, y por ello no se realiza la replanificación que se podría realizar, por ejemplo, al encontrar un obstáculo no indicado en el problema en PDDL. Además, no se puede saber si realmente se ha empujado una caja, únicamente se puede suponer que siempre se hace bien. Los experimentos realizados en el entorno real muestran que para problemas como los utilizados en este trabajo, que incluyen empujar una misma caja desde direcciones distintas y por distintos robots, esta asunción es perfectamente válida y los errores en las posiciones finales de las cajas son aceptables (ver sección de Experimentación).

3.2.3 Funcionamiento del sistema

El funcionamiento de los dos módulos por separado se ha presentado en los anteriores apartados. En éste, se combinan y se representan gráficamente mediante la Figura 24, que muestra el funcionamiento del sistema completo mediante un diagrama de secuencia.

En dicho diagrama se muestra la secuencia completa de ejecución del sistema, con un par de salvedades. La primera es que los “Iniciar ejecución” del comienzo deberían ser a todos los nodos de PELEA y todos los sub-módulos de ROS, pero por claridad en el diagrama sólo se han mostrado dos (además, de la forma que se ha implementado, con scripts para cada uno de los dos módulos, el usuario sólo debe realizar dos “inicios”). La segunda es que la parte del registro de los nodos de PELEA y la posterior activación del *Monitoring* se ha obviado, de nuevo por claridad y porque no aportan suficiente conocimiento útil para la descripción de este sistema en concreto.

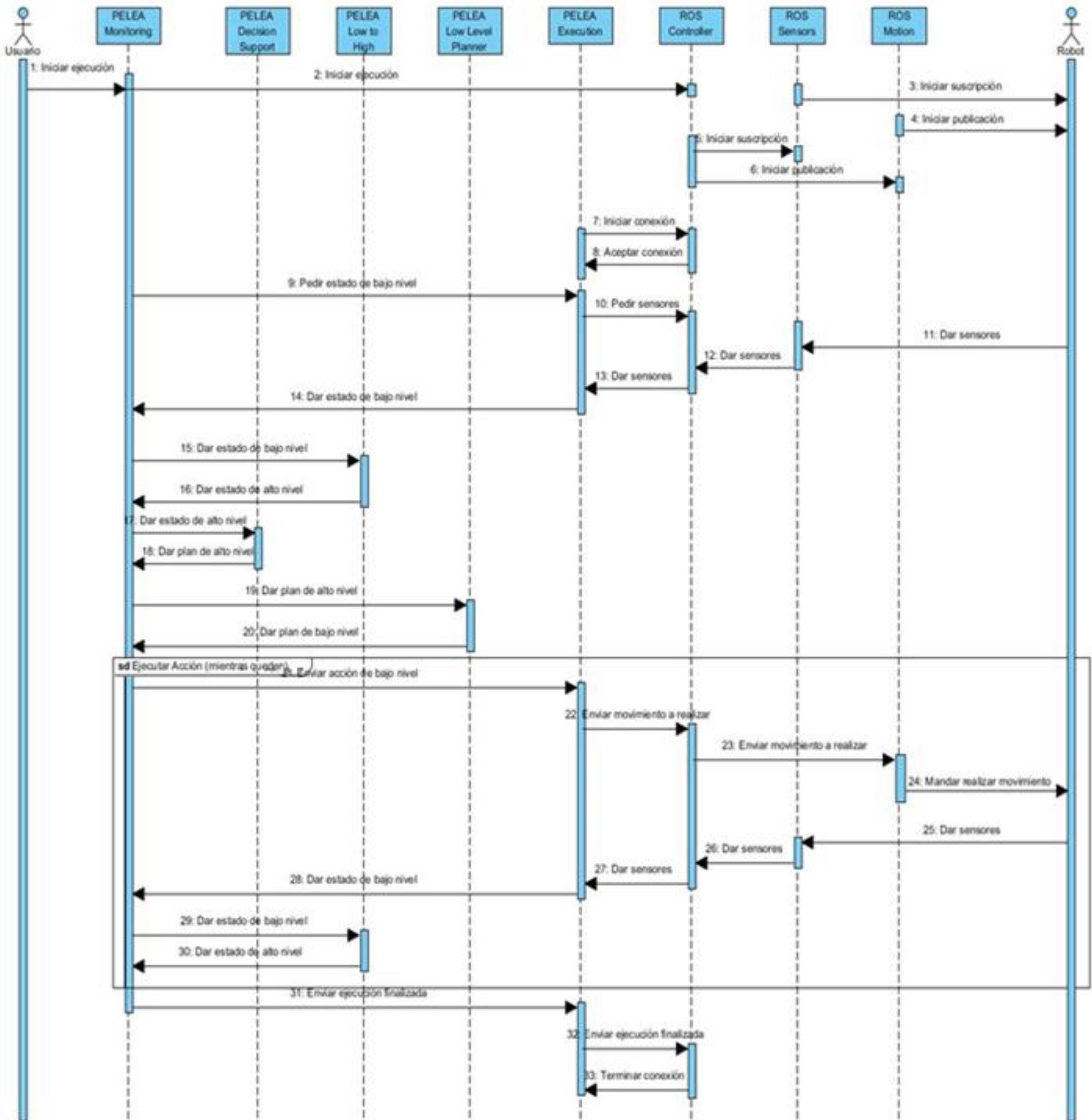


Figura 24 - Diagrama de secuencia del sistema

Capítulo 4: Experimentación

En este capítulo se muestra la experimentación realizada para comprobar el correcto funcionamiento del sistema. Esta experimentación consta de dos partes: las pruebas unitarias y las pruebas del sistema.

En primer lugar se muestran los elementos necesarios para construir el entorno en el que se realizarán las pruebas.

A continuación se muestran las pruebas unitarias, que demuestran que cada pequeña parte del sistema funciona correctamente independientemente de las demás (en la medida de lo posible).

Posteriormente, las pruebas del sistema demuestran que el sistema completo funciona de la forma esperada. Dichas pruebas han sido grabadas en vídeos que pueden verse más adelante en el capítulo, con una pequeña descripción de cada uno.

4.1 Elementos del entorno de pruebas

Los elementos, tanto hardware como software, utilizados para la realización de las pruebas son los siguientes:

- Tres PCs portátiles: En la mayor parte de las pruebas sólo se utilizará uno, pero son necesarios más para aquellas en las que se comprueba la posibilidad de tener más agentes funcionando.
- Dos robots P3DX: En estas pruebas, normalmente se utilizará el simulador, pero los robots son necesarios para ver si el sistema es capaz de utilizarse en robots físicos. De igual manera no será hasta las últimas pruebas donde se utilizarán los dos robots.
- ROS Hydro: Framework necesario para ejercer de módulo de control: conectarse y controlar a los robots (simulados o físicos).
- Pelea: Arquitectura necesaria para ejercer de módulo de planificación, y llevar a cabo la planificación y ejecución de las acciones.
- rosjava: Extensión de ROS para poder ser utilizado en lenguaje Java.

- P2OS: Conjunto de paquetes que ejercen de driver de los robots. Incluye más funcionalidades entre las que se destaca el simulador utilizado en estas pruebas.
  - Simulador: Interfaz gráfica que muestra un robot dentro de un escenario sobre el que se simula la conexión que se lleva a cabo sobre los robots físicos (Figura 25). Por ello, el robot simulado recibe y proporciona, de la misma forma que el robot físico, las funcionalidades más básicas de los P3DX (por ejemplo, no se simulan los sónares). Las que se han utilizado en estas pruebas son la obtención de la odometría y el control de la velocidad (movimientos) del robot.

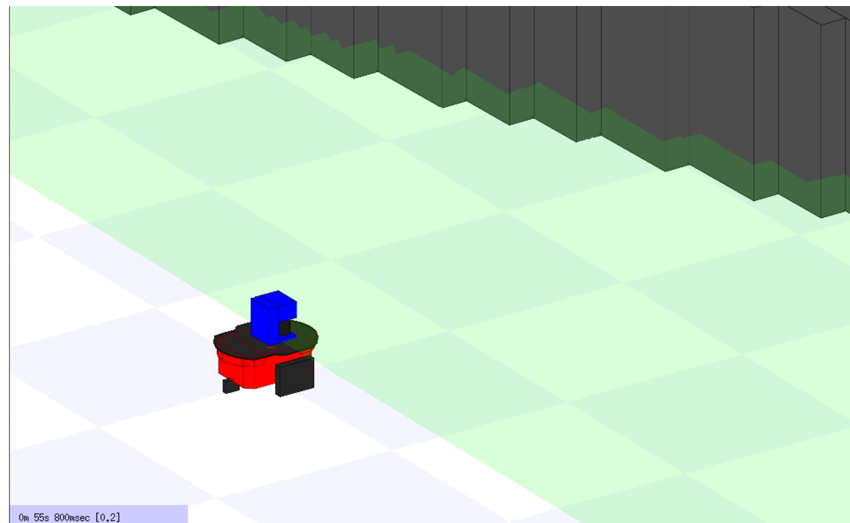


Figura 25 - Simulador P2OS

4.2 Pruebas unitarias

En este apartado se muestran las pruebas unitarias. Dichas pruebas han consistido en la ejecución parcial del sistema, en ocasiones recurriendo a un simulador en lugar de al robot real, para comprobar que cada funcionalidad funciona separadamente a las demás.

4.2.1 Descripción de los atributos de las pruebas unitarias

- Código: Identificación unívoca abreviada de la prueba. Se construye mediante la concatenación de las letras “PU” seguidas de un guion y tres dígitos. Por ejemplo PS-001.
- Requisitos verificados: Requisitos que verifica la prueba.

- Nombre: Identificación extendida de la prueba.
- Descripción: Se realiza una descripción básica de la prueba que ha sido realizada.
- Procedimiento: Secuencia de pasos que se deben realizar para la prueba.
- Resultado esperado: Condiciones que deben cumplirse en el sistema para considerar la prueba como verificada.

4.2.2 Descripción textual de las pruebas unitarias

En este apartado se presentan, mediante las siguientes tablas, las pruebas unitarias realizadas.

Código	PU-001	**Requisitos verificados**	RF-004
Nombre	Obtención de información de los sensores.		
Descripción	Debe ser posible acceder a la información de los sensores del robot una vez esté en marcha el módulo destinado a ello.		
Procedimiento	1. Se lanza el simulador del robot o se conecta el robot físico y se lanza el fichero que lo activa. 2. Se lanza el módulo *Sensors* de ROS (en el caso del simulador, se debe modificar el módulo para que sólo muestre la odometría, así como el canal del que se toma ya que no se llama igual que el del robot). 3. Se ejecuta en una terminal el comando que muestra por pantalla lo que se está publicando en el *topic '/sensors'*.		
Resultado esperado	• La terminal muestra por pantalla los datos de los sensores (sólo odometría para el simulador) correctamente.		

Tabla 36 - Prueba unitaria PU-001

Código	PU-002	**Requisitos verificados**	RF-012, RF-013
Nombre	Movimientos del robot.		
Descripción	Debe ser posible mover (avanzar o girar) el robot una distancia concreta en un tiempo determinado.		
Procedimiento	1. Se lanza el simulador del robot o se conecta el robot físico y se lanza el fichero que lo activa. 2. Se lanza el módulo *Motion* de ROS. 3. Se ejecuta en una terminal el comando que publica en el *topic '/move'* un mensaje del tipo que se ha definido para describir los movimientos del robot, con los datos del movimiento que se quiera realizar.		
Resultado esperado	• El robot (simulado o físico) se mueve la distancia o el ángulo indicados, en el tiempo requerido.		

Tabla 37 - Prueba unitaria PU-002

Código	PU-003	**Requisitos verificados**	RF-005, RF-006, RF-007, RF-001
Nombre	Tratamiento de información de los sensores.		
Descripción	El módulo de planificación debe recibir los sensores del módulo de control y tratarlos de la forma requerida para su utilización.		
Procedimiento	1. Se lanza el simulador del robot o se conecta el robot físico y se lanza el fichero que lo activa. 2. Se lanza el script con los tres módulos de ROS. 3. Se lanza el script con los cinco módulos de PELEA.		
Resultado esperado	• Tras el registro de todos los nodos de PELEA, la terminal del nodo *Monitoring* de PELEA muestra por pantalla los datos de los sensores en XML. • Pocos segundos después, la terminal del nodo *Monitoring* de PELEA muestra por pantalla el estado de alto nivel obtenido a partir de los datos de los sensores en XML.		

Tabla 38 - Prueba unitaria PU-003

Código	PU-004	**Requisitos verificados**	RF-008, RF-001
Nombre	Traducción de acciones a bajo nivel.		
Descripción	El módulo de planificación debe traducir las acciones de alto nivel a un bajo nivel directamente entendible por el robot.		
Procedimiento	1. Se lanza el simulador del robot o se conecta el robot físico y se lanza el fichero que lo activa. 2. Se lanza el script con los tres módulos de ROS. 3. Se lanza el script con los cinco módulos de PELEA.		
Resultado esperado	• Tras la obtención del primer estado de alto nivel, la terminal del nodo *Monitoring* de PELEA muestra por pantalla el plan con las acciones de alto nivel. • Pocos segundos más tarde, la terminal del nodo *Monitoring* de PELEA muestra por pantalla el plan con las acciones de bajo nivel.		

Tabla 39 - Prueba unitaria PU-004

Código	PU-005	**Requisitos verificados**	RF-009, RF-010, RF-011, RF-001
Nombre	Transferencia de la acción.		
Descripción	El módulo de planificación debe enviar la acción, y el de control debe recibirla e interpretarla.		
Procedimiento	1. Se lanza el simulador del robot o se conecta el robot físico y se lanza el fichero que lo activa. 2. Se lanza el script con los tres módulos de ROS. 3. Se lanza el script con los cinco módulos de PELEA.		
Resultado esperado	• Tras la generación del plan de acciones de bajo nivel, el robot simulado o físico realiza la primera acción de dicho plan.		

Tabla 40 - Prueba unitaria PU-005

Código	PU-006	**Requisitos verificados**	RF-014, RF-001
Nombre	Segundo robot en PC remoto.		
Descripción	El sistema debe ser capaz de controlar un segundo robot desde un PC remoto.		
Procedimiento	1. Para dos PCs, se lanza el simulador del robot o se conecta el robot físico y se lanza el fichero que lo activa. 2. Se lanza el script con los tres módulos de ROS para los dos PCs. 3. Se lanza el script con los cinco módulos de PELEA en el PC principal. 4. Se lanza el script con el segundo *execution* en el PC secundario.		
Resultado esperado	• Tras la generación del plan de acciones de bajo nivel, cada robot realiza las acciones que le corresponden, y se queda inmóvil en las que no.		

Tabla 41 - Prueba unitaria PU-006

Código	PU-007	**Requisitos verificados**	RF-015, RF-001
Nombre	Movimientos simultáneos.		
Descripción	Ambos robots deben realizar los movimientos simultáneos que les correspondan.		
Procedimiento	1Para dos PCs, se lanza el simulador del robot o se conecta el robot físico y se lanza el fichero que lo activa. 1. Se lanza el script con los tres módulos de ROS para los dos PCs. 2. Se lanza el script con los cinco módulos de PELEA en el PC principal. 3. Se lanza el script con el segundo *execution* en el PC secundario.		
Resultado esperado	• Tras la generación del plan de acciones de bajo nivel y, cuando llegue una acción simultánea (empujar una caja doble), ambos robots realizan el movimiento correspondiente simultáneamente.		

Tabla 42 - Prueba unitaria PU-007

Código	PU-008	**Requisitos verificados**	RF-001, RF-002, RF-003, RF-016
Nombre	Fin de la ejecución.		
Descripción	Al terminar la ejecución, sin ninguna interacción por parte del usuario, debe cerrarse la conexión entre los módulos de planificación y control.		
Procedimiento	1. Se lanza el simulador del robot o se conecta el robot físico y se lanza el fichero que lo activa. 2. Se lanza el script con los tres módulos de ROS. 3. Se lanza el script con los cinco módulos de PELEA.		
Resultado esperado	• Tras la realización de todas las acciones por el robot, aparece en la terminal del *Controller* del robot que el socket ha sido cerrado.		

Tabla 43 - Prueba unitaria PU-008

Código	PU-009	**Requisitos verificados**	RF-017, RF-001
Nombre	Uso de más de dos robots.		
Descripción	El sistema debe ser capaz de resolver problemas para más de dos robots.		
Procedimiento	1. Se utiliza un problema en PDDL que incluya más de dos robots (por ejemplo tres). 2. Se lanza el simulador del robot en tres PCs (esta prueba no ha podido realizarse en robots físicos por falta de robots). 3. Se lanza el script con los tres módulos de ROS en los tres PCs. 4. Se lanza el script con los cinco módulos de PELEA en el PC principal. 5. Se lanza el script con el segundo *execution* en el PC secundario. 6. Se lanza el script con el tercer *execution* en el tercer PC.		
Resultado esperado	• Tras la generación del plan de acciones de bajo nivel, cada robot simulado realiza las acciones que le corresponden, y se queda inmóvil en las que no.		

Tabla 44 - Prueba unitaria PU-009

4.3 Pruebas del sistema

En este apartado se muestran las pruebas del sistema. Dichas pruebas consisten en la resolución de una serie de problemas de dificultad incremental en un entorno real. Para su realización, ha sido necesaria una preparación previa del entorno, tras la cual se ha procedido a la ejecución del sistema y su grabación.

4.3.1 Preparación de las pruebas del sistema

Para la realización de las pruebas del sistema ha sido necesario preparar un entorno en el que solucionar los problemas. Para ello, se ha utilizado un pasillo de la Universidad suficientemente espacioso como para montar un escenario equivalente a los problemas del *Sokoban* a resolver.

El escenario ha sido montado utilizando el tamaño de las baldosas como el tamaño de una casilla del dominio del juego. Convenientemente, dichas baldosas son de tamaño 50x50 cm, que era el tamaño que se quería considerar para cada casilla desde un principio debido a las dimensiones del robot.

Para la delimitación de dicho escenario se ha utilizado cinta aislante opaca que muestra el borde del tablero de cada problema del *Sokoban*. También se ha utilizado una cinta negra para señalar los sitios por los que los robots no pueden pasar; sería el equivalente a una pared u obstáculo no movible del dominio del juego. Las metas se han marcado con cruces de cinta.

Como cajas, se han utilizado dos tipos distintos. Las cajas normales (llamadas *stone* en el dominio), son cajas del tamaño de una casilla, y las cajas reales utilizadas han sido prácticamente de dicho tamaño. Como cajas dobles (las llamadas *big* en el dominio) se han utilizado dos cajas unidas que ocupan prácticamente dos casillas completas.

Una vez se ha tenido esto para cada problema concreto, se ha podido proceder a la realización de cada experimento correspondiente.

4.3.2 Pruebas del sistema realizadas

A continuación se presentan las tres pruebas del sistema realizadas. Dichos pruebas se corresponden con problemas del dominio del *Sokoban* que han sido resueltos en un entorno real mediante la utilización de los robots P3DX y cajas en un tablero delimitado. En la Figura 26 se puede ver la leyenda utilizada para mostrar gráficamente los problemas correspondientes a dichos experimentos.

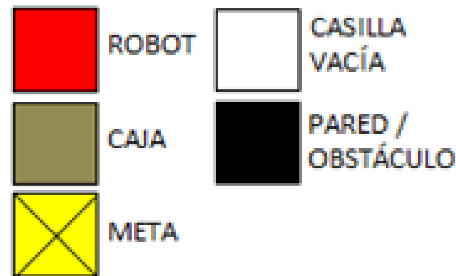


Figura 26 - Leyenda mapas *Sokoban*

4.3.2.1 Problema 1: Resolución de un problema con un robot

Para la comprobación de que el sistema es capaz de resolver un problema de *Sokoban*, lo primero que se hizo fue realizar una prueba para la resolución de un problema con un solo robot, lo que nos permitiría comprobar que la ejecución del sistema se realiza correctamente exceptuando el hecho de utilizar un único robot. El problema se muestra en la Figura 27.

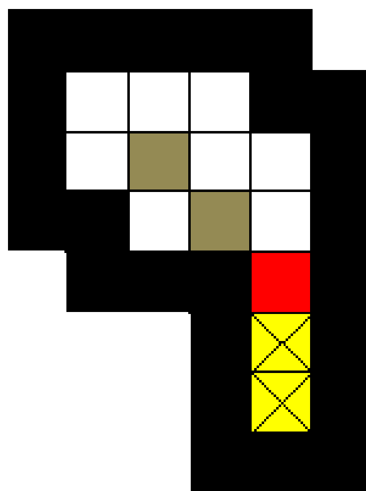


Figura 27 - Problema *Sokoban* 1

En este problema el robot debe ir a por las cajas y llevarlas a las casillas que se encuentran debajo de él en el estado inicial. Con su correcta realización, se demuestra que los módulos de control y planificación funcionan correctamente para un único robot. El único problema de este experimento es que, en ocasiones, las cajas de desviaban, pero esto se explica por el hecho de que las cajas, que deben ser del tamaño de una casilla, tienen una "superficie de empuje" mucho mayor que la superficie del robot que las empuja, por lo que es muy difícil que vayan totalmente rectas. Además, algunos de los giros del robot no son totalmente rectos y esto también afecta a este problema. Por otro lado, se puede ver que la mayoría de los movimientos del robot son totalmente correctos en comparación con los esperados. Tras dos realizaciones de esta prueba (en la primera, uno de los movimientos del robot fue demasiado erróneo y no se llegó a la resolución del problema) se obtuvo el siguiente vídeo, que se puede ver en este enlace:

https://www.youtube.com/watch?v=3-U20U18x8E

4.3.2.2 Problema 2: Colaboración de dos robots para un movimiento simultáneo

En este experimento se demuestra que el sistema es capaz de hacer que dos robots colaboren para mover una caja que no podrían mover individualmente. Con esto, se demostraría que las funcionalidades necesarias para que dos robots funcionen juntos se cumplen satisfactoriamente, lo que unido al anterior problema demuestra que el sistema completo funciona de forma correcta. El problema se muestra en la Figura 28.

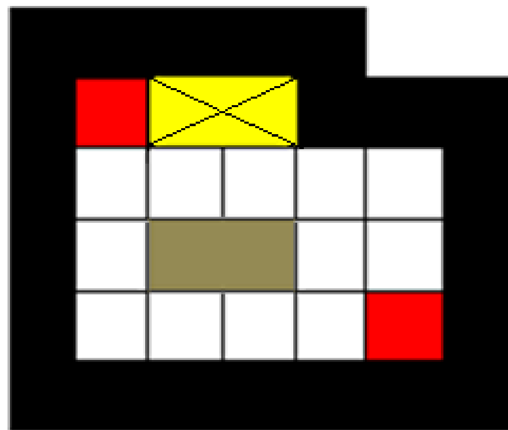


Figura 28 - Problema *Sokoban* 2

En este sencillo problema lo único que deben hacer los robots es mover la doble caja hacia la meta. A pesar de su simpleza, este problema tiene especial interés ya que demuestra que los robots son capaces de colaborar, lo que da sentido al hecho de que éste sea un sistema multi-agente. Como se ha comentado antes, tras su correcta realización, unida a la del anterior experimento, queda demostrado que el sistema cumple todas las funcionalidades pedidas. En este caso, podemos ver que la caja no se desvía prácticamente nada, debido a que, a diferencia del anterior experimento, ahora son dos robots los que empujan y la superficie que empuja la caja es mucho mayor. De nuevo, el movimiento de los dos robots es correcto, y su sincronización es bastante alta en los movimientos que realizan simultáneamente, a pesar de haber un pequeño retardo para uno de los robots en uno de ellos, que no impide que el sistema se ejecute correctamente. Esta sencilla prueba sólo tuvo que ser realizada una vez para obtener el siguiente vídeo. Para verlo, se debe visitar el enlace:

https://www.youtube.com/watch?v=AuoRH3ktHyM

4.3.2.3 Problema 3: Resolución de un problema para dos robots

Tras comprobar que el sistema funciona correctamente, sólo queda comprobar el alcance del sistema demostrando el tipo de problemas que se pueden solucionar con él. En este experimento, se puede ver como el sistema es capaz de resolver un problema especialmente diseñado para dos robots, donde ambos deben colaborar para solucionar el mapa, de una forma más compleja que en el apartado anterior. El problema se muestra en la Figura 29.

En este problema, el robot 1 debe mover la caja que está “encerrando” a su compañero para que éste se pueda mover. Después, colaborarán para mover la caja doble y poder acceder a la caja que queda, colocando cada una en su sitio. Con la correcta realización de este problema se puede ver el interés que puede tener un sistema multi-agente, ya que hay problemas que no pueden ser resueltos por un solo agente, mientras que dos lo hacen de una forma muy sencilla.

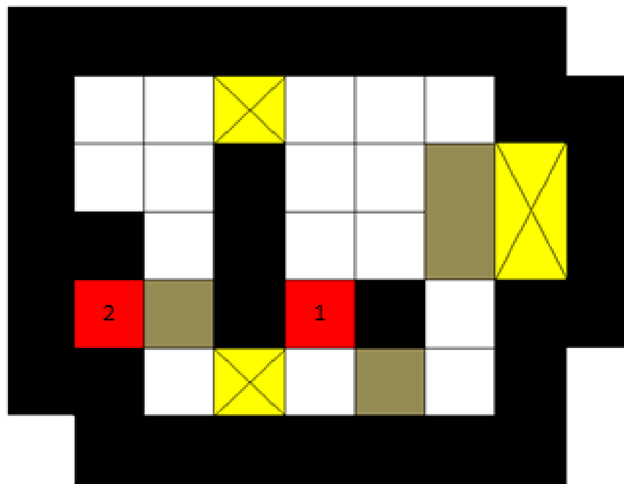


Figura 29 - Problema *Sokoban* 3

En este caso, el problema se resuelve de una forma muy satisfactoria, con todas las cajas acabando en el sitio requerido para ellas, y unos movimientos de los robots correctos. Las cajas se tuercen ligeramente en el último movimiento, pero en este caso los movimientos de los robots son correctos, por lo que puede deberse simplemente al motivo mencionado para el Problema 1 de la superficie de empuje, o a que las cajas no resbalen bien al pasar por las cintas del suelo. Por estos motivos, para obtener una correcta realización, esta prueba fue la más conflictiva. Debió realizarse cuatro veces porque en distintos momentos las cajas se desviaban, de similar manera a como ocurre en el último movimiento de la última prueba, a pesar de que los movimientos de los robots sean correctos. La manera de solucionar esto desde un punto de vista técnico no es sencilla. Incluso utilizando todos los datos de los sensores del robot (sónar y detector de colisiones) es muy complicado determinar que la caja se ha movido incorrectamente. El uso de sensores externos, como cámaras, ayudaría en esta determinación, de forma que se pudiera calcular el movimiento a bajo nivel que debe realizar cada robot para colocar la caja en su posición correcta. El vídeo obtenido puede verse visitando el enlace:

https://www.youtube.com/watch?v=X24vnPdXQxc

Capítulo 5: Gestión del trabajo

En este capítulo se describe el ciclo de vida del trabajo y las fases de las que consta, enumerando las tareas que conforman cada una. A continuación se describirá la planificación de dichas fases, y se mostrará gráficamente mediante un diagrama de Gantt. Por último, se enumerarán los medios necesarios para llevar a cabo el trabajo y se calculará un presupuesto de acuerdo a ello.

5.1 Descripción de las fases del trabajo

En la Figura 30 se puede ver el ciclo de vida elegido para el trabajo. Se ha elegido un ciclo de vida en cascada debido a que es un trabajo de una sola persona en el que la salida de una fase es la entrada de la siguiente y, por tanto, un modelo sencillo y secuencial es el más adecuado.

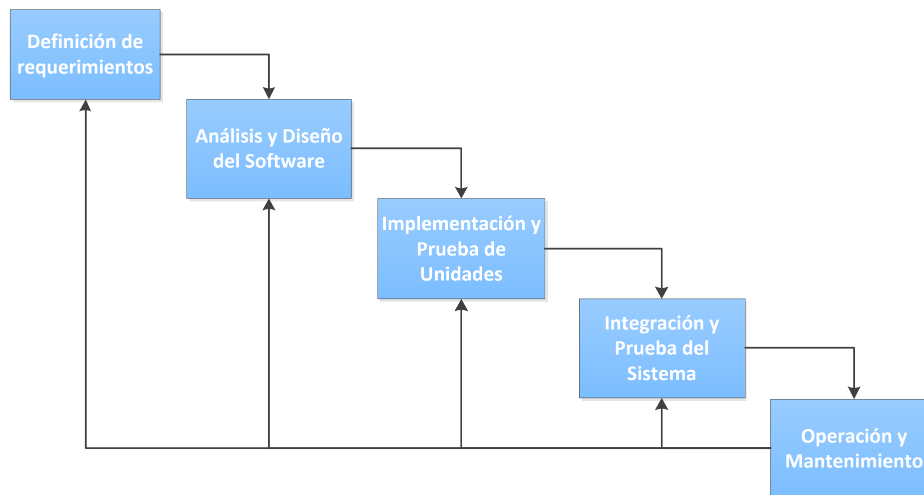


Figura 30 - Ciclo de vida en cascada

A continuación se describe cada una de las fases de forma detallada:

- La primera fase es la definición de los requisitos del trabajo. Esta fase consiste en definir las características que tiene que cumplir el trabajo a realizar. Para ello, se deben describir las funcionalidades del sistema, sus restricciones y los casos de uso que deban poder llevarse a cabo con él.

- La segunda fase se corresponde con el análisis y diseño de software. En esta fase, se debe analizar el software a utilizar para realizar la elección adecuada y, mediante el análisis de este software y los requisitos hallados en la fase anterior, diseñar los componentes software que habrá que implementar para construir el sistema.
- La tercera fase es la implementación de los componentes antes diseñados, y las pruebas unitarias de cada uno de ellos.
- La cuarta fase es la integración de los componentes antes implementados y las pruebas del sistema completo.
- La quinta y última fase es la operación y mantenimiento del sistema, es decir, utilizarlo y realizar las tareas necesarias para que su correcto funcionamiento continúe. Paralelamente a esto, en el caso de este trabajo, se realiza la documentación requerida.

5.2 Planificación

A continuación se muestra, en la tabla 45, la planificación en el tiempo de las tareas recién descritas y, en la Figura 31, el diagrama de Gantt correspondiente.

Nº	Tarea	Duración	Inicio	Fin
1	Definición de requisitos	24 días	22/12/2013	14/01/2014
2	Definición de funcionalidades	11 días	22/12/2013	01/01/2014
3	Definición de restricciones	6 días	02/01/2014	07/01/2014
4	Definición de casos de uso	7 días	08/01/2014	14/01/2014
5	Análisis y diseño de software	41 días	15/01/2014	24/02/2014
6	Análisis de Robocomp y ROS	14 días	15/01/2014	28/01/2014
7	Análisis de PELEA	9 días	29/01/2014	06/02/2014
8	Análisis del resto de software	4 días	07/02/2014	10/02/2014
9	Diseño del controlador del robot	7 días	11/02/2014	17/02/2014
10	Diseño de los módulos de PELEA	7 días	18/02/2014	24/02/2014
11	Implementación y pruebas unitarias	58 días	25/02/2014	23/04/2014
12	Implementación del controlador	29 días	25/02/2014	25/03/2014

Nº	Tarea	Duración	Inicio	Fin
13	Pruebas del controlador	8 días	26/03/2014	02/04/2014
14	Implementación de módulos de PELEA	17 días	03/04/2014	19/04/2014
15	Pruebas de módulos de PELEA	4 días	20/04/2014	23/04/2014
16	Integración y pruebas del sistema	22 días	24/04/2014	15/05/2014
17	Integración del sistema completo	8 días	24/04/2014	01/05/2014
18	Pruebas del sistema completo	14 días	02/05/2014	15/05/2014
19	Operación, mantenimiento y documentación	41 días	16/05/2014	25/06/2014
20	Uso del sistema y documentación	37 días	16/05/2014	21/06/2014
21	Preparación de la presentación	4 días	22/06/2014	25/06/2014

Tabla 45 - Planificación

5.3 Presupuesto

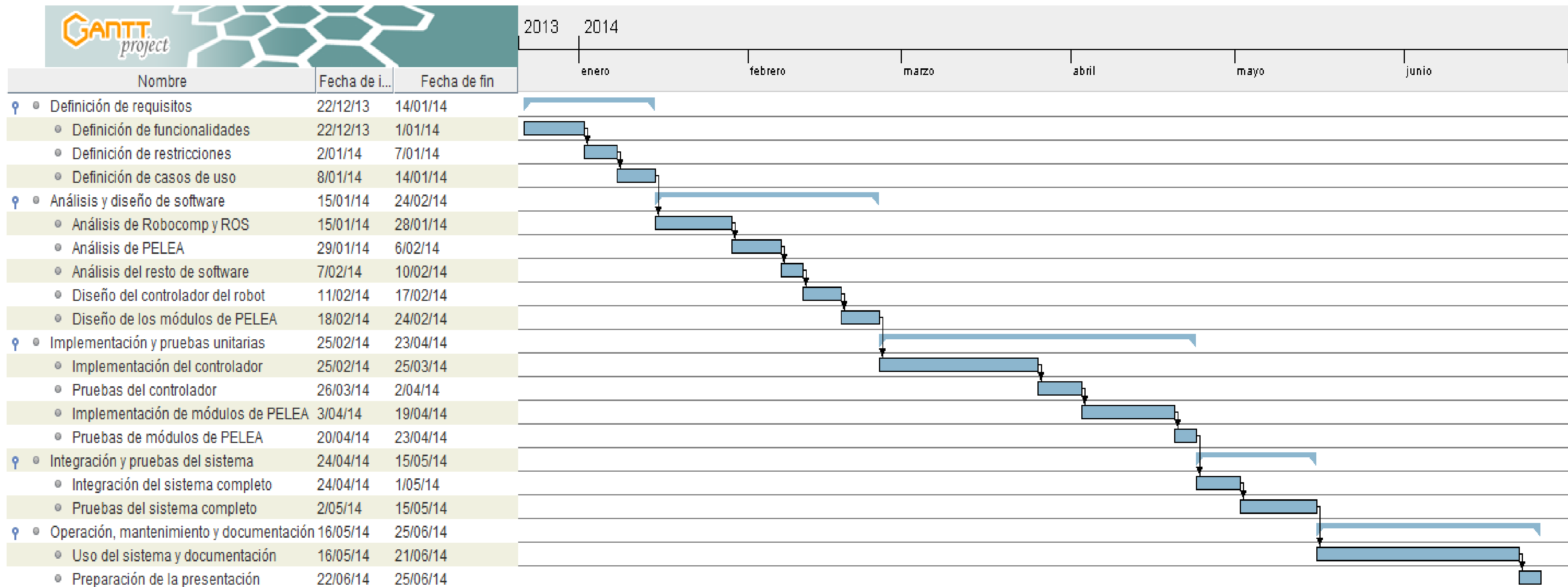
En este apartado se describe el presupuesto total del trabajo y su desglose: materiales utilizados, amortización, coste de personal y seguridad social.

Al salario base de cada empleado se le añade el coste de la seguridad social. El coste de seguridad social de un empleado equivale a un 35,5% de su salario bruto.

Para calcular la amortización de los materiales utilizados se multiplica el precio del material por el cociente de los meses que se ha utilizado entre su periodo de depreciación.

Autor	
Raúl García Jerez	
Departamento	
Informática	
Descripción del trabajo	
Título	Integración de Planificación Automática y ROS para el control autónomo de dos robots en el juego del *Sokoban*
Duración	6 meses

Tabla 46 - Datos del trabajo



Nombre	Fecha de i...	Fecha de fin
Definición de requisitos	22/12/13	14/01/14
Definición de funcionalidades	22/12/13	1/01/14
Definición de restricciones	2/01/14	7/01/14
Definición de casos de uso	8/01/14	14/01/14
Análisis y diseño de software	15/01/14	24/02/14
Análisis de Robocomp y ROS	15/01/14	28/01/14
Análisis de PELEA	29/01/14	6/02/14
Análisis del resto de software	7/02/14	10/02/14
Diseño del controlador del robot	11/02/14	17/02/14
Diseño de los módulos de PELEA	18/02/14	24/02/14
Implementación y pruebas unitarias	25/02/14	23/04/14
Implementación del controlador	25/02/14	25/03/14
Pruebas del controlador	26/03/14	2/04/14
Implementación de módulos de PELEA	3/04/14	19/04/14
Pruebas de módulos de PELEA	20/04/14	23/04/14
Integración y pruebas del sistema	24/04/14	15/05/14
Integración del sistema completo	24/04/14	1/05/14
Pruebas del sistema completo	2/05/14	15/05/14
Operación, mantenimiento y documentación	16/05/14	25/06/14
Uso del sistema y documentación	16/05/14	21/06/14
Preparación de la presentación	22/06/14	25/06/14

Figura 31 - Diagrama de Gantt

En la Tabla 46 se ven los datos del trabajo realizado, y en las tablas 47, 48 y 49 los costes de personal, materiales y totales, respectivamente.

Los salarios base se obtienen de las tablas salariales del BOE, nº 174 del 21 de julio de 2012.

Puesto	Salario base mensual (euros)	Nº de meses	Coste final (euros)	Coste final con SS* (euros)
Analista	1536,07	2,07	3179,66	4308,44
Programador	1193,24	1,77	2112,03	2861,80
Técnico de pruebas	1193,24	0,88	1050,05	1422,82
Jefe de proyecto	1884,17	1,3	2449,42	3318,96
Total Personal				**11912,02**

Tabla 47 - Coste de personal

*Coste final con SS = Coste final * 1,355

Material	Precio (euros)	Dedicación (meses)	Periodo de depreciación (meses)	Amortización* (euros)
HP Pavilion dv6	699,00	6	48	87,38
Portátil extra	299,00	1	48	6,23
Pioneer P3DX (x2)	5770,00	4	48	480,83
Ubuntu 12.04	0,00	6	0	0
NetBeans	0,00	4	0	0
JDK	0,00	4	0	0
ROS	0,00	3	0	0
PELEA	0,00	3	0	0
Total material				**574,44**

Tabla 48 - Coste de material

*Amortización = Precio * (Dedicación/Depreciación)

Personal	11912,02 €
Amortización	574,44 €
Subcontratación de tareas	0,00 €
Costes de funcionamiento	0,00 €
Presupuesto	12486,46 €
Costes indirectos (10%)	1248,65 €
Presupuesto ejecución material	13735,11 €
Beneficio industrial (6%)	824,11 €
Presupuesto total	**14559,22 €**

Tabla 49 - Resumen de costes

El presupuesto total de este trabajo asciende a la cantidad de CATORCE MIL QUINIENTOS CINCUENTA Y NUEVE EUROS CON VEINTIDÓS CÉNTIMOS (14559,22 €).

Leganés, a __ de junio de 2014

El ingeniero proyectista

Fdo. Raúl García Jerez

Capítulo 6: Conclusiones y trabajos futuros

En este apartado se describen las conclusiones obtenidas tras la realización del trabajo. Se comienza con las conclusiones generales del trabajo, seguidas por las conclusiones referentes a las objetivos marcados al inicio de éste, y terminando con la descripción de posibles trabajos futuros a realizar sobre el sistema desarrollado.

6.1 Conclusiones generales

El desarrollo de este trabajo ha resultado en un sistema capaz de integrar Planificación Automática con ROS para el control de robots autónomos.

El objetivo general se ha cumplido: el sistema demuestra, mediante resoluciones de problemas del dominio del *Sokoban* en un entorno real, que ha integrado correctamente ROS con PELEA para lograr el control del robot mediante acciones que vienen dadas por el plan obtenido para los problemas indicados.

El sistema es capaz de funcionar de manera que varios robots trabajan juntos, colaborando si es necesario. La cantidad de robots es irrelevante: se ha mostrado que para modificar el número de robots utilizados los cambios a realizar son mínimos.

Por todo esto, el objetivo general del trabajo queda verificado y su cumplimiento queda demostrado.

6.2 Conclusiones referentes a los objetivos

Con referencia a los objetivos específicos del trabajo, se obtienen las siguientes conclusiones:

- Se ha creado un controlador con ROS (concretamente, con rosjava) que permite el control del robot P3DX, teniendo en cuenta sus particularidades y posibilidades, necesario para problemas como el que ocupa el presente trabajo. El controlador es capaz de mover el robot como se requiera: avances de cualquier distancia y giros en ángulo

recto, todo con control para evitar la acumulación de errores. También es capaz de enviar todos los datos requeridos de los sensores desde el robot al módulo de planificación.

- Se ha modificado el dominio del *Sokoban* de manera que, además de problemas para un solo jugador, se pueden resolver problemas para varios jugadores que deben actuar coordinadamente, añadiendo el objeto “caja grande” y permitiendo la posibilidad de que dos jugadores colaboren para empujar ese tipo de cajas, cosa que sería imposible con un solo jugador.
- Se ha modificado la arquitectura PELEA para adaptarla a las acciones del dominio, así como a la información de los sensores que envía el módulo de control. De esta manera, la arquitectura conforma un módulo de planificación que indica al módulo de control qué acciones se deben realizar en cada momento.
- Se ha realizado una experimentación que verifica todos los requisitos pertinentes y demuestra que el sistema funciona como se espera, tanto en un entorno simulado como, como es realmente interesante, en un entorno real.
- Se ha desarrollado la presente documentación, que describe de forma detallada y precisa el sistema creado, así como el proceso seguido para su desarrollo.

Se puede decir que todos los objetivos propuestos al inicio del presente trabajo se han cumplido satisfactoriamente.

6.3 Trabajos futuros

Sobre el sistema desarrollado se pueden añadir diversas mejoras que harían su funcionamiento más rico y añadirían características nuevas a su ejecución. Algunas de esas mejoras se presentan a continuación:

- Realización de acciones en paralelo: el primer trabajo que se podría realizar sobre el sistema desarrollado sería la adición de la posibilidad de que los robots realicen simultáneamente las acciones que no entren en conflicto con las de los demás robots. Para ello, habría que cambiar el planificador a uno que permita esta opción, generando

planes paralelos en lugar de los planes secuenciales que crea Metric-FF, y adaptar los nodos de PELEA a esta nueva característica. También se podría considerar el uso de planificadores óptimos, secuenciales o paralelos. Metric-FF es un planificador no óptimo por lo que en los planes creados se podrían encontrar acciones redundantes o superfluas. Aunque la planificación óptima es bastante más complicada en la práctica que la planificación no óptima, el tamaño de los problemas que se pueden crear en un entorno real para el dominio del *Sokoban*, hace posible, al menos en teoría, su utilización.

- Uso de las lecturas de los sónares y la batería: el siguiente trabajo para mejorar las funcionalidades del sistema sería el uso de las lecturas de los sónares de forma que se pudiera saber si realmente se tiene una caja en un lugar determinado, añadiendo así la posibilidad de añadir obstáculos/cajas que no estuvieran definidas en el problema en PDDL y permitiendo a PELEA replanificar cuando fuera necesario. Para ello habría que interpretar en PELEA la información de los sónares recibida desde el módulo de control y determinar las posiciones de los obstáculos gracias a dicha información. De forma análoga, se podría utilizar el nivel de batería en PELEA para introducir en los planes la meta de intentar minimizar la batería gastada, o la posibilidad de replanificar si la batería es baja, de forma que el robot vuelva hacia el lugar donde se carga.
- Uso de cámaras y visión artificial: como última mejora propuesta, se podrían incluir cámaras conectadas a los robots de forma que, como añadido a la anterior mejora, se pudieran reconocer nuevos obstáculos y, por el color de estos, se supiera de qué tipo son y, quizá, que cada robot tenga un color asignado para las cajas que tiene que empujar, o cualquier opción del mismo tipo posible. También se podrían utilizar las cámaras para determinar que las cajas no están bien colocadas y colocarlas correctamente, utilizando acciones de bajo nivel.

Glosario

- Ciclo de vida de un proyecto: Proceso de desarrollo de un proyecto. Hay distintos tipos o metodologías de ciclos que se pueden utilizar para un proyecto: cascada, espiral, etc.
- Diagrama de Gantt: Herramienta gráfica para representar el tiempo de dedicación previsto para diferentes actividades o tareas en un tiempo determinado.
- Entorno de desarrollo: Programa compuesto por un conjunto de herramientas de programación, como pueden ser: editor de texto, compilador, intérprete, depurador e interfaz gráfica.
- Framework: Herramienta para el desarrollo y/o implementación de aplicaciones.
- IP (Dirección): Etiqueta numérica que identifica a un dispositivo dentro de una red que utilice el protocolo llamado IP.
- Java: Lenguaje de programación de alto nivel orientado a objetos. Su sintaxis deriva de C y C++, pero tiene menos utilidades de bajo nivel que ellos.
- JDK: Kit de Desarrollo de Java (*Java Development Kit*), es un software que proporciona un conjunto de herramientas para el desarrollo de programas en Java.
- NetBeans: Entorno de desarrollo creado principalmente para el lenguaje Java.
- Sistema multi-agente: Sistema que está compuesto por varios agentes inteligentes que interactúan entre ellos.
- Ubuntu: Sistema operativo basado en Linux y distribuido como software libre.

Bibliografía

1. **Adept MobileRobots.** Adept MobileRobots Pioneer 3-DX (P3DX). [Online] [Cited: Mayo 19, 2014.] http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx.

2. **ICAPS.** ICAPS Competitions. [Online] [Cited: Junio 16, 2014.] http://www.icaps-conference.org/index.php/Main/Competitions.

3. **Barrientos, Antonio.** *Fundamentos de Robótica.* Madrid : McGraw-Hill, 1997.

4. **Stanford Research Institute.** AI Center : Shakey. [Online] [Cited: Mayo 19, 2014.] http://www.ai.sri.com/shakey/.

5. **NASA.** Voyager - The Interstellar Mission. [Online] [Cited: Mayo 19, 2014.] http://voyager.jpl.nasa.gov/.

6. **Honda.** History of Honda's Robot Development. [Online] [Cited: Mayo 19, 2014.] http://world.honda.com/ASIMO/history/p1_p2_p3/index.html.

7. **Lego.** Lego.com Introducción a Mindstorms. [Online] [Cited: Mayo 19, 2014.] http://www.lego.com/es-es/mindstorms/gettingstarted/historypage/.

8. **Honda.** The Honda Worldwide ASIMO Site. [Online] [Cited: Mayo 19, 2014.] http://world.honda.com/ASIMO/.

9. **Sony.** Sony AIBO Europe - Official Website. [Online] [Cited: Mayo 19, 2014.] http://services.sony.co.uk/support/aibo/index.asp.

10. **RoboCup.** RoboCup 2014. [Online] Mayo 19, 2014. http://www.robocup2014.org/.

11. **Aldebaran Robotics.** Nao robot: intelligent and friendly companion | Aldebaran. [Online] [Cited: Mayo 19, 2014.] http://www.aldebaran.com/en/humanoid-robot/nao-robot.

12. **NASA.** Mars Science Laboratory. [Online] [Cited: Mayo 19, 2014.] http://mars.jpl.nasa.gov/msl/.

13. —. NASA - Super Ball Bot - Structures for Planetary Landing and Exploration. [Online] [Cited: Mayo 19, 2014.] http://www.nasa.gov/centers/ames/cct/technology/stp/earlystage/niac_superbot_prt.htm.

14. **ISO.** Online Browser Platform (OBP). [Online] [Cited: Junio 17, 2014.] https://www.iso.org/obp/ui/#iso:std:iso:13482:ed-1:v1:en.

15. **Norvig, Stuart J. Russell and Peter.** *Artificial Intelligence: A Modern Approach.* s.l. : Pearson, 2010.

16. **ICAPS.** ICAPS | Main. [Online] [Cited: Junio 16, 2014.] http://www.icaps-conference.org/.

17. **Helmert, Malte.** Fast Downward Homepage. [Online] [Cited: Mayo 19, 2014.] http://www.fast-downward.org/.

18. **Hoffmann, Joerg.** Metric-FF Homepage. [Online] [Cited: Mayo 19, 2014.] http://fai.cs.uni-saarland.de/hoffmann/metric-ff.html.

19. **Planning and Learning Group - UC3M.** PELEA - Planning & Learning Group - UC3M. [Online] [Cited: Mayo 19, 2014.] http://plg.inf.uc3m.es/pelea/index.php.

20. **Robolab.** Robocomp - An Open-Source Robotics Framework. [Online] [Cited: Mayo 19, 2014.] http://robocomp.sourceforge.net/wordpress/.

21. **Støy, Kasper.** Player Project. [Online] [Cited: Mayo 19, 2014.] http://playerstage.sourceforge.net/.

22. **Open Source Robotics Foundation.** ROS.org | Powering the world's robots. [Online] [Cited: Mayo 19, 2014.] http://www.ros.org/.

23. **Planning & Learning Group - UC3M.** PELEA - Planning & Learning Group - UC3M. [Online] [Cited: Junio 16, 2014.] http://www.plg.inf.uc3m.es/pelea/download.php.

Anexo A: Instalación y configuración del entorno

A continuación se mostrarán los pasos necesarios para poner en funcionamiento el sistema en el sistema operativo Ubuntu 12.04 LTS.

A.1 Instalación y configuración de ROS Hydro

Lo primero que se debe instalar es esta versión de ROS. Como prerrequisito, únicamente hay que tener git instalado. Una vez hecho esto, en primer lugar, hay que configurar el archivo sources.list, lo cual se hace mediante el siguiente comando:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" >
/etc/apt/sources.list.d/ros-latest.list'
```

Y configurar las claves:

```
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

Se comprueba que se tiene el sistema actualizado:

```
sudo apt-get update
```

Y se instala ROS Hydro (durante la instalación se preguntará si se quiere instalar *hddtemp*, en el caso de este trabajo no es necesario):

```
sudo apt-get install ros-hydro-desktop-full
```

Se inicializa *rosdep*, necesario para ejecutar algunos de los componentes de ROS:

```
sudo rosdep init
```

```
rosdep update
```

Se configuran las variables de entorno de ROS de forma que se añaden de forma automática cada vez que se inicia una terminal:

```
echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Se instala P2OS Driver, que es el *driver* del P3DX que se utiliza en este trabajo:

```
sudo apt-get install ros-hydro-p2os-driver ros-hydro-p2os-teleop ros-hydro-p2os-launch ros-hydro-p2os-urdf
```

Se crea el *workspace* a utilizar y se importa P2OS mediante git, compilando finalmente el *workspace* completo:

```
mkdir -p ~/ros_ws/src
cd ~/ros_ws/src
catkin_init_workspace
cd ~/ros_ws/
catkin_make
source devel/setup.bash
echo "source ~/ros_ws/devel/setup.bash" >> ~/.bashrc
cd src git clone https://github.com/allenh1/p2os
source ../devel/setup.bash
cd ~/ros_ws
catkin_make
```

Se instala *rosjava*:

```
sudo apt-get install ros-hydro-catkin ros-hydro-ros ros-hydro-rosjava python-wstool
```

Después, se introduce el contenido de la carpeta *ROS-INSTALLER* del trabajo (que contiene los paquetes del controlador de ROS y del mensaje personalizado) en la ruta:

```
~/ros_ws/src
```

Además, hay que cambiar el campo DEFAULT_P2OS_PORT utilizado en el archivo de la ruta

```
~/ros_ws/src/p2os/p2os_driver/include/robot_params.h
```

a "/dev/ttyUSB0" (por defecto, pero puede corresponder otro).

Por último, se compila una última vez el *workspace*:

```
cd ~/ros_ws
catkin_make
```

Y, con esto, toda la parte de ROS del trabajo estaría totalmente funcional.

A.2 Instalación y configuración de PELEA

A continuación, se procede a instalar la arquitectura de planificación PELEA [23]. Para ello, como prerrequisitos, se debe tener instalado el JDK versión 7 o superior. Para seguir este tutorial se ha utilizado el entorno de desarrollo NetBeans (aunque podría utilizarse otro, como por ejemplo Eclipse), por lo que también debe tenerse instalado en su versión 8.0 o superior.

En primer lugar, debe crearse un proyecto Java en NetBeans, desmarcando la opción de crear clase principal (*main*) (Figura 32).

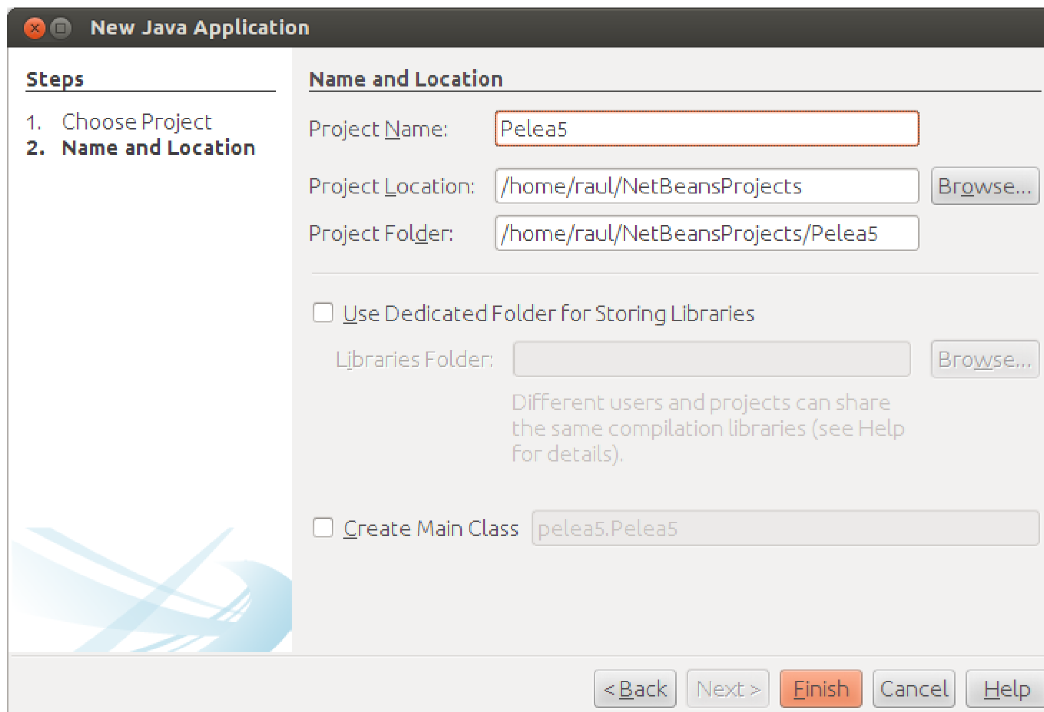


Figura 32 - Instalación de PELEA (I)

A continuación, se introduce el contenido de la carpeta *PELEA-INSTALLER* del trabajo en la carpeta del proyecto recién creado, sustituyendo si es necesario.

De nuevo en NetBeans, se incluyen las librerías necesarias (las de la carpeta *lib* del proyecto) (Figura 33) y se selecciona como clase principal el *main* (Figura 34) y, por último, se compila el proyecto.

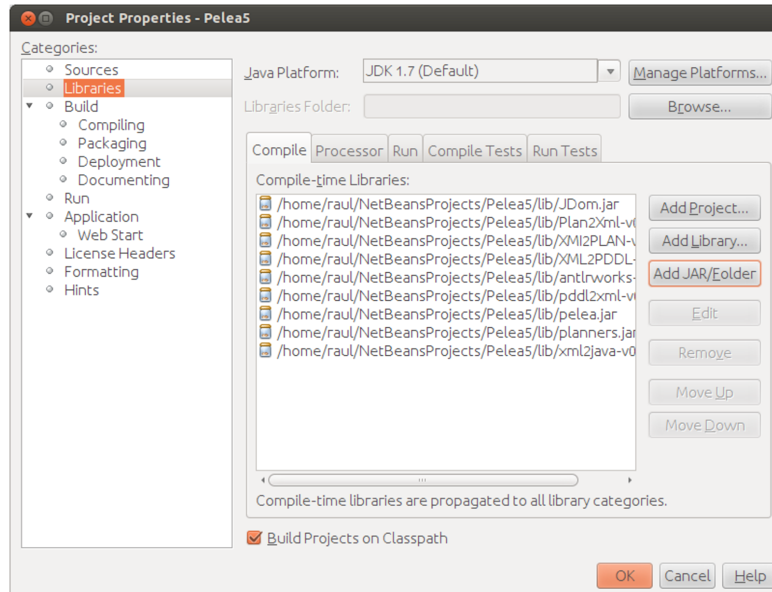


Figura 33 - Instalación de PELEA (II)

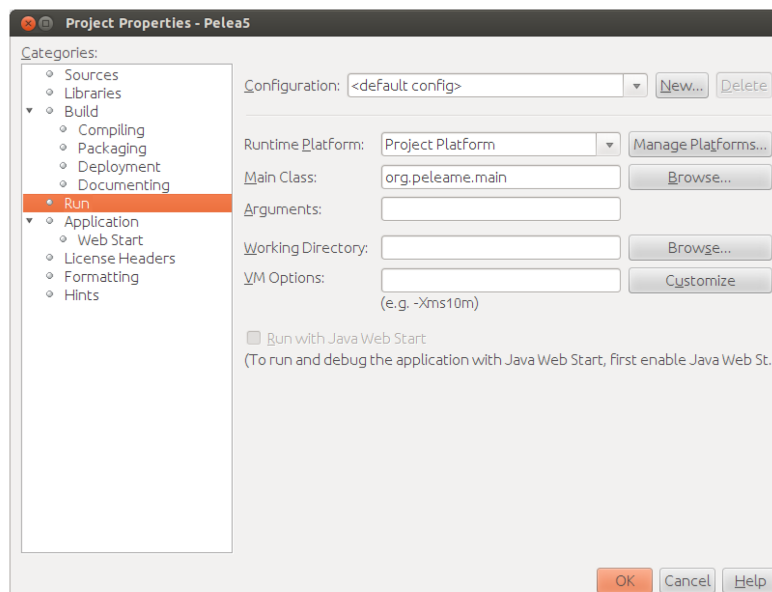
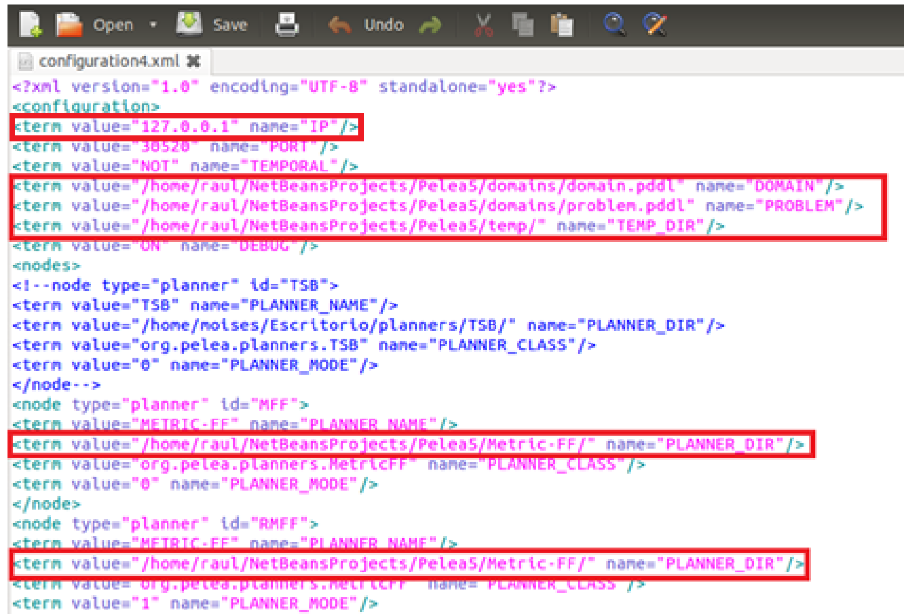


Figura 34 - Instalación de PELEA (III)

Además, en el archivo de configuración de la raíz del proyecto (*configuration4.xml*) hay que señalar las rutas propias de dominio, problema, carpeta temporal y planificadores, así como la IP en el caso del Pelea que se ejecuta en remoto, poniendo la IP del PC principal (Figura 35).



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
<term value="127.0.0.1" name="IP"/>
<term value="30520" name="PORT"/>
<term value="NOT" name="TEMPORAL"/>
<term value="/home/raul/NetBeansProjects/Pelea5/domains/domain.pddl" name="DOMAIN"/>
<term value="/home/raul/NetBeansProjects/Pelea5/domains/problem.pddl" name="PROBLEM"/>
<term value="/home/raul/NetBeansProjects/Pelea5/temp/" name="TEMP_DIR"/>
<term value="ON" name="DEBUG"/>
<nodes>
<!--node type="planner" id="TSB">
<term value="TSB" name="PLANNER_NAME"/>
<term value="/home/moises/Escritorio/planners/TSB/" name="PLANNER_DIR"/>
<term value="org.pelea.planners.TSB" name="PLANNER_CLASS"/>
<term value="0" name="PLANNER_MODE"/>
</node-->
<node type="planner" id="MFF">
<term value="METRIC-FF" name="PLANNER_NAME"/>
<term value="/home/raul/NetBeansProjects/Pelea5/Metric-FF/" name="PLANNER_DIR"/>
<term value="org.pelea.planners.MetricFF" name="PLANNER_CLASS"/>
<term value="0" name="PLANNER_MODE"/>
</node>
<node type="planner" id="RMFF">
<term value="METRIC-FF" name="PLANNER_NAME"/>
<term value="/home/raul/NetBeansProjects/Pelea5/Metric-FF/" name="PLANNER_DIR"/>
<term value="org.pelea.planners.MetricFF" name="PLANNER_CLASS"/>
<term value="1" name="PLANNER_MODE"/>
```

Figura 35 - Instalación de PELEA (IV)

Con esto, la parte de PELEA del trabajo estaría totalmente funcional.

Anexo B: Manual de uso del sistema

A continuación se mostrarán los pasos necesarios para utilizar el sistema desarrollado en el presente trabajo. El único uso que se va a dar y mostrar es la resolución de un problema de *Sokoban* multi-agente en el mundo real.

En primer lugar, debe haberse seguido el manual de instalación del sistema, incluyendo la copia de los scripts proporcionados (se muestran en el Anexo D.3) a la carpeta personal del usuario en Ubuntu.

Debe incluirse en la carpeta *domains* de Pelea del PC principal el problema en PDDL que se quiera solucionar, o utilizar uno de los que se proporcionan de ejemplo. Los problemas deben escribirse de acuerdo al dominio proporcionado. Si se utiliza un nombre distinto a *problem.pddl* debe indicarse en el archivo *configuration4.xml* de Pelea. También debe indicarse aquí el número de *Executions* a utilizar (el mismo que el número de robots a utilizar; es el segundo valor del atributo *Network* del nodo *Monitoring*).

También se debe cambiar, en los dos PCs a utilizar, la IP del archivo /etc/hosts que corresponde al nombre del equipo, indicando la IP dada por el router (ambos PCs deben estar conectados al mismo, la IP se puede consultar con el comando *ifconfig*) en lugar de la IP local. Además, en el PC que ejecutará el *execution* remoto debe cambiarse, como se mostró en el manual de instalación, la IP (por defecto, la local) por la IP del PC principal.

Posteriormente, ya en el entorno real del problema, con las cajas y casillas correspondientes, se colocan los robots a utilizar en el problema en su posición inicial, teniendo en cuenta que la orientación que tengan al principio será considerada como orientación "hacia la derecha", es decir, positiva en el eje de las X.

Cada robot se conecta a un PC utilizando un cable de serie a USB. Se deben dar los permisos necesarios al puerto USB utilizado. Tomando como ejemplo el puerto por defecto, se usaría el comando:

```
sudo chmod 777 /dev/ttyUSB0
```

Por último, se encienden los robots y se ejecuta, en primer lugar, en el PC principal el script *ROS.sh* y en el secundario *ROSB.sh* y, cuando los robots estén activos (producirán un sonido) se deben ejecutar, en este orden, en el PC principal el script *Pelea5.sh*, y en el secundario el script *Pelea5B.sh*.

Tras esto, solo queda observar como los robots solucionan el problema proporcionado.

Anexo C: Resumen en inglés

En este anexo se presenta un resumen del documento en el idioma inglés, haciendo hincapié en los apartados de introducción y conclusiones.

C.1 Introduction

During the last years, robotics is changing. Formerly, robots only carried out repetitive and monotonous tasks but, lately, they have started to pursue much more ambitious goals. It is what it is called intelligent robotics.

Intelligent robotics is a branch of robotics that, nowadays, is at its peak thanks to the advance as much in hardware as in software. The "intelligence" of these robots comes from techniques that are not new, but that were not used until recently in this field, like Automated Planning (AP)

Despite AP initially appears as an answer to a problem set out by robotics (the autonomous control of *Shakey* the robot), for many years they have been two disciplines completely separated, mainly due to lack of maturity. Nevertheless, recently we have reached a point of development where trying this integration will make sense again.

It is easy to see the utility of this kind of robots in real world problems, when the NASA uses them in all their missions, where they want to obtain information of Earth's external environment.

Furthermore, a type of system that is gaining recognition lately is the multi-agent one. The most known example of multi-agent intelligent robotics is the annual competition Robocup, where a group of robots play football using AI techniques.

However, the multi-agent intelligent robotics has not only research or ludic purposes but, solving problems such as the ones presented in the present work, we can see that the possibilities of this kind of systems are, until very recently, greater than we could imagine just some years ago.

C.1.1 Description of the problem

The problem to be solved by this work is troubleshooting the famous *Sokoban* puzzle in a real environment with 2 players, where each one will be a PD3X robot controlled by Automated Planning.

Figure 1 illustrates an example of a *Sokoban* screen (in Japanese, warehouse manager) with two players. This puzzle is about pushing all the boxes into the squares marked as a goal, the player only being able to push the boxes, not pull them. The player can just move horizontally or vertically, only when the square he wants to move is empty or has a box but the square below is empty, the movement resulting in the square change of both the player and the box. Additionally, it must be achieved with the fewest number of steps and thrusts.

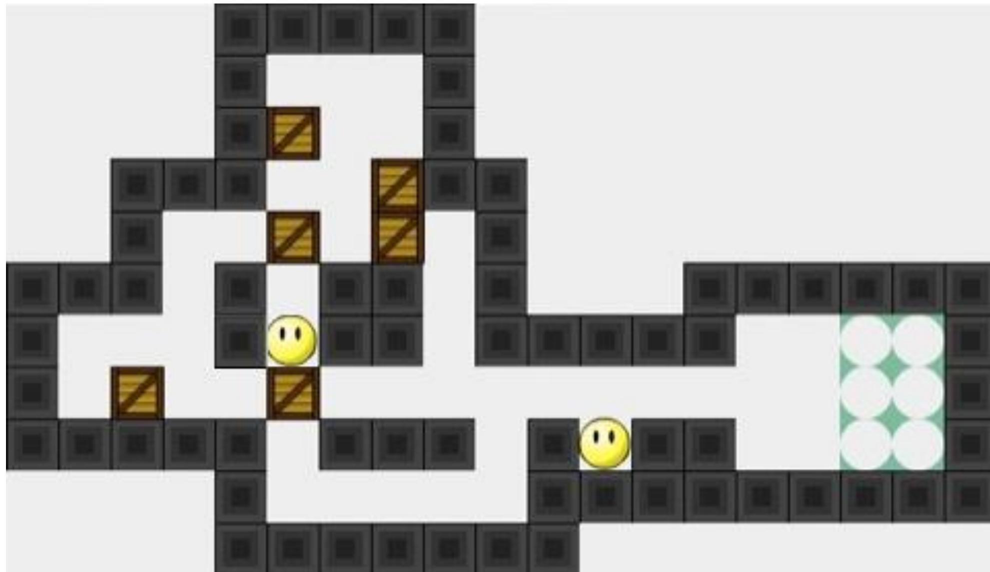


Figure 1 - Example of a problem in *Sokoban*

As described above, this work will be applied on a real environment. That is to say, each of the players will be a robot and they will have to push a set of boxes. For this, the structure of the classic *Sokoban* domain will be modified, allowing two players to solve the problem simultaneously, executing some actions concurrently.

The robots used in this project will be P3DX Pioneer. The P3DX is one of the most popular robots for research and teaching. It is a small box about 45x38x24cm with two wheels, which can

reach 1.2 m / s linear velocity and 300 degrees / s rotation speed. It consists of sonar and collision detectors (bumpers) that provide information about their environment [1]. Figure 2 illustrates a couple of them.

Figure 2 - P3DX Robots

C.1.2 Motivation

As mentioned in the introduction, intelligent robotics is growing up at present. Tasks that required human intelligence but now are beginning to be carried out by robots, are becoming more frequent.

Automated Planning is one of the most applied techniques to achieve this intelligence in robots, as their characteristics are easily adapted to something understandable by a robot, possibly because the AP was originally designed for this task.

Nonetheless, we can add a feature to intelligent robotics that immensely increases its potential, being a multi-agent system. For instance, the specific subject of the work, the *Sokoban*, is quite interesting nowadays since really big and important companies (such as Amazon) use this kind of systems to organize their warehouses, so it is totally at the cutting edge of current technology.

C.1.3 Goals of the work

The main goal of this work consists in the development of a system that integrates Automated Planning with ROS (Robot Operating System) for autonomous robots control. As evidence of its correct operation, this system must allow two P3DX type robots to solve problems of the *Sokoban* domain. As specific goals, it can be considered the following:

- Familiarization with the ROS framework: in order to perform any action with the robots, first we must know how to use a kind of system to connect with them and access to their functionalities. ROS is the chosen system.
- Familiarization with the P3DX robot: besides ROS, we must know the singularities of the robot we will use in this study, as well as the possibilities it offers.
- Reformulation of *Sokoban* domain: the *Sokoban* domain, as defined in the International Planning Competition (IPC) is not suitable for this work, since it only concerns one player, so we will have to make minor modifications to it.
- Control system development: there must be a system that is able to make the robot perform the desired movements and returns the information from its sensors when required.
- Integration with AP systems: the problem of this work requires the AP to determine which actions should perform the robot using the information obtained from the sensors.
- Experimentation: once the system is developed, experiments should be performed to determine if the result has been satisfactory.
- Development of the documentation: everything done during the development of the work must be properly reflected both herein and in the presentation made in dissertation.

C.1.4 Summary structure

This summary is divided into 3 chapters. A brief description of each is shown below:

- Chapter 1

  This chapter contains an introduction to the work presented as well as the problem that has been solved, showing work motivation, listing the objectives proposed for the realization of this along with a brief description of each one, and finally, making this description of the chapters of the summary.
- Chapter 2

  In this chapter it is shown the description of the system, separated in the descriptions of the analysis and design, which will explain, respectively, the characteristics to be met by the system and how it is formed.
- Chapter 3

  The conclusions obtained after completion of the work as well as the enumeration of possible future works on the system are presented in this chapter.

C.2 System description

In this section, it is shown a description of the system that has been developed for this work. It is a system that allows troubleshooting multi-agent *Sokoban* in the real world with P3DX robots, using the ROS operating system and the planning architecture PELEA.

The analysis and design steps that have been followed for the development of the system in this document are presented below.

C.2.1 System analysis

The system developed in this work must be able to solve multi-agent problems from the *Sokoban* domain with physical P3DX robots. To do this, it must have two main features.

The first one is the control of robots. The system must be able to control the movements of the robots (move and rotate) and retrieve information from their sensors (sonar readings, odometry information (position and velocity), etc.).

The second one is to obtain the actions that solve the problem. The system must be able to read the PDDL domain and problem files, to generate a sequence of actions to solve them, and after the execution of each action by the control module, to check that the result of the action is the expected one.

In addition, the classic *Sokoban* domain must be modified with the possibility of several players solving the problem together, adding the object "double box" or "big box", which has the characteristics of occupying two adjacent squares and two robots needed to push on it. This item has been added because the domain of *Sokoban* with the simple inclusion of multiplayer is very similar to *Sokoban* with a single player, and thus the actual potential for a multi-agent system is displayed.

C.2.2 System design

The system consists of two major modules: the control module and the planning module.

The control module is responsible for dealing directly with the robot, obtaining data from its sensors and commanding it to move in the required manner. Meanwhile, the planning module deals with providing the actions that are required to the control module. To do this, a sequence of actions is obtained by solving the problem given by the state of the world generated from the problem in PDDL and the sensor data provided by the control module. Next, each action is sent to the control module and when the planning module receives a notice that the control module is over, it sends the following action; and so on until the plan is fulfilled. If one action fails, the planning module creates a new plan that solves the problem that occurs when an unexpected state of the world is found.

In Figure 3 the general diagram of the system architecture can be seen. The two mentioned modules and their main components are shown on it, as well as the elements that are exchanged between them.

C.2.2.1 Control module

The control module has been developed under ROS. Therefore it uses the internal communication system based on topics from ROS. It consists of three main parts: the controller sub-module, the sub-module that deals with the movement and the sub-module that deals with the sensors. Each of them is a Java class. It is shown on the right side of Figure 3.

C.2.2.2 Planning module

The planning module uses PELEA architecture. The architecture used has five nodes: Monitoring, Decision Support, Execution, Low to High and Low Lever Planner. For this work only the last three nodes have had to be modified, although minor modifications had to be made to the Monitoring for proper control of more than one robot. It is shown on the left side of Figure 3.

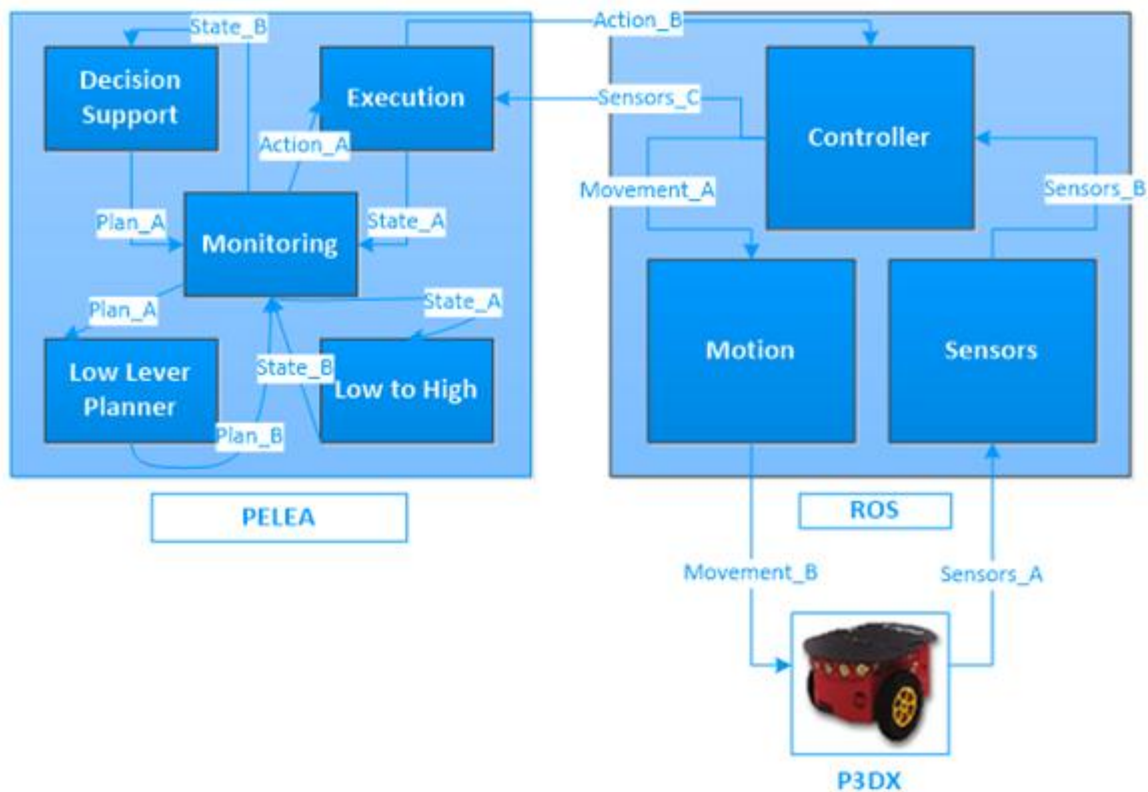


Figure 3 - General diagram of the system architecture

C.2.2.3 Sokoban *domain mofication*

It has been necessary to modify the PDDL domain of *Sokoban* to allow taking joint actions given that in the utilized domain it was possible to use several robots at the same time, but without joint actions. It was required just to add variations of the actions that move in the original domain in which the preconditions include that the two robots are together and have a big box in their front and that both squares after that big box are free; and in the effects, analogous to traditional actions, that the starting squares are empty, the robots are where the box was and the box is in the two previously empty squares. To make this easier, a new type of object has been included in the domain to represent the big/double boxes (*big* object), and specific *at* and *at-goal* predicates for it, called *big-at* and *big-at-goal* have been included as well. The original ones could have been adapted, but it was done this way for simplicity.

C.3 Conclusions and future work

This section describes the conclusions obtained after completion of the work. It begins with the general conclusions of the work, followed by conclusions concerning the objectives set at the beginning, and ending with a description of possible future works to be performed on the developed system.

C.3.1 General conclusions

The development of this work has resulted in a system capable of integrating Automated Planning with ROS to control autonomous robots.

The overall objective has been fulfilled: the system demonstrates by solving problems of *Sokoban* domain in a real environment that has successfully integrated ROS with PELEA to achieve control of the robot by actions given by the plan obtained for the identified problems.

The system is able to operate so that multiple robots work together collaborating if necessary. The number of robots is irrelevant: it has been shown that to modify the number of robots used, the changes are minimal.

Taking into account the former, the general objective of this work is verified and its fulfillment is demonstrated.

C.3.2 Conclusions concerning the objectives

Referring to the specific objectives of the study, the following conclusions are obtained:

- It has been created a driver with ROS (specifically, rosjava) that allows to control the P3DX robot, considering its features and capabilities necessary for problems such as the present work. The controller is able to move the robot as required: to progress in any distance and right angle turns, all of them controlled to prevent accumulation of errors. It is also capable of sending all sensors data required by the robot to the planning module.
- The *Sokoban* domain has been modified so that, in addition to problems for a single player, problems for several players can be solved, adding the "big box" object and allowing the possibility of two players collaborating to push that kind of boxes , which would be impossible with just a single player.
- PELEA architecture has been modified to adapt it to actions of the domain as well as to the information from the sensors that the control module sends.
- It has been performed an experimentation which verifies all relevant requirements and shows that the system works as expected in both simulated and, most interesting, real environments.
- This documentation has been developed, describing in detail and accurately the system created as well as the process carried out for its development.

It can be said that all the objectives proposed at the beginning of this work have been satisfactorily completed.

C.3.3 Future work

Various enhancements that would make the performance of the developed system richer and would add new features to its execution can be considered. Some of these improvements are presented below:

- Performing actions in parallel: the first work that could be performed on the developed system would be adding the possibility that robots simultaneously perform actions that do not conflict with those of other robots. For this, we need to change the planner to one that enables this option and adapt PELEA nodes to this new feature. Also it could be considered the use of optical planners, whether they are sequential or parallel. Metric-FF is a non-optimal planner, so the plans created by it could have redundant or superfluous actions. Although optimal planning is much more complicated in practice than non-optimal planning, the size of problems that can be created in a real environment for the domain of *Sokoban* makes it possible, at least in theory, to be used.
- Using the readings of the sonar and the battery: the following work to improve the system's functionality would be to use readings of the sonar so that we could know if we really have a box in a certain place, thus adding the possibility of having obstacles / boxes that were not defined in the PDDL problem and allowing PELEA replan when necessary. This would require PELEA to interpret the sonar information received from the control module and to determine the positions of obstacles thanks to this information. Similarly, the battery level in PELEA could be used to introduce in the plans the goal of trying to minimize the battery consumption, or the ability to replan if the battery is low, so that the robot comes back to the place where it is loaded.
- Use of cameras and artificial vision: as last proposed improvement we could include cameras attached to the robots so that, as an addition to the above improvement, new obstacles could be recognized and, by the color of these, we would know what kind they are. And perhaps, that each robot would have a color assigned to the boxes it has to push, or any possible option of the same type.

Anexo D: Documentos de interés

En este anexo se presentan los mensajes, dominios y scripts de ejecución utilizados.

D.1 Mensajes

El presente apartado contiene los mensajes de ROS utilizados, el de movimiento del robot y el de los sensores (personalizado) utilizados en el trabajo.

D.1.1 geometry_msgs/Twist

```
geometry_msgs/Vector3 linear
            float64 x
            float64 y
            float64 z
geometry_msgs/Vector3 angular
            float64 x
            float64 y
            float64 z
```

D.1.2 myjava_msgs/SensorsData

```
p2os_driver/SonarArray sonar
      int32 ranges_count
      float64[] ranges
p2os_driver/MotorState motor
      int32 state
p2os_driver/BatteryState battery
      float32 voltage
nav_msgs/Odometry odometry
      geometry_msgs/PoseWithCovariance pose
          geometry_msgs/Pose pose
              geometry_msgs/Point position
                  float64 x
```

```
                float64 y
                float64 z
            geometry_msgs/Quaternion orientation
                float64 x
                float64 y
                float64 z
                float64 w
        float64[36] covariance
    geometry_msgs/TwistWithCovariance twist
        geometry_msgs/Twist twist
            geometry_msgs/Vector3 linear
                float64 x
                float64 y
                float64 z
            geometry_msgs/Vector3 angular
                float64 x
                float64 y
                float64 z
        float64[36] covariance
```

D.2 Dominios

El presente apartado contiene los dominios utilizados en el trabajo: el dominio original del *Sokoban* y el modificado para incluir cajas grandes y acciones para ellas.

D.2.1 *Sokoban* original

```
(define (domain sokoban-sequential)
  (:requirements :typing )
  (:types thing location direction - object
          player stone - thing)
  (:predicates (clear ?l - location)
             (at ?t - thing ?l - location)
             (at-goal ?s - stone)
             (IS-GOAL ?l - location)
             (IS-NONGOAL ?l - location)
              (MOVE-DIR ?from ?to - location ?dir - direction))
  (:action move
   :parameters (?p - player ?from ?to - location ?dir - direction)
   :precondition (and (at ?p ?from)
                      (clear ?to)
                      (MOVE-DIR ?from ?to ?dir)
                      )
   :effect       (and (not (at ?p ?from))
                      (not (clear ?to))
                      (at ?p ?to)
                      (clear ?from)
                      )
   )
  (:action push-to-nongoal
   :parameters (?p - player ?s - stone
                ?ppos ?from ?to - location
                ?dir - direction)
   :precondition (and (at ?p ?ppos)
                      (at ?s ?from)
```

```
                      (clear ?to)
                      (MOVE-DIR ?ppos ?from ?dir)
                      (MOVE-DIR ?from ?to ?dir)
                      (IS-NONGOAL ?to)
                      )
   :effect       (and (not (at ?p ?ppos))
                      (not (at ?s ?from))
                      (not (clear ?to))
                      (at ?p ?from)
                      (at ?s ?to)
                      (clear ?ppos)
                      (not (at-goal ?s))
                      )
   )
  (:action push-to-goal
   :parameters (?p - player ?s - stone
                ?ppos ?from ?to - location
                ?dir - direction)
   :precondition (and (at ?p ?ppos)
                      (at ?s ?from)
                      (clear ?to)
                      (MOVE-DIR ?ppos ?from ?dir)
                      (MOVE-DIR ?from ?to ?dir)
                      (IS-GOAL ?to)
                      )
   :effect       (and (not (at ?p ?ppos))
                      (not (at ?s ?from))
                      (not (clear ?to))
                      (at ?p ?from)
                      (at ?s ?to)
                      (clear ?ppos)
                      (at-goal ?s)
                      )
   )
)
```

D.2.2 *Sokoban* modificado

```
(define (domain sokoban-twoplayers)
  (:requirements :typing )
  (:types thing location direction - object
          player stone big - thing)
  (:predicates (clear ?l - location)
             (at ?t - thing ?l - location)
             (at-goal ?s - stone)
             (IS-GOAL ?l - location)
             (IS-NONGOAL ?l - location)
              (MOVE-DIR ?from ?to - location ?dir - direction)
             (big-at ?b - big ?l - location)
             (big-at-goal ?b - big))

  (:action move
   :parameters (?p - player ?from ?to - location ?dir - direction)
   :precondition (and (at ?p ?from)
                      (clear ?to)
                      (MOVE-DIR ?from ?to ?dir)
                      )
   :effect       (and (not (at ?p ?from))
                      (not (clear ?to))
                      (at ?p ?to)
                      (clear ?from)
                      )
   )

  (:action push-to-nongoal
   :parameters (?p - player ?s - stone
                ?ppos ?from ?to - location
                ?dir - direction)
   :precondition (and (at ?p ?ppos)
                      (at ?s ?from)
                      (clear ?to)
                      (MOVE-DIR ?ppos ?from ?dir)
```

```
                    (MOVE-DIR ?from ?to ?dir)
                    (IS-NONGOAL ?to)
                    )
 :effect       (and (not (at ?p ?ppos))
                    (not (at ?s ?from))
                    (not (clear ?to))
                    (at ?p ?from)
                    (at ?s ?to)
                    (clear ?ppos)
                    (not (at-goal ?s))
                    )
 )


(:action push-to-goal
 :parameters (?p - player ?s - stone
              ?ppos ?from ?to - location
              ?dir - direction)
 :precondition (and (at ?p ?ppos)
                    (at ?s ?from)
                    (clear ?to)
                    (MOVE-DIR ?ppos ?from ?dir)
                    (MOVE-DIR ?from ?to ?dir)
                    (IS-GOAL ?to)
                    )
 :effect       (and (not (at ?p ?ppos))
                    (not (at ?s ?from))
                    (not (clear ?to))
                    (at ?p ?from)
                    (at ?s ?to)
                    (clear ?ppos)
                    (at-goal ?s)
                    )
 )
```

```
(:action push-big-to-nongoal
:parameters (?p1 ?p2 - player ?b - big
             ?ppos1 ?ppos2 ?from1 ?from2 ?to1 ?to2 - location
             ?dir ?dir2 - direction)
:precondition    (and   (at ?p1 ?ppos1)
                        (at ?p2 ?ppos2)
                        (MOVE-DIR ?ppos1 ?ppos2 ?dir2)
                        (big-at ?b ?from1)
                        (big-at ?b ?from2)
                        (clear ?to1)
                        (clear ?to2)
                        (MOVE-DIR ?ppos1 ?from1 ?dir)
                        (MOVE-DIR ?from1 ?to1 ?dir)
                        (MOVE-DIR ?ppos2 ?from2 ?dir)
                        (MOVE-DIR ?from2 ?to2 ?dir)
                        (IS-NONGOAL ?to1)
                        (IS-NONGOAL ?to2)
                    )
:effect          (and   (not (at ?p1 ?ppos1))
                        (not (at ?p2 ?ppos2))
                        (not (big-at ?b ?from1))
                        (not (big-at ?b ?from2))
                        (not (clear ?to1))
                        (not (clear ?to2))
                        (at ?p1 ?from1)
                        (at ?p2 ?from2)
                        (big-at ?b ?to1)
                        (big-at ?b ?to2)
                        (clear ?ppos1)
                        (clear ?ppos2)
                        (not (big-at-goal ?b))
                    )
)
```

```
   (:action push-big-to-goal
   :parameters (?p1 ?p2 - player ?b - big
                ?ppos1 ?ppos2 ?from1 ?from2 ?to1 ?to2 - location
                ?dir ?dir2 - direction)
   :precondition    (and   (at ?p1 ?ppos1)
                           (at ?p2 ?ppos2)
                           (MOVE-DIR ?ppos1 ?ppos2 ?dir2)
                           (big-at ?b ?from1)
                           (big-at ?b ?from2)
                           (clear ?to1)
                           (clear ?to2)
                           (MOVE-DIR ?ppos1 ?from1 ?dir)
                           (MOVE-DIR ?from1 ?to1 ?dir)
                           (MOVE-DIR ?ppos2 ?from2 ?dir)
                           (MOVE-DIR ?from2 ?to2 ?dir)
                           (IS-GOAL ?to1)
                           (IS-GOAL ?to2)
                       )
   :effect          (and   (not (at ?p1 ?ppos1))
                           (not (at ?p2 ?ppos2))
                           (not (big-at ?b ?from1))
                           (not (big-at ?b ?from2))
                           (not (clear ?to1))
                           (not (clear ?to2))
                           (at ?p1 ?from1)
                           (at ?p2 ?from2)
                           (big-at ?b ?to1)
                           (big-at ?b ?to2)
                           (clear ?ppos1)
                           (clear ?ppos2)
                           (big-at-goal ?b)
                       )
   )
)
```

D.3 Scripts de ejecución

El presente apartado contiene los scripts de ejecución utilizados para ejecutar ROS y PELEA en los PCs requeridos.

D.3.1 Script ROS.sh

```
cd ~/ros_ws/src/rosjava_controller/p3dx/
gnome-terminal \
--tab -e "bash -c \"roslaunch ../../p2os/p2os_driver/launch/p2os_driver.launch;
exec bash\"" --title "P2OS - ROS" \
--tab  -e  "bash  -c  \"echo  Esperando...;sleep  8;./build/install/p3dx/bin/p3dx
org.ros.p3dx_controller.Motion; exec bash\"" --title "Motion - ROS" \
--tab  -e  "bash  -c  \"echo  Esperando...;sleep  8;./build/install/p3dx/bin/p3dx
org.ros.p3dx_controller.Sensors; exec bash\"" --title "Sensors - ROS" \
--tab  -e  "bash  -c  \"echo  Esperando...;sleep  8;./build/install/p3dx/bin/p3dx
org.ros.p3dx_controller.Controller; exec bash\"" --title "Controller - ROS"
```

D.3.2 Script PELEA5.sh

```
cd ~/NetBeansProjects/Pelea5/
gnome-terminal \
--tab -e "bash -c \"echo Esperando...;sleep 16;java -jar ./dist/Pelea5.jar -c
./configuration4.xml -n LLP -t 5 -m 2; exec bash\"" --title "Low-level planner -
Pelea" \
--tab -e "bash -c \"echo Esperando...;sleep 12;java -jar ./dist/Pelea5.jar -c
./configuration4.xml -n L2H -t 6 -m 2; exec bash\"" --title "LowToHigh - Pelea"
\
--tab -e "bash -c \"echo Esperando...;sleep 8;java -jar ./dist/Pelea5.jar -c
./configuration4.xml -n PLAYER-01 -t 2 -m 2; exec bash\"" --title "Execution -
Pelea" \
--tab -e "bash -c \"echo Esperando...;sleep 4;java -jar ./dist/Pelea5.jar -c
./configuration4.xml -n DS1 -t 1 -m 2; exec bash\"" --title "Decision support -
Pelea" \
```

```
--tab -e "bash -c \"java -jar ./dist/Pelea5.jar -c ./configuration4.xml -n M1 -t 7 -m 1; exec bash\"" --title "Monitoring - Pelea"
```

D.3.3 Script PELEA5B.sh (segundo PC)

```
cd ~/NetBeansProjects/Pelea5/
gnome-terminal \
--tab -e "bash -c \"echo Esperando...;sleep 8;java -jar ./dist/Pelea5.jar -c ./configuration4.xml -n PLAYER-02 -t 2 -m 2; exec bash\"" --title "Execution - Pelea"
```