View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Universidad Carlos III de Madrid e-Archivo

Universidad Carlos III de Madrid

Escuela Politécnica Superior

Bachelor's degree in Computer Science

Bachelor's Thesis

Solving Multi-agent Planning Tasks by Using Automated Planning

Author: Sofía Herrero Villarroya

Advisors: Nerea Luis Mingueza

Moisés Martínez Muñoz

Acknowledgements

A huge thank you to:

my parents, Ángel Luis and Lucía, for a lifetime of love and support;

my sister, Beatriz, for always being there;

Jose Carlos, for helping me when you didn't have to;

Moisés and Nerea, for opening me a door to this world;

the friends I have made in university, for making this an even better experience;

everyone that has encouraged me and has made this project possible, thank you.

Abstract

This dissertation consists on developing a control system for an autonomous multi-agent system using Automated Planning and Computer Vision to solve warehouse organization tasks.

This work presents an heterogeneous multi-agent system where each robot has different capabilities. In order to complete the proposed task, the robots will need to collaborate. On one hand, there are coordinator robots that collect information about the boxes to get their destination storage position using Computer Vision. On the other hand, there are cargo robots that push the boxes more easily than the coordinators but they have no camera devices to identify the boxes. Then, both robots must collaborate in order to solve the warehouse problem due to the different sensors and actuators that they have available.

This work has been developed in Java. It uses JNAOqi to communicate with the NAO robots (coordinators) and rosjava to communicate with the P3DX robots (cargos). The control modules are deployed in the PELEA architecuture. The empirical evaluation has been conducted in a real environment using two robots: one NAO8 Robot and one P3DX robot.

Resumen

Este trabajo presenta el desarrollo de un sistema de control para un sistema autónomo multi-agente con Planificación Automática y Visión Artificial para resolver tareas de ordenación de almacenes.

En el proyecto se presenta un sistema multi-agente heterogéneo donde cada agente tiene diferentes habilidades. Para poder completar la tarea propuesta, los agentes, en este caso robots, deben colaborar. Por un lado, hay robots coordinadores que recogen información de las cajas medinte Visión Artificial para conocer la posición de almacenaje de la caja. Por otro lado, hay robots de carga que empujan las cajas hasta su destino con mayor facilidad que los coordinadores pero que no tienen cámaras de video para identificar las cajas. Por ello, ambos robots tienen que colaborar para resolver el problema de ordenación debido a los diferentes sensores y actuadores que tienen disponibles.

El proyecto se ha desarrollado en Java. Se ha utilizado JNAOqi para comunicarse con los robots NAO (coordinadores) y rosjava para comunicarse con los robots P3DX (carga). La evaluación empírica se ha realizado en un entorno real utilizando dos robots: un robot NAO y un robot P3DX.

Contents

1 Introduction **1**

1.1 Problem description 3

1.2 Motivation 5

1.3 Objectives 5

1.4 Document structure 6

2 State of the art **9**

2.1 Software Agents 9

2.1.1 Types of agents 11

2.1.2 Robots 12

2.2 Control systems for autonomous agents 18

2.2.1 Deliberative Control Systems 18

2.2.2 Reactive Control Systems 19

2.2.3 Hybrid Control Systems 20

2.2.4 PELEA 22

2.3 Automated Planning 25

2.3.1 Conceptual model 25

2.3.2 Modelling language 26

2.3.3 Algorithms 27

2.4 Computer Vision 28

2.4.1 Colour spaces 28

2.4.2 Histograms 29

2.4.3 Blobs 30

2.4.4 QR 31

3 System description **33**

3.1 Introduction . . . 33

3.2 System analysis . . . 35

3.2.1 Functional characteristics description . . . 35

3.2.2 System restrictions . . . 35

3.2.3 Operating environment . . . 37

3.2.4 Use case specification . . . 38

3.2.5 Requirements specification . . . 48

3.3 System design . . . 59

3.3.1 Problem design . . . 59

3.3.2 Computer Vision module . . . 61

3.3.3 System architecture . . . 64

3.3.4 Components description . . . 65

3.3.5 System operation . . . 78

4 Experiments **81**

4.1 Experimentation environment . . . 81

4.2 NAO unitary experiments . . . 82

4.2.1 Vision experiments . . . 83

4.2.2 Movement skills experiments . . . 90

4.3 System experiments . . . 95

4.3.1 Experiment 1: Problem 1 . . . 95

4.3.2 Experiment 3: Problem 2 . . . 100

4.3.3 Experiment 4: Problem 3 . . . 103

4.3.4 System experimentation conclusions . . . 105

5 Project management **107**

5.1 Phases description . . . 107

5.2 Planning . . . 109

5.3 Budgeting . . . 112

6 Conclusions **115**

6.1 General conclusions 115

6.2 Conclusions concerning the objectives 117

6.3 Future work 118

A Installation **121**

A.1 JDK 121

A.2 JNAOqi and NAO environment 122

A.3 ZXing 122

A.4 ROS 122

A.5 PELEA 124

B User guide **125**

C PDDL source code and scripts **127**

List of Figures

1.1 Example of the Sokoban game. (Source) 3
1.2 Problem environment. 4

2.1 Interaction between agent and its environment. 10
2.2 Robot built with LEGO NTX Mindstorm kit. (Source) 13
2.3 The ASIMO robot. (Source) 13
2.4 The Romeo robot. (Source) 14
2.5 The Atlas robot in three different positions. (Source) 15
2.6 The Pepper robot. (Source) 15
2.7 Pioneer 3 DX robot. (Source) 16
2.8 Pioneer 3 DX robot dimensions. (Source) 17
2.9 NAO robot.(Source) 17
2.10 Basic structure of a deliberative architecture. 18
2.11 Basic structure of a reactive architecture. 20
2.12 Basic structure of a hybrid architecture. 21
2.13 PELEA architecture with variables. (Source) 22
2.14 R, G and B Histograms. The x-axis represents the intensity value for the colour and the y-axis the number of pixels with that intensity. (Source) 30
2.15 Original image (a) and image with red blolb detection (b). 31
2.16 QR illustration. (Source) 32

3.1 System high level diagram. 34
3.2 Use Case diagram. 39

3.3 System architecture diagram. 65
3.4 Components and connections for the NAO Execution module. . . 67
3.5 Flowchart for the turn correction algorithm. 69
3.6 Initial position (a) 90 degrees turn with error (b) Position after correction (c). 70
3.7 Flowchart for the walk correction algorithm. 71
3.8 Flowchart for the colour detection algorithm. 73
3.9 Components and connections for the P3DX Execution module. . . 75
3.10 Stages for the push action. 77
3.11 Sequence diagram for the system operation. 80

4.1 Environment diagrams legend. 82
4.2 Experiment 1 environment diagram. 83
4.3 Experiment 1 real environment. 84
4.4 Experiment 1 NAO pictures. 84
4.5 Experiment 2 environment diagram. 85
4.6 Experiment 2 real environment. 85
4.7 Experiment 2 NAO pictures. 86
4.8 Experiment 3 environment diagram. 87
4.9 Experiment 3 real environment. 87
4.10 Experiment 3 NAO pictures. 88
4.11 Boxes with different QR size. 88
4.12 Experiment 1 environment diagram. 91
4.13 Walk action error in situations (1) and (2). 92
4.14 Experiment 2 environment diagram. 93
4.15 NAO position in front of the box in an unsuccessful execution. . . 94
4.16 Experiment 2 modified diagram. 94
4.17 Problem 1 environment diagram. 96
4.18 Problem 1 plan. 97
4.19 Problem 1.1 plan. 99
4.20 Problem 2 environment diagram. 100

4.21 Problem 2 plan. 101
4.22 NAO pictures for different size boxes. 102
4.23 Problem 3 environment diagram. 103
4.24 Problem 3 plan. 104

5.1 Waterfall development process. 108
5.2 Project Gantt Chart. Dates are in MM/DD/YYYY format. 111

List of Tables

3.1 Use case 001. 40
3.2 Use case 002. 40
3.3 Use case 003. 41
3.4 Use case 004. 41
3.5 Use case 005. 42
3.6 Use case 006. 42
3.7 Use case 007. 43
3.8 Use case 008. 44
3.9 Use case 009. 44
3.10 Use case 010. 45
3.11 Use case 011. 46
3.12 Use case 012. 47
3.13 Functional requirement 001. 50
3.14 Functional requirement 002. 50
3.15 Functional requirement 003. 50
3.16 Functional requirement 004. 51
3.17 Functional requirement 005. 51
3.18 Functional requirement 006. 51
3.19 Functional requirement 007. 52
3.20 Functional requirement 008. 52
3.21 Functional requirement 009. 52
3.22 Functional requirement 010. 53
3.23 Functional requirement 011. 53

3.24 Functional requirement 012. 54
3.25 Functional requirement 013. 54
3.26 Functional requirement 014. 54
3.27 Functional requirement 015. 55
3.28 Functional requirement 016. 55
3.29 Functional requirement 017. 55
3.30 Functional requirement 018. 56
3.31 Functional requirement 019. 56
3.32 Functional requirement 020. 56
3.33 Non-functional requirement 001. 57
3.34 Non-functional requirement 002. 57
3.35 Non-functional requirement 003. 58
3.36 Non-functional requirement 004. 58
3.37 Non-functional requirement 005. 58
3.38 Objects in the domain. 60
3.39 Predicates in the domain. 60
3.40 Actions in the domain. 61

5.1 Date and task definition of the project development. 110
5.2 Estimated staff direct costs. 112
5.3 Estimated equipment direct cost for a use of 6 months. 113

Chapter 1

Introduction

Warehouse and industrial scenarios have been one of the first environments where tasks were automatized by using robots. With the industrialization, programmed and teleoperated machines were introduced in factories to improve the production process. Robots were machines programmed for a specific task and would always behave the same way. They were experts in their tasks and soon replaced human workforce. Teleoperated robots have been used, later on, for dangerous tasks that could be risky for humans, like mine detection [1], or for tasks that a robot could do with more precision, such as surgery [2]. In these situations, the robot is not aware of what is doing and cannot make decisions, it is just a puppet controlled remotely by the human.

With autonomous robots, warehouse environments were found again to be suitable scenarios for autonomous navigation as well as organization and decision-based problems. They pose challenges in sensing, control and deliberation tasks. However, the critical part consist on designing the control system or control architecture. This must determine the most suitable execution of the actions. In addition, the process to choose these actions could be a difficult task depending on the available information about the environment or the available time to make decisions. There are several types of control systems that change the way the decisions are made. From reactive architectures, which are based on acting depending on the inputs, to deliberative architectures, which are based on long-term reason-

ing techniques. Different architectures will result in different ways of controlling the robot in a system.

To overcome the challenges posed by warehouse environments when building control systems, Artificial Intelligence (AI) techniques have shown to be a suitable approach. For instance, the Kiva warehouse-management system [3] was successfully implemented with path planning as a homogeneous multi-agent system.

To control a robot it is necessary to generate a sequence of actions. This can be acomplished with Automated Planning (AP). This is the branch of Artificial Intelligence that studies the generation of an ordered set of actions (plan) that allows a system to transit from a given initial state to a state where a set of goals has been achieved. AP has been successfully used to solve real world problems such as planning Mars exploration missions [4] and controlling underwater vehicles [5].

In addition, if information from the environment is obtained through images then Computer Vision is used. This is a technique used in Artificial Intelligence that allows the agent to process and extract useful information from images or video of the environment. After processing this information, the agent uses it in its decision process or for other purposes such as face detection [6].

Finally, the availability, nowadays, of low-cost robots with capabilities to implement AI and complex techniques has enabled the creation of automated task systems not only in scientific or business research but also in academia. These robots are often available for educational purposes which encourages students to tackle by themselves problems that, otherwise, would be out of their hands.

In this dissertation, I propose a multi-agent system to solve planning tasks with heterogeneous agents. Two different robots will be used to cooperate and solve warehouse organizational tasks in a small scale.

The system I propose is a hybrid control system that combines reactive and deliberative control. This architecture incorporates the previously explained AI techniques, Automated Planning and Computer Vision. Consequently, the architecture should interleave Planning and Execution as well as an environment model used to generate the plans. Using Computer Vision, information about the

environment can be detected and use it to solve the tasks.

1.1 Problem description

The problem for this dissertation proposes a warehouse with a set of boxes that needs to be organized. The boxes must be placed in specific locations in the warehouse. Each box is tagged with information about its destination. When the boxes arrive to the warehouse, this information is unknown and the tag needs to be read to know the destination.

This problem is inspired in the *Sokoban* game[1]. In this Japanese puzzle the main task for the player is to move a set of stones in a board to their goal locations. The player can move in four directions (North, South, East and West) to empty cells and can push a stone but never pull it. The game finishes when all the stones are in their storage location. In Figure 1.1 an example of the Sokoban game is presented. The purple walls delimit the board and are obstacles and the blue cells inside are positions to which the player can move. The stones destination cells are marked with a diamond symbol and stones already in the destination cell are purple. The player is at the center-bottom of the board.

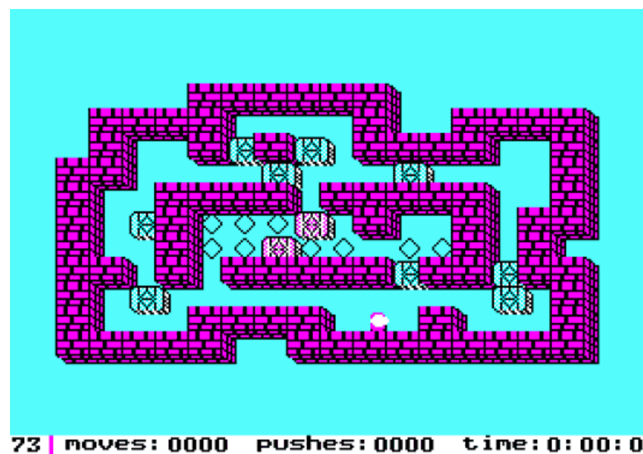


Figure 1.1: Example of the Sokoban game. (Source)

[1] `http://www.mobygames.com/game/soko-ban`, last visit 18 May 2016

The problem environment is designed based on this game and depicted in Figure 1.2. The warehouse is modelled as the board composed by a set of cells or locations that can be traversed in the four directions. The boxes are the stones in the original game and can only be pushed as well. The boxes storage locations are the stones destination cells. For the proposed problem, each box will have the same colour as its storage location. This is the information needed to know where to place the box.

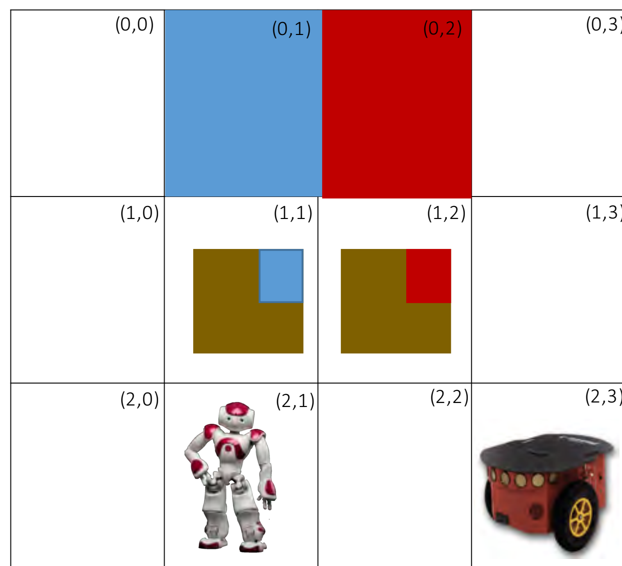


Figure 1.2: Problem environment.

The solution I propose in this dissertation to solve the problem is a multi-agent system composed of two types of robot: a humanoid robot and a mobile robot. The system uses Computer Vision to extract from the environment information about the storage locations. It also uses Automated Planning to generate plans for the robots to execute and solve the problem. The humanoid robot is able to see the environment but it is not efficient and precise when interacting with objects or walking from one position to another. Meanwhile, the mobile robot, which has two wheels, will reinforce this lack of skill of the humanoid robot. It can move faster through the environment but cannot see. This leads to a necessary

collaboration between the robots to successfully complete the warehouse organization task. Therefore, a hybrid control architecture using Automated Planning and Computer Vision will control the robots and solve the proposed task.

1.2 Motivation

Current technology and AI has successfully shown that homogeneous multi-agent systems can navigate autonomously and solve tasks, such as the Kiva warehouse-management system mentioned before. In heterogeneous multi-agent systems, the robots are different and have different capabilities and characteristics. To take advantage of them, they can collaborate to solve the tasks. This allows to propose more complex tasks. Motivated by this, this work will built a heterogeneous multi-agent system and experiment in real environments that simulate small warehouse environments.

The problem proposed in this work faces the challenges related to organizational tasks in warehouse environments. These are a wide variety of challenges from autonomous navigation to deliberation and sensing. This makes it a very complete work where several Computer Science techniques are involved making it very interesting.

Finally, the final result is one of the main motivations. Building an autonomous system where the robots coordinate correctly to solve tasks without human intervention is very promising. This work is in small-scale what bigger scientific researches with current technology have built but is ideal to understand the techniques, challenges and solutions that apply to this type of systems and an excellent closure to my degree in Computer Science.

1.3 Objectives

As previously stated, the main objective of this work consist on building a hybrid control system able of controlling a heterogeneous multi-agent system to relocate

some boxes in a warehouse. To accomplish so, several sub-goals need to be reached in order to achieve the main goal. These sub-goals are:

- Carry out an initial study of the technologies that will be used to become familiar with them and learn how to use them.
- Design a model to represent the problem environment based on the Sokoban and implement it.
- Develop the NAO control system that must enable the robot to move and turn.
- Develop a module that uses Computer Vision techniques to identify the colours of the boxes and integrate it with the NAO functionality.
- Coordinate the NAO and P3DX robots to solve tasks in the domain by using a hybrid control system.
- Evaluate the correctness of the system with experiments in a real-world environment trying to simulate a small warehouse.
- Generate the corresponding documentation.

1.4 Document structure

This document has been organized in 6 chapters and 3 appendices. Here is provided a general overview of each of them.

- Chapter 1 presents a detailed description of the problem, the motivation and the objectives. Finally, it is presented a description of the document's structure.
- Chapter 2 includes a study of the theoretical base used for this dissertation.

- Chapter 3 provides a description of both the analysis and the design of this work. In the first part of the chapter it is included the functionality description, hardware and software restrictions, use cases specification and requirements specification. In the second part is described the design and implementation of the system.
- Chapter 4 includes the empirical evaluation: unitary and system experiments.
- Chapter 5 describes the project management that has been used with the planification and the budgeting.
- Chapter 6 presents the conclusions drawn from this dissertation after its completion as well as proposing future work.
- Finally, the appendices A, B and C include the installation instructions, a user guide and the PDDL source code created for the dissertation, respectively.

Chapter 2

State of the art

This chapter provides a general overview of the theoretical background in which this work is included. First, a detailed definition of Software Agents and its types is presented. It includes a section in robotics since robots are the agents used in the system built for this dissertation. Second, the different types of control architectures are described. Then, Automated Planning is introduced with its formal definition and its components definition. Finally, Computer Vision techniques proposed for this dissertation are explained.

2.1 Software Agents

As it usually happens with abstract concepts, a unique definition for software agent has not been agreed yet. One of the reasons for this is that the word itself represents a heterogeneous body of development and research [7]. However, in Computer Science, we agree that software agents are computer programs that carry out tasks on behalf of their user. This, usually, implies some degree of authority and autonomy to decide which action to perform [8].

To be more precise, software agent systems have evolved from Multi-Agent Systems (MAS), which are closely related to Distributed Artificial Intelligence (DAI). This area solves problems of AI, specially if they require large data sets, by distributing parts of the problem to several processing agents. The main goal

is to exploit large scale computation and parallel computation to be able to solve complex problems. MAS inherited the main goals and benefits of this area and focused on how agents could coordinate their activities and knowledge. From these two fields, DAI and MAS, the concept of software agent was first coined by Carl Hewitt as [9]: “A self-contained, interactive and concurrently-executing object, possessing internal state and communication capability.”

According to Wooldridge in his book “An introduction to multi-agent systems” [10], the agent is a system situated in an environment and reacting to it as it is depicted in Figure 2.1 from his book. This definition differentiates the agent from ordinary computer programs because the latter will not have an effect on the environment with its output. Meanwhile, an agent has a continuous interaction with the environment.

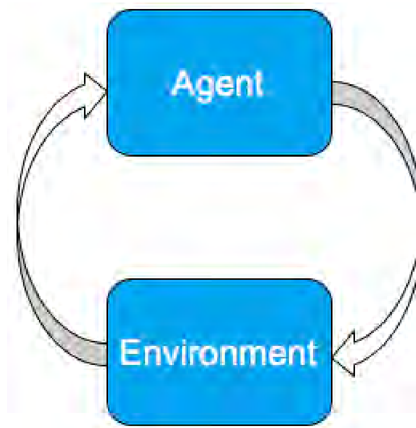


Figure 2.1: Interaction between agent and its environment.

2.1.1 Types of agents

There are several types of software agents according to their capabilities. A basic idea is provided for each of the types but agents can fall under several types depending on their capabilities:

- Mobile agents: these agents have the capability of transporting their state across platforms, processes or networks, deciding when to do so or having a fixed path. This way, they can achieve better performance by migrating to different environments when needed for reasons, for instance, of efficiency. When an agent does not have this capability it can be considered static. When dealing with physical agents, such as robots, mobile refers to their ability to navigate throughout the environment.

- Autonomous agent: these agents have the capability of performing on behalf of the owner without its intervention. These agents, usually, use sensors and actuators that provide data from the environment to decide which action is the appropriate to achieve their goal.

- Intelligent agent: these agents exhibit an intelligent behaviour usually through AI techniques that can be very simple or very complex. This capability involves in a high degree the autonomous capability and thus agents are often called autonomous intelligent agents [11].

- Multi-agent systems: these agents work in a cooperative system where the goal cannot be achieved by one of them alone and they have the ability to communicate with each other.

2.1.1.1 Multi-agent systems

As defined previously, the type of problems that can be addressed with Multi-Agent Systems (MAS) are those where cooperation is needed and are hard or impossible to be solved by individual agents. For that, the study of MAS focuses on developing the best way to achieve the cooperation between agents.

A MAS involves an environment, objects, agents, the actions that can be performed with the changes that result and the relationships between all entities [12]. Agents can be software agents, such as robots, but also humans. They are classified in passive, when the agent has no goal, or active, when the agent has a goal in the system [13]. Regarding the environments, they can be modelled with a wide range of characteristics. For instance, its determinism, to characterize the actions as deterministic or not in the environment or its accessibility, to determine how much visibility (information from the environment) can be obtained.

2.1.2 Robots

The concept of robot was first introduced by Karel Capek [14] in 1920 but the concept that the term robot represents was present for long before. Since ancient mythologies, the idea of creating an agent that could work on behalf of us has been exploited [15]. With remote-controlled systems, the idea seemed to be closer and in 1928 we had one of the first humanoid robots: Eric by H. W. Richardson[1]. Since then, the robotic field has seen enormous advances towards new forms of robots in different industrial and general-purpose areas. Some of the most modern robots that have had great impact are:

- The Lego Mindstorms[2] was launched in 1998 as a customizable robotic kit set aimed towards children. It was created by LEGO and the Massachusetts Institute of Technology (MIT) and introduce the robotics concepts in other fields, such as the educational field. Figure 2.2 shows different robots that have been built and customize with LEGO Mindstorm NXT

[1] `http://www.reffell.org.uk/people/ericrobot.php`, last visit 6th June 2016

[2] `http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com`, last visit 17th June 2016

Figure 2.2: Robot built with LEGO NTX Mindstorm kit. (Source)

- In 2000, Honda released ASIMO[3], a more modern, small and faster robot than the previous ones it had released. The ASIMO is a humanoid robot able to interact with humans by recognizing postures, gestures and faces. This work led to further research in walking-assist gear. Figure 2.3 shows the ASIMO robot.

Figure 2.3: The ASIMO robot. (Source)

[3] `http://asimo.honda.com/`, last visit 17th June 2016

- Aldebaran Robotics started developing Romeo[4] in 2009 and is nowadays a 140 cm tall humanoid robot. It was designed for research in assisting people losing their autonomy and elderly people. Due to its size, it can open doors, climb stairs and reach objects on a table. Figure 2.4 shows the Romeo robot.

Figure 2.4: The Romeo robot. (Source)

- The Atlas robot[5] is a humanoid robot aimed towards rescue and search tasks. It was financed by the United States Defense Advanced Research Projects Agency (DARPA) and developed by Boston Dynamics. The robot is robust and strong and prepared to rough terrains and tasks such as climbing. Figure 2.5 shows the Atlas robot.

[4] `http://projetromeo.com/en`, last visit 17 June 2016.
[5] `http://www.bostondynamics.com/robot_Atlas.html`, last visit 17 June 2016.

Figure 2.5: The Atlas robot in three different positions. (Source)

- The Pepper robot[6] was launched in 2014 by Aldebaran Robotics as a robot with the ability to perceive emotions. Based on the user's voice, expressions, movements and words it will interpret your emotion and offer content accordingly. It is sold as a companion and used to welcome customers in stores in Japan. Figure 2.6 shows the Pepper robot.

Figure 2.6: The Pepper robot. (Source)

[6] https://www.ald.softbankrobotics.com/en/cool-robots/pepper, last visit 17 June 2016.

The robot is, therefore, the most common physical agent and is present, nowadays, in a wide variety of forms, not limited only to humanoid shape. Among the different robots that have been created for research and academic purposes, we can find the Pioneer 3 DX robot and the NAO robot, both of which are used in this thesis.

The P3DX robot (Figure 2.7) is a mobile robot fully programmable distributed by the company ActivMedia. The motion controller of this robot has three physical components: motors and wheels to move, ultrasonic sonar sensors, front and rear, to analyse the environment and a CPU to automatically measure the velocity and provide control and state information. As mentioned before, it does not have any kind of vision devices. All of this, with other more specifications[7], makes the P3DX an all-purpose robot for applications such as monitoring, autonomous navigation, multi-robots' cooperation, etc. Figure 2.8 shows the P3DX dimensions.

Figure 2.7: Pioneer 3 DX robot. (Source)

[7] http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx, last visit 18 May 2016

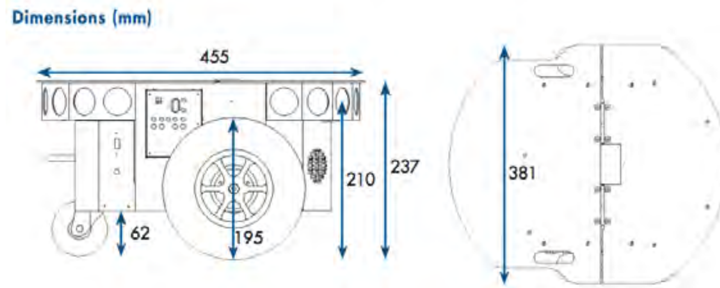


Figure 2.8: Pioneer 3 DX robot dimensions. (Source)

The NAO robot was developed by Aldebaran Robotics as a humanoid robot fully programmable as well. An Academic Edition was released in 2008 and is widely used in many institutions and a lot of projects in the educational and medical area with children have been carried out with this robot [16]. The version used in this work is the NAO H25 with 25 degrees of freedom, that is, the number of joints that it has. It includes an inertial measurement system with two components: accelerometer and gyrometer. The NAO also counts with microphone, speakers and two cameras, as well as, sonar, infrared, tactile sensors and pressure sensors. The operating system of the robot is Linux-based with an API in C, called NAOqi. Figure 2.9 shows the NAO robot and its parts.

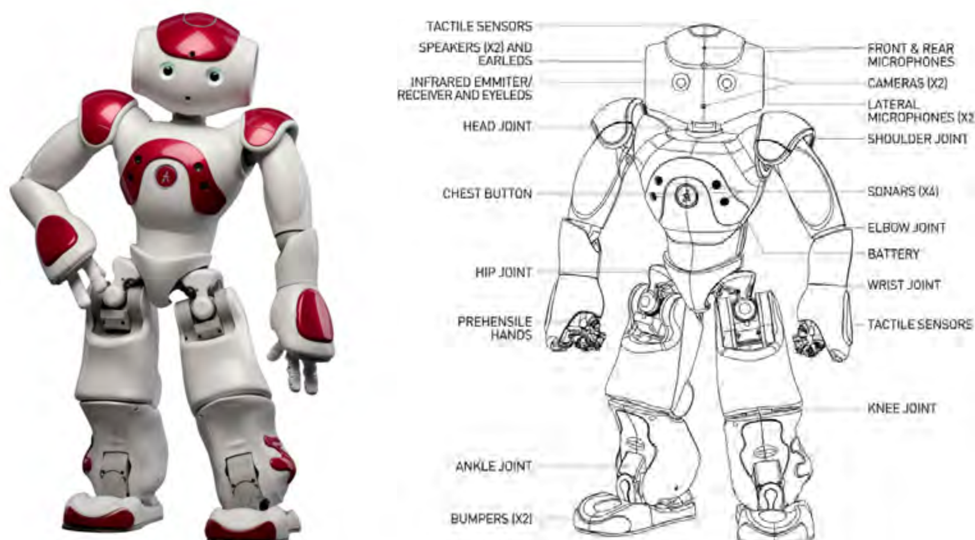


Figure 2.9: NAO robot.(Source)

2.2 Control systems for autonomous agents

The control system is the component in charge of controlling the agent. If the agent is a robot with a basic degree of autonomy, then the control system needs to enable the robot to: sense its environment and react to it, perform actions in a secure way, handle malfunctions (robustness and reliability) and exhibit an intelligent behaviour solving the tasks, among other characteristics. The control system must ensure that the robot will perform the task taking into account these requirements and constraints.

There exist three main control paradigms [17] to control an agent that will be discussed in the following sections: deliberative, reaction and hybrid. The difference between them is on the process by which they decide the actions to perform in the environment.

2.2.1 Deliberative Control Systems

A deliberative control system was first used in the robot Shakey [8] developed by the Stanford Research Institute in 1966. This system uses a global world model to plan the optimal path creating a plan that the robot will, then, execute. The architecture includes three modules. Figure 2.10 depicts the interaction of them.

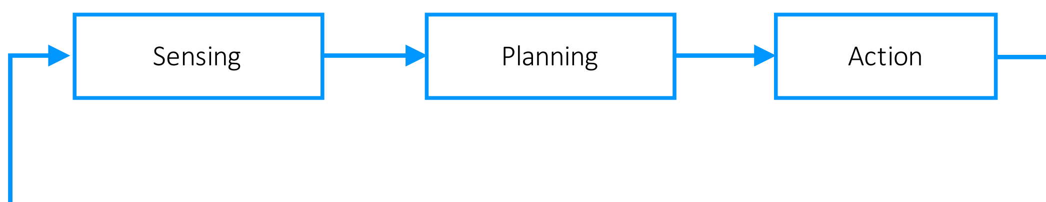


Figure 2.10: Basic structure of a deliberative architecture.

[8] http://www.ai.sri.com/shakey/, last visit 18 May 2016

- Sensing module: first, using information form its sensors, the robot processes the information from the static environment to create a world model.
- Planning module: then, this module searches for the solution of the problem using Automated Planning and the model. It generates a plan for the robot.
- Action module: finally, the robot will execute the actions in the plan. However, after each action it will stop to update the information and re-plan if necessary.

In the deliberative control system, time and resources are spent in the planning process for decision-making. This requires efficient processing capabilities and enough memory, posing a limitation to the system. Besides that, the planning process depends on the world model. Therefore, a plan without having received first the information from the environment could be a risk to the robot's integrity in a dynamic environment.

These two aspects are an important limitation in systems where the environment is very dynamic since the planner cannot keep up with the rate of changes in the environment. If the planner takes too much time generating a plan and the environment changes meanwhile, the plan could not be valid anymore. Therefore, they are also inconveniences for systems that need to react quickly. However, this architecture is useful for complex problems where long term reasoning is necessary to reach the goals.

2.2.2 Reactive Control Systems

To tackle the drawbacks in deliberative architecture for dynamic environments, Rodney Brooks proposed in 1986 the Subsumption architecture [18], a reactive control architecture. In this system, all possible behaviours are defined as strict rules that are activated depending on the sensory input and have an output action for the robot to carry out. The main loop of action is shown in Figure 2.11, in this case composed of two modules.

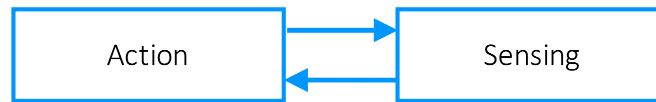


Figure 2.11: Basic structure of a reactive architecture.

- Sensing module: first, using information form its sensors, the robot processes the information to determine which predefine behaviour is activated.
- Action module: the robot executes the action of the behaviour that was activated.

As it can be seen, the robot uses the local model of environment to take actions, without a time-consuming planning process. This allows the control system to quickly provide a response to the changes in the environment. In addition, the response will always be the same since behaviours and responses are already predefined.

One of the disadvantages in this architecture is that, due to the lack of the planning module, is not appropriate for complex problems where long term reasoning is necessary. Another one is that it cannot correctly react to unexpected behaviours since they have not been predefined. However, for problems where this does not happen, this type of control architecture makes the robot experts in their tasks and enables it to react rapidly in very dynamic environments.

2.2.3 Hybrid Control Systems

Deliberative paradigms are mechanisms that allow the robot to make decisions while reactive have predefined behaviours. If both are combined it is obtained a hybrid control system with both characteristics. In this type of systems, while being deliberative, it still has a constant direct communication with the environment [19]. This system is composed of four modules depicted in Figure 2.12.

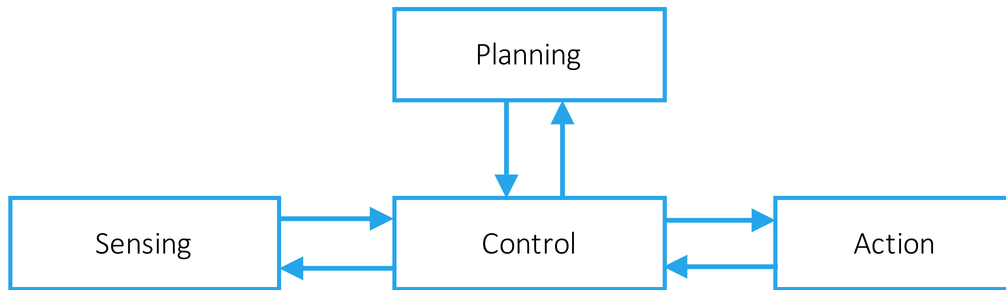


Figure 2.12: Basic structure of a hybrid architecture.

- Sensing module: gathers information from its surroundings and sends it to the Control module.

- Planning module: as part of the deliberative approach, there is a planner or any other deliberative system that will provide the Control module with a plan of actions.

- Control module: decides which action to perform: a reactive one from the sensors or a deliberative one. Then sends it to the Execution module.

- Execution module: receives the actions and ensures the robot carries them out.

It can be seen that this approach still requires processing time but is able to face both scenarios: very dynamic environments and complex problems.

This paradigm is the one used in this project for the control system that has been built. As it will be described in the chapter 3, the control system uses a planner to determine the sequence of action but relies on sensor information to do so, which will be processed through computer vision. Therefore, from the sensors, it does not only obtain information to create a world map but also information to decide which action performs. The Computer Vision will be the reactive part while the Automated Planning will be the deliberative part.

2.2.4 PELEA

The system architecture that will be used in this dissertation is called PELEA (Planning, Execution and Learning Architecture), a system developed by three Spanish universities, one of them being the Carlos III University [20]. PELEA is a generic architecture that uses Automated Planning and Machine Learning to solve automated tasks. The architecture is distributed in different components that can be easily substituted depending on the task to be solved.

The architecture has 8 modules[9] which are depicted in Figure 2.13 together with the messages and variables passed in the system. Next, the modules are briefly described.

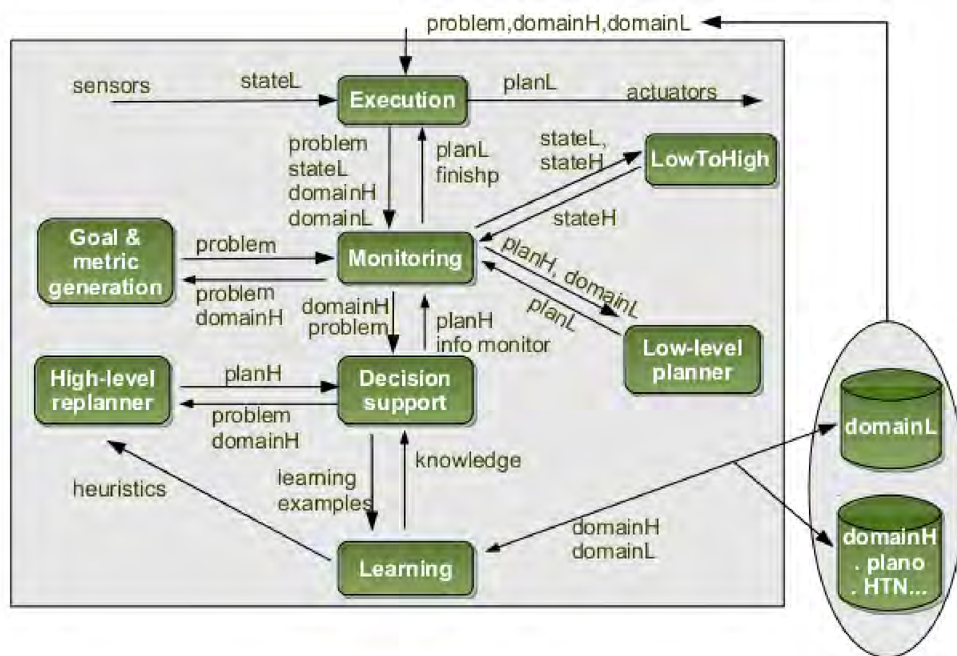


Figure 2.13: PELEA architecture with variables. (Source)

[9] http://www.plg.inf.uc3m.es/pelea/modules.php, last visit 23 May 2016

- *Monitoring*. This module ensures the synchronization between the planning process (high level) and the execution process (low level). For that, it will manage the communication between the modules and will ensure that the low-level predicates have the expected value. From the *DecisionSupport* module it will receive which predicates to monitor along with their range values. Actions to be executed will be sent to the *Execution* module from the plan and will receive the new state after the execution of each action.

- *Execution*. This module is used for communicating with the real environment, usually, with the robot. It receives the high-level actions or low-level actions and sends them to the robot. It also receives information from the robot sensors to send them to *Monitoring*. The system should have one *Execution* module per robot.

- *Decision Support*. This module will decide which predicates to monitor and their valid values. Also, it will call the $High-level\ replanner$ when the current state does not match to the expected one in high-level to determine if re-planning is needed and do so.

- $High-level\ replanner$. This module encapsulates a domain-independent automatic planner. It receives the problem and the domain in XPDDL and translates them to the language that the planner uses. Then, receives the plan created by the planner and translates it to XPDDL to send it to the *Decision Support* module.

- $Low-level\ planner$. In cases where high-level actions correspond to several low-level actions, this domain-independent planner will do the planning for the transformation. In other cases, the transformation can be done directly and this planner is not needed.

- *LowToHigh*. Gathers information in low-level format from the sensors and converts it to high-level to build the high-level state.

- *Learning.* This module generates different type of knowledge from the results of the plans and past executions. This gives support to the high-level replanner and refines the domainH, for instance.

- *Goal and metric generation.* This is an optional module that manages the goals and metrics. For example, it calculates subset of goals or establishes the metric criteria to optimize the plans.

Information is passed between the modules in the form of messages with variables. There exist variables for the high-level part of the architecture, such as $domainH$, $stateH$ and $planH$, where information is expressed in high-level instructions. There are also variables for the low-level part such as $domainL$, $stateL$ and $planL$, with the information expressed in low-level instructions. Finally, there are also monitoring variables with their associated valid ranges and variables to store problem information like the goals, the heuristics and metrics used in the planner.

An execution cycle with PELEA starts with the Execution module. This modules receives high-level information about the problem, initial state and goals as well as $domainL$ and $domainH$ with the possible actions that can be executed. This information is sent to the $Monitoring$ module. If necessary, this module will call $LowToHigh$ to get a high-level state $stateH$ from the low-level $stateL$.

The high-level state $stateL$ and $domainL$ are sent to $DecisionSupport$. This sends them to the High-level planner which creates $planH$ and returns it to Decision Support. This module, with a plan, calculates the variables that need to be monitored and sends everything to $Monitoring$.

$Monitoring$ sends the plan to the Low-level Planner module to get the $planL$ and send it to $Execution$. $Execution$ executes the first action and returns to $Monitoring$ the low-level state. $Monitoring$ checks if goals have been reached, otherwise, it checks if the current state is valid. If not, it is resent to *Decision Support* for replan or repair. Otherwise, next action is executed. This cycle is repeated until the goals are reached.

PELEA can be deployed using different configurations where there are only 3 modules (Monitoring, Decision Support and Execution) or 5 modules (Monitoring, Decision Support, Execution, LowToHigh and Low-level planner), depending on the problem. In this dissertation, the 3-module configuration has been used and how it has been specifically used will be described in section 3.3.

2.3 Automated Planning

Automated Planning (AP) is a field of Artificial Intelligence which generates sequence of actions, called plans, that transform the current state of the environment into another state where a set of goals are reached. Commonly, these plans are used for the execution on autonomous robots, intelligent agents or unmanned vehicles providing the agent with decision-making capabilities.

There are different planning paradigms depending on both the observability level of the environment and the structure of the action model (deterministic or non-deterministic). Classical Planning is considered the most simple paradigm which assumes that the environment is fully observable and the action execution is deterministic.

A Classical planning task can be modelled with three elements: a conceptual model to describe the environment and the problems; a description language, such as PDDL, to encode the conceptual model in a generic way; and an algorithm to solve the task and generate a solution.

2.3.1 Conceptual model

The conceptual model for a Classical Planning task can be defined as a tuple $\Pi = (F, A, I, G)$, where:

- F is a finite set of grounded literals (also known as facts or atoms).
- A is a finite set of grounded actions derived from the action schemes of the domain.

- $I \subseteq F$ is a finite set of grounded predicates that are true in the initial state.
- $G \subseteq F$ is a finite set of goals.

Any state s is a subset of facts that are true at a given time step. Each action $a_i \in A$ can be defined as a tuple $a_i = (Pre, Add, Del)$, where $Pre(a_i) \subseteq F$ are the preconditions of the action, $Add(a_i) \subseteq F$ are its add effects, and $Del(a_i) \subseteq F$ are the delete effects. $Eff(a_i) = Add(a_i) \cup Del(a_i)$ are the effects of the action. Actions can also have a cost, $c(a_i)$ (the default cost is one). An action a is applicable in s_i, if $Pre(a) \subseteq s_i$. Then, the result of applying an action a in state s_i generates a new state that can be defined as: $s_{i+1} = (s_i \setminus Del(a)) \cup Add(a)$. A plan π for a planning task Π is an ordered set of actions (commonly, a sequence) $\pi = (a_1, \ldots, a_n), \forall a_i \in A$, that transforms the initial state I into a state s_n where $G \subseteq s_n$. This plan π can be executed if the preconditions of each action are satisfied in the state in which it is applied; i.e. $\forall a_i \in \pi, Pre(a_i) \subseteq s_{i-1}$ such that state s_i results from executing the action a_i in the state s_{i-1}. s_0 is the initial state I.

2.3.2 Modelling language

The modelling language describes the semantics of the Classical Planning task. One of the first languages was STRIPS [21] which represents the world model with first-order predicate calculus. Then, in 1987 Pednault proposed the Action Description language (ADL) to adapt STRIPS to more realistic problems. However, the standard came with the Planning Domain Definition Language (PDDL), which is a superset of ADL and STRIPS.

In PDDL [22], planning tasks are described in terms of objects of the world (robots, locations, boxes, etc), predicates which describe static or dynamic features of these objects or relations among them (e.g. boxes are at locations), actions that manipulate those relations (a robot can move from one location to another, a box can be pushed by a robot), an initial state that describes the initial situation before plan execution, and a goal definition which describes the objectives that

must be reached by the solution plan. This information is provided in two input files: domain and problem. The domain file contains a definition of a set of generalized actions A, a set of ungrounded predicates F and a set of object types. The problem file defines a set of objects, an initial state (I), and a set of goals (G).

2.3.3 Algorithms

The third element of a planning task is the algorithm that will generate the plan to solve a proposed task. Some of the algorithms that Classical Planning uses are Forward Chaining, Backward Chaining and State Space Search.

Forward Chaining and Backward Chaining are both algorithms that follow a reasoning based in propositional logic. They consist on the application of the logic concept *Modus Ponens* on a set of rules. Forward Chaining reasoning starts from the problem to get to the goals while Backward Chaining reasoning starts with the list of goals and reasons backwards until it gets to the problem's initial state.

State space search models the problem as a set of states in a state space. The goals is a set of states or a state that need to be reached. There exist operators to transit from one state to another. The algorithm checks for each state the operators that can be applied to transition from state to state until it reaches the goal. This way it builds a search graph.

State search can be enhanced using heuristic. These are functions that ranks the possible operators that can be applied for a given state. Then, the algorithm will choose the best option. Heuristic search space for planners was lost after STRIPS and it was not until 1996 that was again contemplated, with the UNPOP planner [23] and Bonet and Geffner with HSP planner [24].

With that in mind, the FastForward (FF) [25] planner uses a very fast search algorithm that combines forward chaining local searches with the heuristic developed for the HSP planner. FF has been used and considered the fastest until Lama [26], a planner by Richter, won the International Planning Competition

in 2011. This algorithm is based on heuristic forward search built on the Fast Downward planning system [27]. The core feature of this algorithm is the use of pseudo-heuristics to optimize the search.

2.4 Computer Vision

Computer Vision is a field that studies techniques to extract information from images, video or any other visual input to understand them [28]. These Computer Vision techniques allow systems to analyse the environment and recognise objects, faces, routes or patterns in it. In a general way, the phases that a Computer Vision algorithm follows are:

1. Image acquisition: receive the image either through sensors and cameras or from an external supplier.
2. Pre-processing: process the image to change the colour space (RGB, HSV, etc.) or re-sample it to adjust its quality and size. In this stage, modify the image so that the relevant features can be easily extracted later on.
3. Feature extraction and segmentation: in this stage methods are applied to determine locally where the relevant features are. These can be lines, edges or points such as blobs.
4. High-level processing: at this point, the feature is isolated and it can be processed to achieve the goal, e. g.: classification, comparison or verification.

In this dissertation, Computer Vision is used to extract information of an object in the image. There are several techniques to extract this information from an image. The most common ones are explained in detail in the following sections.

2.4.1 Colour spaces

The colour space specifies the organization of colours in the image with a notation to represent them. There is a wide variety of colour spaces [29] with colour models

to represent colours in an image or video. Two of the most used ones are:

- RGB (Red, Green and Blue) colour model: additive colour model where red, green and blue light are added together to produce other colours.
- HSV (Hue, Saturation and Value) colour model: it was derived from the RGB space but represents the colours in terms of their three characteristics, hue, saturation and value, which is more similar to the way humans understand colours and therefore, more intuitive.

It is possible to perform a conversion from one space to another without any difficulty. Most of image software nowadays has space colour conversion tools available.

2.4.2 Histograms

Colour histograms are a graphical representation of the colours of each one of the pixels in an image. They give the distribution of the colour over a range in different colour spaces, usually RGB or HSV. Since these spaces are three-dimensional, there are three histograms, one for each dimension. That is R, G and B in RGB space. Each of them represents the quantity of that colour that each pixel has. Figure 2.14 shows an example of histograms representation. There is one per color channel. In this case the colour space is RGB so there is one for Red, another for Blue and another for Green.

As an example of this technique, histograms are used to detect moving objects in an image [30]. For that, the histogram representing the colour distribution in the background image and the current frame histogram are compared to obtain the similarity. The background histogram is calculated from an image with no objects, just the background. The current frame histogram is obtained from the image in which the object wants to be detected. Two experiments are described in the paper, one with colour histograms and another one with edge histograms. The results show better performance for the latter.

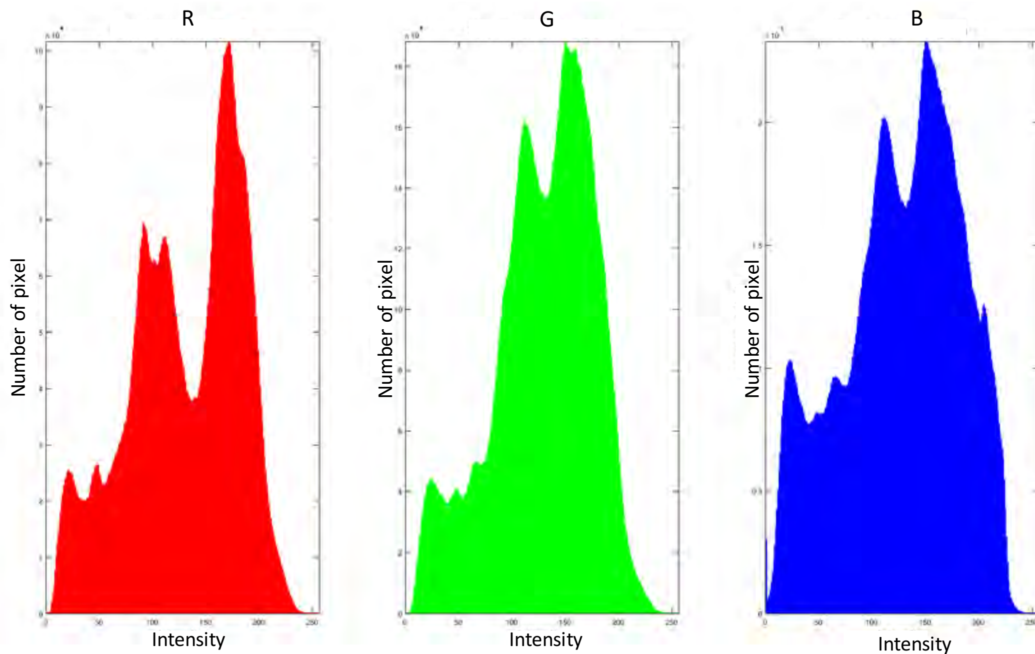


Figure 2.14: R, G and B Histograms. The x-axis represents the intensity value for the colour and the y-axis the number of pixels with that intensity. (Source)

2.4.3 Blobs

Blobs are regions of an image where certain properties are constant and differ from other regions surrounding. It is based on the algorithm of graph theory of connected-component labelling [31].

Therefore, given an image or a sequence of images, with blob detection we can locate similar areas according to properties such as colour or brightness. Once the blobs are located, other relevant information can be extracted such as the position and the points that are part of them, as well as calculating the area. This possibility has been one of the main reasons to consider using blobs in this dissertation, since edge and corner detection, which are similar techniques, did not provide this information. Figure 2.15 shows an image before being processed (a) where red balls are going to be detected and an image after applying blob detection (b) where the red balls have been identified.

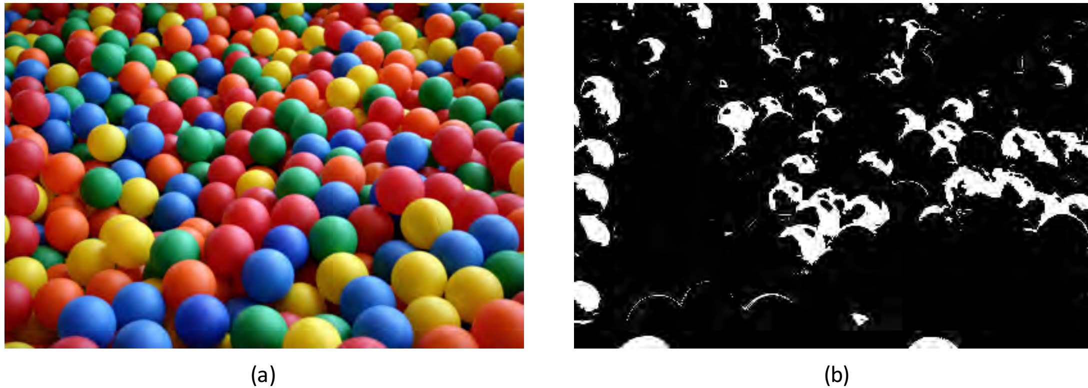


Figure 2.15: Original image (a) and image with red blolb detection (b).

This technique is used in [32] to identify human bodies. They use blobs to represent the global aspects of the shape of a human body forming blobs for parts with similar colours or close. For instance, a human would be composed of: a blob for the head, a blob for the neck and torso, a blob for each arm, a blob for both legs and a blob for each foot. Combined with scene analysis and statistical techniques, the results show successful human detection in real-time.

2.4.4 QR

A QR code (Quick Responsive code) is a matrix barcode that can store data in four types of encoding (numeric, alphanumeric, binary and kanji). It was first designed by Denso Wave [10] in 1994 for automotive purposes in Japan. However, nowadays it has been extended to a much broader market aimed at mobile-phone users.

Computer Vision can be used to detect objects in an image such as a QR. Once the QR is detected it can be read and extract from it relevant information. It is another way of obtaining information from the environment. This is used in this dissertation to store information about the boxes. The coordinator robot identifies the QR through Computer Vision and then reads it with a QR reader to get the information about the colour of the box.

[10] `http://www.globaldenso.com/en/`, last visit 18 May 2016

A QR code is decoded following a pattern [11] which is shown in Figure 2.16. The three distinctive big squares help orientate the QR code so that information can be found. Around them we have format information that includes the error correction and the mask pattern. The error correction level can be specified and indicates how much code can be restored. There are four levels: L, M, Q and H, being L the one where the least information can be restored (7%) and H where the most information can be restored (30%).The mask pattern indicates how data was mixed so that it can be decoded without leaving blank areas. Then, the message is placed following the encoding type and including alignment and error correction.

There is an open-source library very common, ZXing[12], that implements functions to process an image to detect QR codes as well as a QR reader (or decoder). It is available in several programming languages.

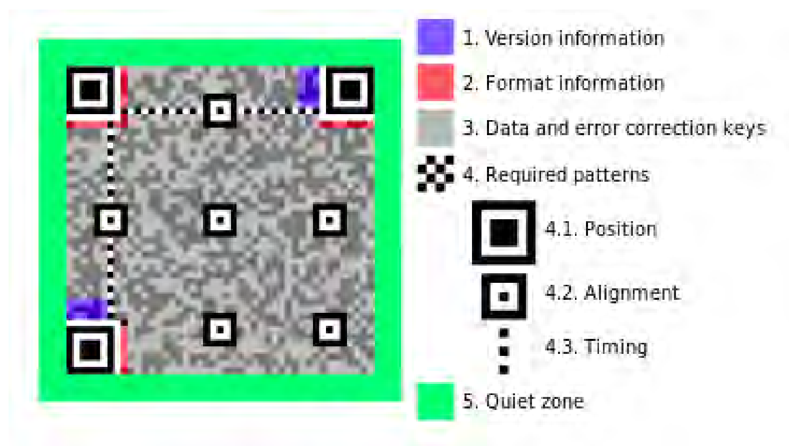


Figure 2.16: QR illustration. (Source)

[11] `http://www.qrcode.com/en/`. last visit 18 May 2016
[12] `https://github.com/zxing/zxing`, last visit 18 May 2016

Chapter 3

System description

This chapter describes the system developed to solve the problem proposed in the first chapter. A system analysis is described in detail with its corresponding use cases definition and requirements definition, as well as the restrictions that the system has. Finally, the system architecture and components developed will be described in details.

3.1 Introduction

This dissertation consists on building a multi-agent system that solves problems using Automated Planning and Computer Vision. The problems take place in a warehouse environment where several boxes need to be moved to their corresponding destination locations. Two different robots cooperate using their different capabilities to carry out the actions that solve the problem because the destination location of the boxes is unknown.

There is a coordinator agent that has the capability to extract information about the environment using camera devices. It is called coordinator because it retrieves information from the boxes to determine where they must go. There is also a cargo agent which is able to move the boxes. The coordinator agent is a humanoid robot while the cargo agent is a mobile robot. Therefore, both robots can execute two different types of actions: (1) common actions related to the

navigation process (move and turn); (2) specific actions related to the capabilities of each robot type. The coordinator robot is capable of identifying the colour so that the final destination of the box is known and the box can be pushed by the cargo robot.

The system is composed of the following elements:

- A laptop to execute the control system.
- The coordinator agent, in this case the NAO robot. There can be several of them.
- The cargo agent, in this case the P3DX robot. There can be several of them. This robot requires a PC to connect to the system.
- A router for the communications.

Figure 3.1 show a high-level diagram of the whole system.

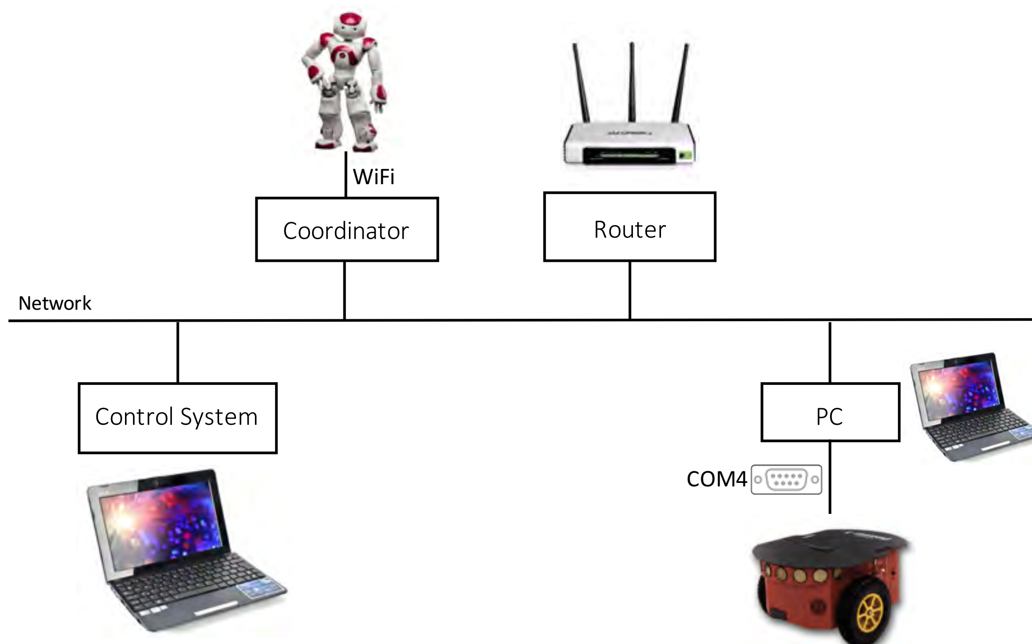


Figure 3.1: System high level diagram.

3.2 System analysis

In this section, it is presented an analysis from the point of view of functionality, restrictions and the operating environment. Then, the uses case and subsequent requirements will be described.

3.2.1 Functional characteristics description

The system developed for this dissertation must be able to solve multi-agent tasks in a warehouse domain based on the Sokoban domain. The functional characteristics describe aspects of the functionality that the system must include to achieve the whole system functionality. These characteristics are:

- Be able to identify the colours of the boxes through Computer Vision techniques.
- Be able to push the boxes around the board as precise as possible.
- Be able to move around the board as precise as possible.
- Be able to turn to change the agents' orientation as precise as possible.
- Be able to determine a plan to solve the problem using Automated Planning.
- Have a representation of the domain and problems for the planner (in high-level).
- Process the actions from the planner and adapt them for the robot to carry them out (translate form high-level to low-level).
- Connect all components of the system so that it works automatically.

3.2.2 System restrictions

In this section, the restrictions that have been taken into account when designing the system are described as well as the ones found during its development and experimentation.

3.2.2.1 Hardware restrictions

These restrictions refer to physical limitations imposed by the hardware used, in this case, the NAO, P3DX robots and the laptop.

- The model of NAO used corresponds to the H25 V3.2 that has 25 degrees of freedom (DOF). These are located in the following way: two in the head, five in each arm, one in each hand, one in the pelvis and five in each leg.
- The connection to the NAO robot is through Wi-Fi.
- Regarding the vision, the NAO robot has two video cameras in the head with a resolution of 1.22 Mp and 30 frames per second.
- The NAO robot has an inertial unit with a 3-axis gyrometers with angular speed of around 500º/s and a 3-axis accelerometer with an acceleration of around 2g.
- The laptop characteristics are: double core Intel Atom N550 1.5 GHz, 1GB RAM DDR3, 250GB hard disk and graphics Intel GMA 3150.
- The P3DX has to be physically connected through a COM4 port to a laptop so that is connected to the whole system.

3.2.2.2 Software restrictions

Software restrictions include limitations due to the software used in the system.

- The operating system requires Java SE 1.7 or above.
- For Computer Vision, version 2.2 of XZing library is needed.
- The JNAOqi library is the version 2.1.4.13.
- Metric-FF planner is version 2.1.
- PELEA is version 2.1.

3.2.3 Operating environment

In this section, the operating environment is described in terms of hardware and software.

3.2.3.1 Hardware environment

This environment includes hardware devices that are required for the system to work.

- NAO H25 V3.2 robot from Aldebaran Robotics.
- Pioneer 3 DX robot.
- Router.
- Laptop with software characteristics described in the following section.
- COM4 wire.

3.2.3.2 Software environment

This environment includes the operating system used and the libraries used. These are:

- The operating systems that have been used are: OS X EL Capitan 10.11.4, Ubuntu 12.04 LTS.
- The control architecture is PELEA with Metric-FF planner version 2.1.
- Java SE 1.7.
- Library and API versions: XZing 2.2, JNAOqi 2.1.4.13, rosjava.

The programs that have been used to develop and test are:

- Eclipse Mars.
- Webots 8.3.2 for NAO: this is a NAO simulator.

- Choregraph 2.1.4: graphic interface to control the NAO and execute behaviours.

Links for these can be found in the installation manual in the appendix A.

3.2.3.3 Problem environment

This environment refers to the physical environment where the system is going to be deployed, including the ground, light and other external agents.

- The ground should allow the robot to walk without difficulties, having no bumps that can make the robot fall.
- The environment lighting should be adequate for the robot's camera to take clear pictures. It cannot be dark or very bright since objects in the image will not be recognizable. It is, therefore, suggested an indoor space with artificial lighting.
- The boxes used should not be very heavy so that the robot can push them and must fit in a board location.
- The board cells need to be big enough to fit the robots.

3.2.4 Use case specification

In this section, the use cases that cover the objectives of the project are presented. In Figure 3.2 it is depicted the Use Case Diagram with the use cases and the actors that take part in each of them.

3.2.4.1 Actors description

The actors that participate in the use cases are the following:

- User: corresponds to the person that uses the system.
- NAO robot: corresponds to the physical NAO robot.
- P3DX robot: corresponds to the physical P3DX robot.

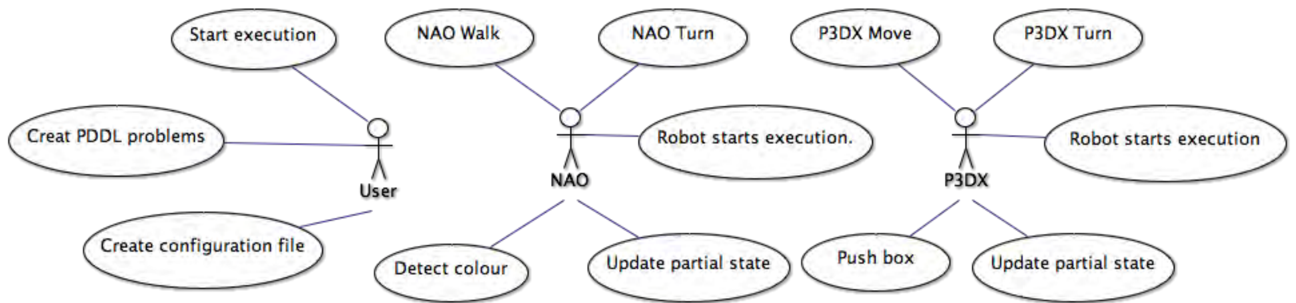


Figure 3.2: Use Case diagram.

3.2.4.2 Use case attributes description

For the textual description of the use cases several attributes have been selected. They are described in detail next:

- ID: Abbreviated univocal identification of the use case. It is formed by *UC* followed by a - and three digits. For example, UC-001.
- Name: Extended identification of the use case.
- Actors: Set of entities that interact in the use case. The use case represents a functionality demanded by the actors.
- Description: this is a basic description of the use case functionality.
- Preconditions and effects: this is a description of the conditions that need to be true before (preconditions) and after (effects) the functionality described by the use case is performed.
- Scenario: this is a basic description of the actions that will be executed in the use case.

3.2.4.3 Use case textual description

The following tables contain the textual description of each use case.

Use case	
ID	UC-001
Name	Model PDDL problem.
Actors	User.
Description	The user models the problem in PDDL.
Preconditions	None.
Effects	Problem is modelled.
Scenario	The user models the problem following the domain. The user generates a .pddl file for the problem.

Table 3.1: Use case 001.

Use case	
ID	UC-002
Name	Create configuration file.
Actors	User.
Description	The user creates the configuration file for PELEA.
Preconditions	The problem has been defined in PDDL for the system (Use Case in Table 3.1).
Effects	Configuration file is created.
Scenario	The user creates a .xml configuration file.

Table 3.2: Use case 002.

Use case	
ID	UC-003
Name	Start execution.
Actors	User.
Description	The user starts the robots and the control system to start the execution.
Preconditions	The problem and the configuration file have been defined.
Effects	System execution starts.
Scenario	The user starts every robot. The user turns on the router for the Wi-Fi connection. The user starts the control system execution (PELEA) executing a script.

Table 3.3: Use case 003.

Use case	
ID	UC-004
Name	Robot starts execution.
Actors	NAO, P3DX.
Description	The robot starts the execution.
Preconditions	The user has started the system (Use Case in Table 3.3).
Effects	Execution has started. The robot sends the domain and problem to Monitoring.
Scenario	The robot sends the domain and problem to Monitoring to start the execution.

Table 3.4: Use case 004.

Use case	
ID	UC-005
Name	NAO Walk.
Actors	NAO.
Description	The NAO robot walks a specified distance.
Preconditions	NAO robot has been started and it is connected to the Wi-Fi (Use Case in Table 3.1). NAO has received from control system the order.
Effects	The NAO robot has walked the specified distance.
Scenario	The NAO robot receives from the control system the *walk* order. NAO walks.

Table 3.5: Use case 005.

Use case	
ID	UC-006
Name	NAO Turn.
Actors	NAO.
Description	The NAO robot turns the specified degrees.
Preconditions	NAO robot has been started and it is connected to the Wi-Fi (Use Case in Table 3.1). NAO has received from control system the order.
Effects	The NAO robot has turned the specified degrees.
Scenario	The NAO robot receives from the control system the *turn* order. NAO turns.

Table 3.6: Use case 006.

Use case	
ID	UC-007
Name	Detect colour.
Actors	NAO.
Description	The NAO robot detects and reads a box QR.
Preconditions	NAO robot has been started and it is connected to the Wi-Fi (Use Case in Table 3.1). NAO has received from control system the order.
Effects	The NAO colour tries to detect the QR and read it. QR can be detected or not. In case it is not detected PELEA will replan.
Scenario	The NAO robot receives from the control system the $get - colour$ order. The NAO robot takes a picture. Applies image filters. Attempts to detect a QR. If detected, it reads the QR and obtains information (colour and box name) from it.

Table 3.7: Use case 007.

Use case	
ID	UC-008
Name	NAO Update partial state representation.
Actors	NAO.
Description	The robot updates its world model with the changes after executing an action.
Preconditions	The robot has executed an action, either *walk*, *turn* or $get - colour$.
Effects	The partial state representation is updated with the changes produced by the execution of the action.
Scenario	The changes are interpreted and translated to high-level to update the model in PDDL. The changes can be: change in position (*walk*), change in orientation (*turn*) and change in environment information ($get - colour$).

Table 3.8: Use case 008.

Use case	
ID	UC-009
Name	P3DX Move.
Actors	P3DX.
Description	The P3DX robot moves a specified distance.
Preconditions	P3DX robot has been started and it is connected to the system (Use Case in Table 3.1). P3DX has received from control system the order.
Effects	The P3DX robot has moved the specified distance.
Scenario	The P3DX robot receives from the control system the *move* order. P3DX moves.

Table 3.9: Use case 009.

Use case	
ID	UC-010
Name	P3DX Turn.
Actors	P3DX.
Description	The P3DX robot turns the specified degrees.
Preconditions	P3DX robot has been started and it is connected to the system (Use Case in Table 3.1). P3DX has received from control system the order.
Effects	The P3DX robot has turned the specified degrees.
Scenario	The P3DX robot receives from the control system the *turn* order. P3DX turns.

Table 3.10: Use case 010.

Use case	
ID	UC-011
Name	Push box.
Actors	P3DX.
Description	The P3DX robot pushes a box to the next cell.
Preconditions	P3DX robot has been started and it is connected to the Wi-Fi (Use Case in Table 3.1). P3DX has received from control system the order.
Effects	The P3DX has pushed a box to the next cell.
Scenario	The P3DX robot receives from the control system the *push* order. The P3DX moves to the cell with the box to push it. S¿It stops when the box is in the next cell. Then, it returns to its cell, the one where the box was.

Table 3.11: Use case 011.

Use case	
ID	UC-012
Name	P3DX Update partial state representation.
Actors	P3DX.
Description	The robot updates its world model with the changes after executing an action.
Preconditions	The robot has executed an action, either *move*, *turn* or *push*.
Effects	The partial state representation is updated with the changes produced by the execution of the action.
Scenario	The changes are interpreted and translated to high-level to update the model in PDDL. The changes can be: change in position (*move*), change in orientation (*turn*) and change in environment information (*push*).

Table 3.12: Use case 012.

3.2.5 Requirements specification

In this section, the requirements that the system must fulfill are described. For each requirement presented a detailed description in section 3.2.5.1. Then, the functional requirements are presented in section 3.2.5.2 and the non-functional in 3.2.5.3.

3.2.5.1 Requirements attributes description

For the textual description of the requirements, several attributes have been selected. The meaning of each attribute is described next:

- ID: Abbreviated univocal identification of the use case. It is formed by requirement's code followed by a - and three digits. The requirements are classified into functional and non-functional and their codes will be FR and NFR, respectively. For instance, FR-001.
- Name: Extended identification of the requirement.
- Description: this is a basic description of the use case functionality.
- Source: indicates from which source the requirement was identified. Usually, this corresponds to one or more use cases.
- Significance: determines the implementation degree of the requirement. It can take the following values:
  - Essential: the requirement must be implemented.
  - Desirable: it is preferable to implement the requirement, but is not mandatory.
  - Optional: the requirement can be implemented, but is not important nor mandatory.
- Priority: defines the requirement importance to determine the order in which it will be included in the design and implementation process. It can take the following values:

  - High: The requirement must be implemented in the initial development stages.
  - Medium: the requirement must be implemented once all high-priority requirements have been implemented.
  - Low: the requirement must be implemented in the final development stages. These requirements will not affect the correct system operation.

- Stability: defines the requirements stability during the software useful life. This implies whether the requirement can be modified or not during the software's life cycle. It can take the following values:
  - Stable: the requirement cannot be modified during the system's life cycle.
  - Unstable: the requirement can be modified during the system's life cycle.

- Verifiability: defines the verifiability degree of the requirement, that is, indicates in which degree is possible to check that the requirement has been implemented in the system. It can take the following values:
  - High: it can be verified that the requirement has been implemented. This type of requirements correspond to the basic system functionality.
  - Medium: it can be verified that the requirement has been implemented but requires a complex verification or involves the system source code.
  - Low: it is difficult to verify if the requirement has been implemented. In some cases is not possible.

3.2.5.2 Functional requirements textual description

In this section the functional requirements are presented. This requirements define the system functionality.

System Requirement			
ID	FR-001	**Source**	UC-003.
Name	Execution script.		
Description	Creation of a script with the commands to start the system execution .		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.13: Functional requirement 001.

System Requirement			
ID	FR-002	**Source**	Client and UC-004.
Name	Execution without user interaction.		
Description	Once the execution has started, the user does not have to interact with the system to ensure the correct execution (the system is autonomous).		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.14: Functional requirement 002.

System Requirement			
ID	FR-003	**Source**	Analyst.
Name	Continuous connection between all components.		
Description	All modules as well as the robots must be connected during the whole execution process.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.15: Functional requirement 003.

System Requirement			
ID	FR-004	**Source**	Client.
Name	Modify Sokoban PDDL domain.		
Description	Modify the original Sokoban PDDL domain to introduce the changes required for this project.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.16: Functional requirement 004.

System Requirement			
ID	FR-005	**Source**	Analyst.
Name	Create problems in PDDL.		
Description	Create a set of PDDL problems following the PDDL domain restrictions to define the tasks and for the experimentation.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.17: Functional requirement 005.

System Requirement			
ID	FR-006	**Source**	Client.
Name	Solve the problem.		
Description	The system must be able to solve the problems proposed using automated planning.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.18: Functional requirement 006.

System Requirement			
ID	FR-007	**Source**	Analyst.
Name	Generate a plan.		
Description	The system must be able to decompose the problem solution in a sequence of actions.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.19: Functional requirement 007.

System Requirement			
ID	FR-008	**Source**	Analyst.
Name	Send action.		
Description	The Monitoring module must be able to send actions to the Execution module that must be able to receive them.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.20: Functional requirement 008.

System Requirement			
ID	FR-009	**Source**	UC-005
Name	NAO Walk.		
Description	The NAO robot must walk when received the order.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.21: Functional requirement 009.

System Requirement			
ID	FR-010	**Source**	UC-005 and Client
Name	NAO Walk correction.		
Description	The NAO Execution module must be able to correct the *walk* action. Due to the inherent error, the movement must be just acceptable to allow the correct performance of the system.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.22: Functional requirement 010.

System Requirement			
ID	FR-011	**Source**	UC-006
Name	NAO Turn.		
Description	The NAO robot must turn when received the order.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.23: Functional requirement 011.

System Requirement			
ID	FR-012	**Source**	UC-006 and Client.
Name	NAO Turn correction.		
Description	The NAO Execution module must be able to correct the *turn* action. Due to the inherent error, the movement must be just acceptable to allow the correct performance of the system.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.24: Functional requirement 012.

System Requirement			
ID	FR-013	**Source**	UC-009
Name	P3DX Move.		
Description	The P3DX robot must be able to walk when received the order.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.25: Functional requirement 013.

System Requirement			
ID	FR-014	**Source**	UC-009
Name	P3DX Stop.		
Description	The P3DX Execution module must be able to calculate when to stop the P3DX to complete the *move* action.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.26: Functional requirement 014.

System Requirement			
ID	FR-015	**Source**	UC-010
Name	P3DX Turn.		
Description	The P3DX robot must be able to turn when received the order.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.27: Functional requirement 015.

System Requirement			
ID	FR-016	**Source**	UC-007 and Analyst.
Name	NAO take picture.		
Description	The NAO robot must be able to take a picture when received the order.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.28: Functional requirement 016.

System Requirement			
ID	FR-017	**Source**	UC-007 and Analyst.
Name	NAO move head.		
Description	The NAO robot must be able to move the head to take several pictures at different angles.		
Significance	Desirable	**Priority**	High
Stability	Unstable	**Verifiability**	High

Table 3.29: Functional requirement 017.

System Requirement			
ID	FR-018	**Source**	UC-007 and Analyst.
Name	Detect QR.		
Description	The NAO Execution module must be able to detect a QR in the picture.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.30: Functional requirement 018.

System Requirement			
ID	FR-019	**Source**	UC-007 and Analyst.
Name	Read QR.		
Description	The NAO Execution module must be able to read a QR.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.31: Functional requirement 019.

System Requirement			
ID	FR-020	**Source**	UC-011.
Name	Push.		
Description	The P3DX robot must be able to push a box to the next cell and finish in its position.		
Significance	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.32: Functional requirement 020.

3.2.5.3 Non-functional requirements textual description

In this section the non-functional requirements are presented. These define characteristics that the system must have related to its operation.

System Requirement			
ID	NFR-001	**Source**	Analyst.
Name	Security		
Description	The system will not jeopardize the robot's or user's integrity during the execution.		
Benefit	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.33: Non-functional requirement 001.

System Requirement			
ID	NFR-002	**Source**	UC-003 and UC-004
Name	Autonomy		
Description	The robots will have to be switched on and with battery for the whole execution.		
Benefit	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.34: Non-functional requirement 002.

System Requirement			
ID	NFR-003	**Source**	Operational environment.
Name	Use the NAO robot.		
Description	The coordinator robot will be the NAO robot.		
Benefit	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.35: Non-functional requirement 003.

System Requirement			
ID	NFR-004	**Source**	Operational environment.
Name	Use the P3DX robot.		
Description	The cargo robot will be the P3DX robot.		
Benefit	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.36: Non-functional requirement 004.

System Requirement			
ID	NFR-005	**Source**	Analyst.
Name	Use Java.		
Description	The programming language used to program and for any library or API is Java.		
Benefit	Essential	**Priority**	High
Stability	Stable	**Verifiability**	High

Table 3.37: Non-functional requirement 005.

3.3 System design

In this section is explained in detail the system design. First, the problem domain design in PDDL is described in section 3.3.1. Second, the Computer Vision techniques analysis is presented in section 3.3.2. Then, a general vision of the whole system architecture is presented in section 3.3.3. In section 3.3.4 each component is described with its functionality. Finally, it is described how the system works in section 3.3.5.

3.3.1 Problem design

To model the warehouse tasks in PDDL it has been used the Sokoban PDDL definition because the structure and the actions are similar. The modifications included in the Sokoban Domain to create the Warehouse Domain are described next:

- Colour for the boxes. This includes predicates to define the color of the boxes and which boxes had their colour detected or not.
- Multi-agent. The original Sokoban domain had a single agent. For this case, two type of agents exist and more than two agents can exist in a problem.
- Additional actions. For the NAO robot the *turn* and *get_color* actions were added. For the P3DX, the *turn* action and *push* action were modified.

The objects defined in the modelled domain are presented in table 3.38 with their names and their types.

Object	**Type**
LOC (location)	Object
DIR (direction)	Object
Thing	Object
Box	Thing
Robot	Thing
P3DX	Robot
NAO	Robot
Colour	Object

Table 3.38: Objects in the domain.

The predicates defined in the modelled domain are presented in table 3.39 with the properties or relations between objects that they model under the *Characteristics* column.

Predicate	**Characteristic**
(at ?t - thing ?l - LOC)	A thing t is in cell with location l of the board.
(robot-dir ?robot - robot ?d - DIR)	Robot robot is facing direction d.
(at-goal ?b - BOX)	Box b is in a goal cell.
(is-goal ?l - LOC)	The cell l is a goal cell.
(adjacent ?l1 - LOC ?l2 - LOC ?d - DIR)	Cell l1 is adjacent to cell l2 in the direction d. Than is, you can go from l1 to l2 moving in the direction d.
(clear ?l - LOC)	Cell l can be accessed.
(colour-cell ?l - LOC ?c - colour)	Cell l has colour c.
(has-colour ?b - BOX ?c - colour)	Box b has colour c.
(known-colour ?b - BOX)	The colour of box b has been identified.
(unknown-colour ?b - BOX)	The colour of box b has not yet been identified.
(no-storage ?b - BOX)	Box b is not yet is a goal cell.

Table 3.39: Predicates in the domain.

The actions defined in the modelled domain are presented in table 3.40. For each action it is described: (1) the preconditions that need to be true before

the action's execution and (2) the effects after the execution given by an *added predicates* list and a *deleted predicates* list.

Action name	Preconditions	Added	Deleted
Move	(clear ?to) (at ?robot ?from) (adjacent ?from ?to ?dir) (robot-dir ?robot ?dir)	(at ?robot ?to) (clear ?from)	(at ?robot ?from) (clear ?to)
Push-goal	(at ?robot ?rloc) (at ?b ?bloc) (clear ?floc) (robot-dir ?robot ?dir) (adjacent ?rloc ?bloc ?dir) (adjacent ?bloc ?floc ?dir) (is-goal ?floc) (known-colour ?b) (has-colour ?b ?c) (colour-cell ?floc ?c) (no-storage ?b)	(at ?robot ?bloc) (at ?b ?floc) (clear ?rloc) (at-goal ?b)	(at ?robot ?rloc) (at ?b ?bloc) (clear ?floc) (no-storage ?b)
Push-not-goal	(at ?robot ?rloc) (at ?b ?bloc) (clear ?floc) (robot-dir ?robot ?dir) (adjacent ?rloc ?bloc ?dir) (adjacent ?bloc ?floc ?dir) (known-colour ?b) (no-storage ?b)	(at ?robot ?bloc) (at ?b ?floc) (clear ?rloc)	(at ?robot ?rloc) (at ?b ?bloc) (clear ?floc)
Get-colour	(at ?robot ?rloc) (at ?b ?bloc) (adjacent ?rloc ?bloc ?dir) (robot-dir ?robot ?dir) (unknown-colour ?b)	(known-colour ?b)	(unknown-colour ?b)
Turn	(robot-dir ?robot ?dr)	(robot-dir ?robot ?db)	(robot-dir ?robot ?dr)

Table 3.40: Actions in the domain.

The PDDL domain is included in Appendix C. Besides, a benchmark of problems has been created in order to perform the experimental evaluation. Each problem defines the warehouse enviroment (cells and connections among them), the positions of the robots and boxes and the cell colours.

3.3.2 Computer Vision module

In order to detect the box colour several Computer Vision techniques have been contemplated. They were studied in the first stages of the dissertation and their possible viability was discussed to finally chose the one that seemed the most appropriate. The three possible ideas were: colour histograms, blobs and QR

detection. QR detection was the one chosen. The colour of the box and its identifier are stored in the QR but more information could be stored to extend the functionality of the system.

3.3.2.1 Detection mechanism based on Histrograms

The main idea for colour histograms was to extract the histograms of an input image and determine which colour was maximum, red or blue (in RGB space), since the boxes were decided to be of those two colours. This is based on the idea that if there is a blue box in the image the blue colour will be maximum in the histograms, thus identifying the box and the same for the red one.

However, histograms do not determine the position of the colour, just the quantity. Therefore, an image with a blue box in the centre could have the same histograms as an image with several small blue boxes scattered around. Also, the surroundings of the box in the image would affect the colour level and a threshold would be needed to determine when a colour is considered to have enough intensity to determine if it is a blue or a red box. In addition, in RGB variations of a color are a mixture of the three primary colours. If the red colour is dark it would also have blue color. In this case, the histogram would not be maximum for one colour and it would not be accurately detected. Histograms are, therefore, more useful to detect variations of colour in an image.

3.3.2.2 Detection mechanism based on Blobs

The blob detection idea was also analysed. The algorithm would find blobs in the image according to a colour (in the HSV space) and a minimum area given. To specifically find the boxes, their colour and minimum area needed to be known so that they could be passed as parameters to the algorithm. Given an image, the algorithm would return if there was a box or not in it and its colour. Therefore, before each system execution or every time the light conditions changed, a calibration process would be needed to determine the parameters for the boxes and colours that are going to be used. This calibration process would be manual: the

robot would take a picture of each box and the developer would carry out the image processing to determine the values and set them in the program.

However, one major problem is found when determining the value of the parameters. Since boxes are displayed in a real environment with several light sources, the colour value for one box under certain light sources is not the same as the one for another box in another location of the environment, with light incising differently. It could require, then, to have different parameters for each box and calibration process and this is, clearly, not practical nor scalable. In addition, the calibration process should be automatically done, but that would require an initial phase where the NAO robot is located in front of each box and process an image of the box with an algorithm that would need to be implemented. This algorithm would carry out the tasks that are done manually at the moment and would need to be repeated everytime the lightning conditions changed.

The approach was discarded due to the external factor of the real environment and the tedious calibration process, the blob idea was not very practical and seemed the less appropriate.

3.3.2.3 Detection mechanism based on QR codes

The QR option seemed to be the best one. This would require that each box has in each side a QR barcode glued with information about its ID and colour. The QR technique eliminates the idea of detecting the colour and processing an image. It also has no threshold like the histograms and it is accurate in the information that it provides. An advantage for future work is that other information can be stored in the QR very easily. This would allow to extend the systems box ordering criteria. Finally, no calibration process is required. Thus, the QR is the technique chosen.

3.3.3 System architecture

The system architecture for this project uses the 3-module configuraiton of PELEA, described previously in section 2.2.4. This architecture allows modules of three types: Monitoring, Decision Support and Execution. For this work, I have configured the architecture to deploy 4 modules: one Monitoring, one Decision Support and two Execution. Each of the execution modules implements two features: the robot low-level functionality (robot controller) and the partial state representation.

The partial state representation describes the world from the point of view of the robot. An action execution can produce changes in the environment which are, then, updated by the robot in its partial model. Changes made in the environment by the other robots will not be visible in its own model until Monitoring updates it. The Monitoring module merges the information that receives from the different robots in order to keep a global state of the environment. Since the Monitoring module is the one sending the actions from the plan to the robots, it also ensures that the robots do not collide since it has the state partial model from both robots.

The whole architecture is depicted in Figure 3.3 showing how these modules are connected among them. If more robots were added to the system, it would only require to add an Execution module for each additional robot. In this architecture the NAO Execution module has been entirely developed. The Monitoring and Decision Support modules have not been modified and the P3DX Execution module has a few small modifications to adapt it to this dissertation.

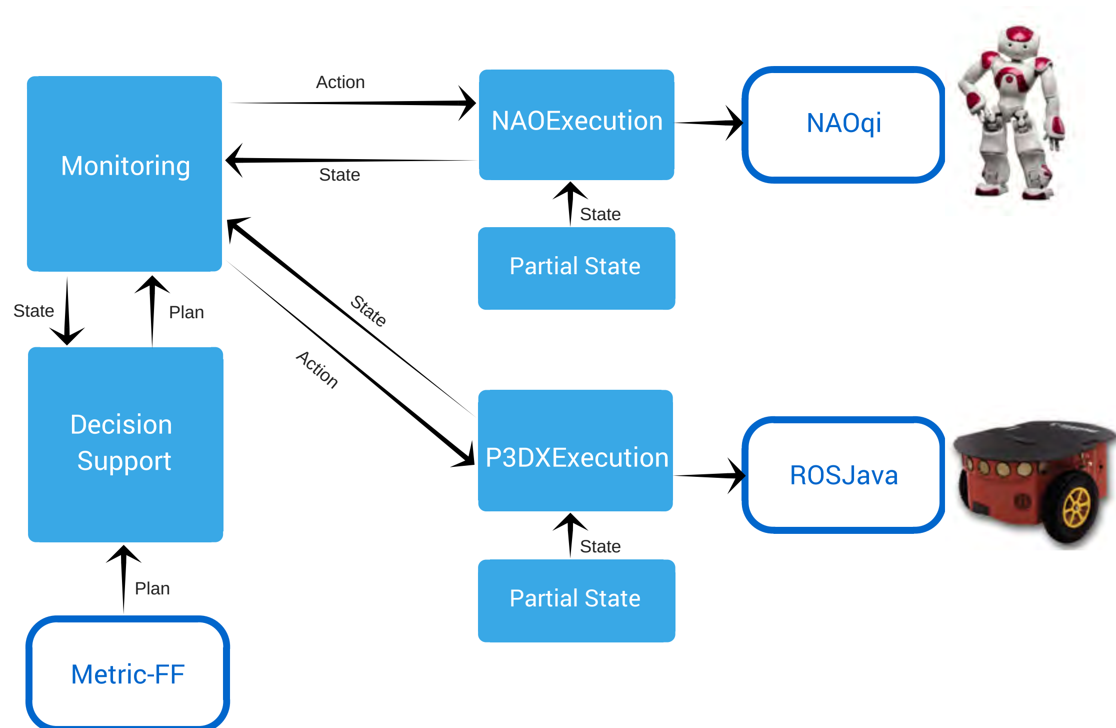


Figure 3.3: System architecture diagram.

3.3.4 Components description

In this section, the four components of the system will be described indicating the modifications that have been made will be described.

3.3.4.1 Monitoring module

The monitoring module is the main module in the architecture. This module synchronizes the execution of the other modules using messages. This messages are sent using Java RMI. Besides, this module deploys the main monitoring algorithm that controls the planning and execution process.

The execution can be started by Monitoring or by one Execution module which requests to Monitoring solve a problem. Once the execution has started, Monitoring will receive the plan from Decision Support. Then, it will have to send

the actions one by one to both Execution modules. After each action execution, it updates the state of the environment with the information received from the Execution modules and checks whether the goals have been reached. In case they have the execution would stop. Otherwise, before sending the next action, the Monitoring module compares the predicates in the current state with the preconditions of the next action that is going to be executed. If the preconditions are fulfilled then the action is executed. Otherwise, it means that the state is not the expected one and requests for a new plan to Decision Support.

No modifications have been made to this module since the algorithm that it uses is perfectly valid for this dissertation and no further functionality is needed.

3.3.4.2 Decision Support module

This module generates a plan of actions for a domain and a specific problem. There is no need to monitor variables other than the state, domain and plan. Therefore, it will not have to determine any extra monitoring information for the Monitoring module.

The Decision Support module offers two functionalities: (1) planning from scratch and (2) replanning or repair. Decision Support is called at the beginning of the execution to provide an initial plan which is generated from scratch. If the execution continues as expected in this first plan then Decision Support is not called again. However, if, at some point of the execution, the state is not the expected one the Decision Support module is called to fix this. For instance, this situation would happen if the NAO robot is not be able to detect the QR of a box. When such situations happen, Decision Support is called to replan. In this case, it determines if the plan can be repaired (2) or if it needs to generate a new plan using the current state as initial state (1). The new plan is sent to Monitoring and the execution continues, with the new valid plan.

For this module, no modifications have been made since the algorithm that it uses is perfectly valid for this dissertation and no further functionality is needed.

3.3.4.3 NAO Execution module

This module has been the main work in this dissertation. It has been entirely programmed to implement the functionality for the NAO robot: move, turn and colour detection. To achieve them these libraries have been used:

- JNAOqi: it is an API in Java provided by the robot manufacturers Aldebarán Robotics to control de actuators (servos) and collect information from the sensors (gyrometer, accelerometer, cameras, etc).
- Zxing: it is an open-source API in Java for QR image processing.

Figure 3.4 shows the different parts this module has and their interactions. The NAO Execution module is represented by the black box. Inside it has the different parts. First, there is the controller that implements the control primitives (move, turn and get color) which corresponds to the PDDL actions. Second, the Zxing library. Finally, the partial state that stores the state of the environment from the NAO point of view. The NAO Execution to the NAO robot using TCP/IP.

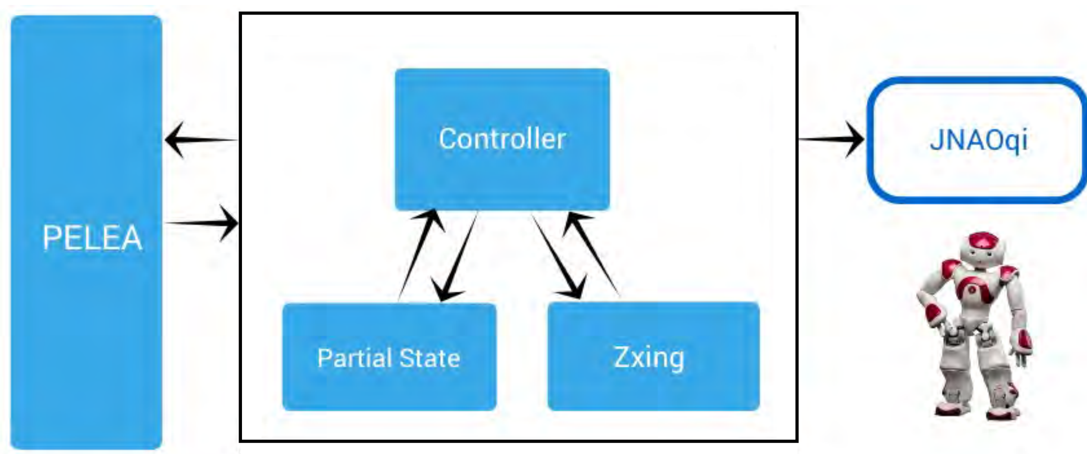


Figure 3.4: Components and connections for the NAO Execution module.

Turn

The turn primitive allows the robot to make a turn (in radians). The direction of the turn, either left or right, is given by the sign of the degrees (negative or positive).

The robot has an internal reference system which consist on a circumference (in radians) from 0 to Π (first half) and from –Π to 0 (second half), instead of from 0 to 2Π. Due to this fact, for calculations in the turn, the angle is converted to an angle between 0 and 2Π to make it easier. The robot calculates its position using this reference system. Therefore, at every moment, the robot is at a certain position, which is expressed as an angle in radians with respect to the reference system.

Given that the board has squared cells and the robot can be in the board facing four possible directions, the turn function receives as an argument the new direction (orientation) to face. Therefore, the degrees to turn will be calculated taking into account the initial and the new orientation and can either be 90 degrees (to the left or to the right) or 180 degrees to turn.

The main problem with the turn is that the robot staggers while walking due to how the movement is performed. This process generates an error which has been corrected using the algorithm shown in Figure 3.5.

The algorithm uses an error threshold in degrees that has been chosen during the developed process manually. It is set to Π/16 (11.25 degrees), which means that if the error is smaller the position will not be corrected. It cannot be set to a smaller value because given the NAO's feet size, it cannot turn such small values with precision. The algorithm works in the following way:

1. The robot performs a turn (initial turn). Then, the actual degrees that the robot has moved are computed. This value is computed using the internal reference system that the robot has: store the position before the turn and after the turn and subtract them.

2. Compute the error (in radians) by subtracting the expected degrees to turn

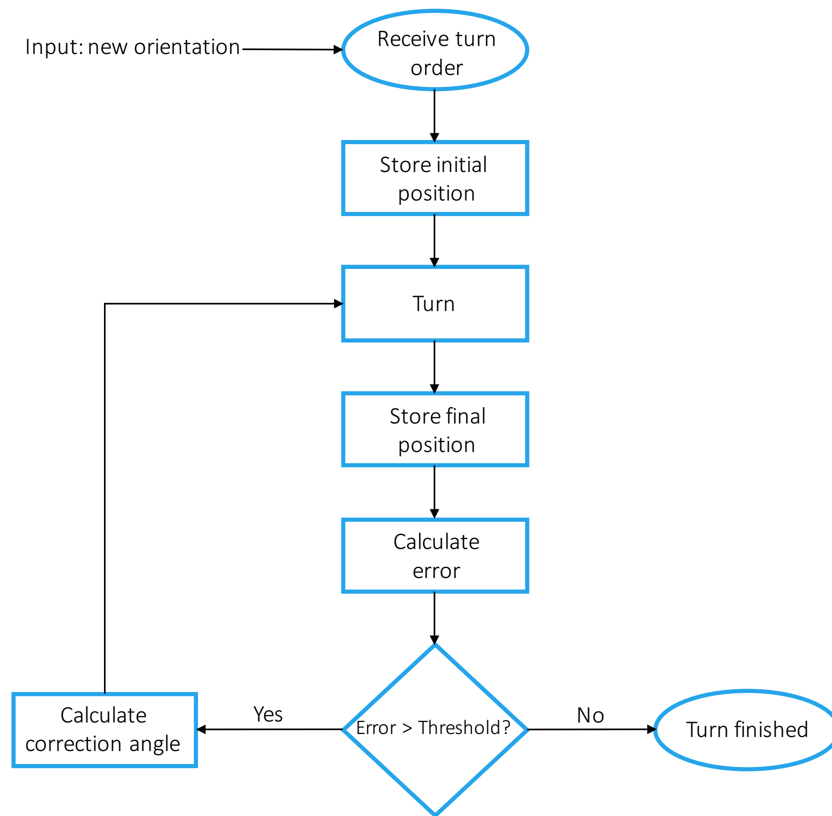


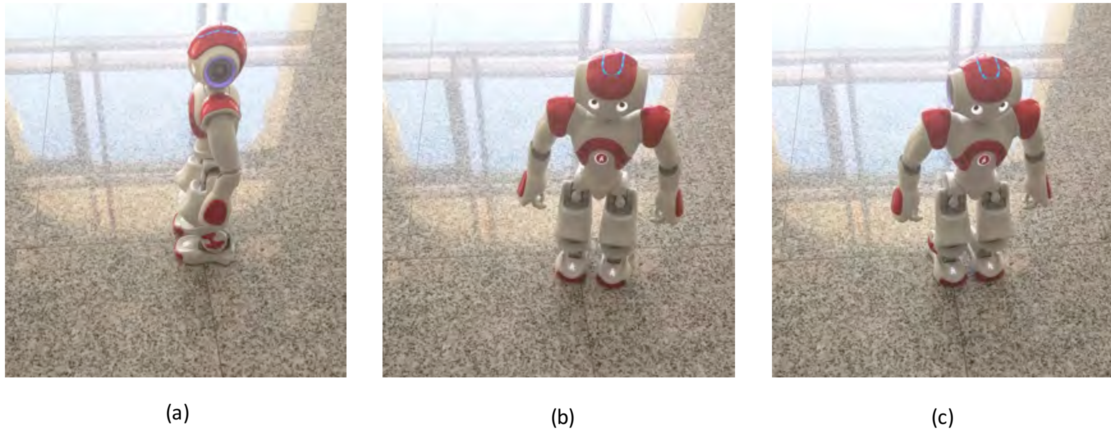
Figure 3.5: Flowchart for the turn correction algorithm.

and the actual degrees turned. If the error is smaller than a threshold then the turn is acceptable, otherwise, it needs to be corrected.

3. If an error is detected, a second turn is performed (correction turn) in the opposite direction. The direction of the correction turn needs to be calculated taken into account the direction of the initial turn and if the error was by excess or by defect. Once the direction has been calculated the algorithm is repeated (go to step one) for the correction turn until the error is smaller than the threshold. For the correction turn the degrees to turn is the error magnitude.

Figure 3.6 shows an example of the turning process of the NAO robot. The first image (a) shows the position before the turn. The NAO will turn 90 degrees to its left. The second image (b) shows the position after having turned. The NAO has turned less than 90 degrees. Finally, the third image (c) shows the final

position after having corrected the turn.



(a) (b) (c)

Figure 3.6: Initial position (a) 90 degrees turn with error (b) Position after correction (c).

Walk

The walk primitive allows the robot to walk straight given a distance in meters. When walking straight, the robot has an error that increases with the distance, making it unable to accurately walk in a straight line. This is due to both the walking mechanism implemented in NAOqi and the floor texture.

A correction mechanism has been implemented in order to compensate this error. This algorithm is similar as the one implemented for the turn and it uses as well the internal reference system. Figure 3.7 shows the flowchart of the correction algorithm.

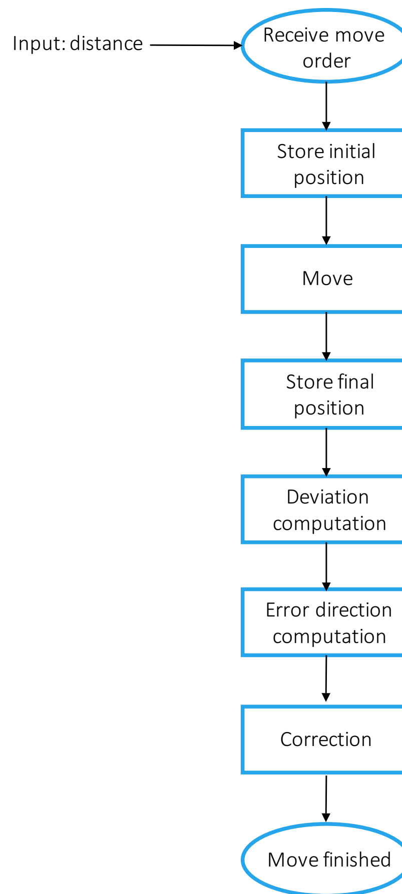


Figure 3.7: Flowchart for the walk correction algorithm.

1. Before executing the *move* action, the current position is stored (initial position) with respect to the internal reference system (an angle in radians).

2. The action is executed and the current position is stored (final position).

3. Error calculation. This process is composed of two steps:

    (a) Deviation computation. The deviation is calculated subtracting the initial position from the final position to get the deviation that the action has had. This value is calculated with respect to the internal reference system explained in the *turn* action. Consequently, the deviation is in radians.

    (b) Error direction computation. Given that the angles are in the range [0,

2Π], depending on the sign of the error calculated in the previous step, it can be determined if the deviation was to the left or to the right.

4. Correction. The correction is performed using the information computed in the previous phase. First, the error sign is changed according to the error direction (it should correct in the opposite direction). Then, a turn is performed to correct the deviation (which is in radians), so that the robot positions itself where it should have ended if the walk action had been performed accurately.

In this case, no threshold is needed because for the distance walked the error is very small. Also, the turn to correct the error has no error correction because it is also very small and the NAO robot performs it accurately.

Colour detection

This module controls the extraction process of the QR codes from the boxes and decodes it. The algorithm has been implemented using the ZXing library. The algorithm is composed of three main operations:

- Take picture: The NAO robot takes a picture of the box using the top camera. To ensure that the NAO will capture in the picture the QR, it has been programmed to take three pictures at different angles of the head. Therefore, for the first picture it will move the head upwards to take it. For the second picture it will position the head looking at middle distances. For the third picture it will move the head downwards to take the picture looking down. This procedure will allow to identify QR in boxes of different sizes and at different distances since it covers the whole range of possible vision.

- Get QR: identify the QR in the picture taken with a pre-processing method.

- Get colour: read the QR to obtain its information, which is the box's colour.

Figure 3.8 shows the flowchart of the algorithm described below. The algorithm receives no input.

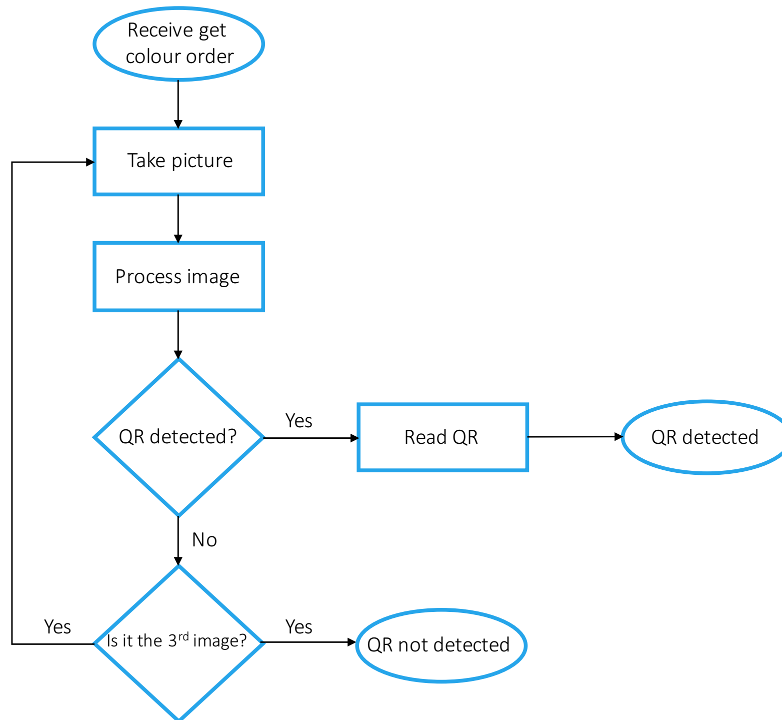


Figure 3.8: Flowchart for the colour detection algorithm.

This algorithm does not ensure the detection of the QR. If the QR is not detected PELEA will replan. On the other hand, the walk and turn actions were ensured and every time the NAO executes them it completes them. In this dissertation, the robot is supposed to be reliable for the *walk* and *turn* primitives.

Partial state representation

As briefly introduced in previous sections, the NAO robot will keep a model of the world from its point of view. After performing each action the robot will update the model with the changes produced in the environment. For the NAO robot, it has three possible actions that have the following changes in the world:

- Walk. When the NAO walks from one cell to another its position changes. The position that the NAO has walked to is calculated and updated in the model.
- Turn. When the NAO turns its orientation changes and it faces another direction. This is also a change that is calculated and updated in the model
- Get a colour. When the NAO reads a QR code from a box and unveils its colour it introduces it into the environment. The box's colour is no longer unknown.

3.3.4.4 P3DX Execution module

The P3DX Execution module implements the P3DX functionality and the partial state representation module. Similar to the NAO module, it can be grouped into sub-modules. The controller module implements the actions of move, turn and push. Then, it has the module to handle the partial state representation. It connects to PELEA through RMI and to rosjava to communicate with the robot through a COM4 wire. These modules are depicted in Figure 3.9 where the P3DX Execution module is the black blox with the sub-modules inside.

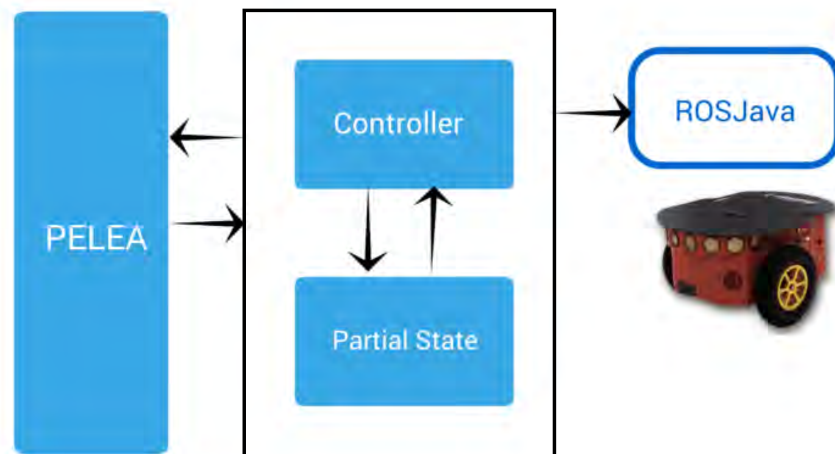


Figure 3.9: Components and connections for the P3DX Execution module.

ROS (Robotic Operating System) is the OS that the P3DX uses. ROS works with three elements: nodes, messages and topics. Each node is a process that can publish messages to a topic or subscribe to a topic to receive messages. To control the P3DX, a node in the remote computer is used to connect to the robot trough a TCP port. The node is subscribed and is a publisher in topics that control the robot. Therefore, by publishing on them, the robot is controlled. This node is the *p2os_driver*. Its installation is explained in the appendix A.

The node is in a launch file in which the node parameters are specified. One of them is the port to which it has to connect. Therefore, the available port has to be identified before starting the execution. That has been done in experimentation and it is usually the same port always, in this case */dev/ttyUSB0*.

Minor modifications to the algorithms to adjust the actions to the board size were made, but overall no modifications were necessary in this module.

Move

Contrary to the NAO robot, the P3DX robot does not move a given distance. Instead, it receives as an input a value for the linear velocity and it will start moving at that velocity infinitely until it is stopped. To move a specific distance, it is calculated the time that it takes the robot to travel the distance at a given

speed. When the time has passed it is stopped. Notice that, before moving, the motor needs to be on.

To send the velocity to ROS, the P3DX Execution module publishes it to the topic *cmd_vel*. Then, the P3DX Execution module calculates the time needed for the P3DX to move from one cell to another given the velocity. To calculate this, the dimension of the cells is passed to the P3DX Execution module. When the time is finished it publishes again to the same topic to send velocity zero and stop the P3DX. The topic *cmd_vel* uses messages of the type *geometry_msgs/Twist*. These have two tridimensional vectors (x,y,z). The first vector is for linear velocity and the second one for angular velocity. Therefore, this same topic can be used for the second action: turn.

Turn

Turning the P3DX follows the same idea as in the *move* action. However for this one, the *p2os_driver* node will publish a message with only angular velocity to turn. To calculate the time it takes to turn, the P3DX Execution module will use in this case the degrees to turn. In addition, it calculates the direction of the turn. The possible turns are: 90 degrees left, 90 degrees right and 180 to either direction.

Push

The push functionality consist on pushing a box to the next cell. After that, the P3DX must end in the cell where the box previously was. Therefore, it moves a cell forward. The box does not occupy the whole cell and it should be placed in the center of the next cell when pushed. Also, because the size of the cells is larger than the robot, once the P3DX has pushed the box it is not in the cell it should be. Consequently, it will need to move backwards to position itself in the cell where it should be.

To achieve so, the P3DX performs three movements for the push action. In the first one, it moves the distance to the center of the cell where the box is (Move 1). In the second one, it moves the distance to push the box to the next cell and

leave it in the center (Move 2). In the third one, it moves backwards the same distance as in the second move (Move 3) so that the P3DX ends in the center of the cell where the box was. This concept is illustrated in Figure 3.10. The three stages for the turn action are depicted with the distances mentioned before. Each square is a cell.

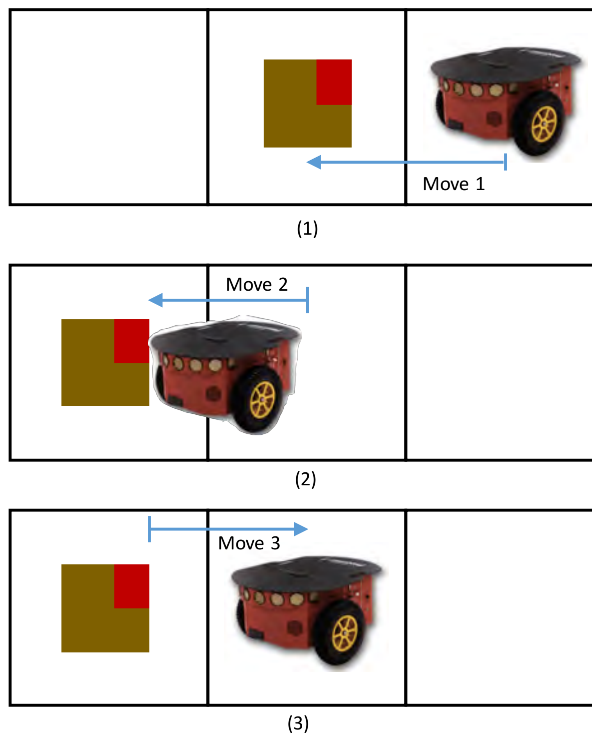


Figure 3.10: Stages for the push action.

Partial state representation

The P3DX robot also keeps a model of the world from its point of view. After performing each action the robot will update the model with the changes produced in the environment. For the P3DX robot, it has three possible actions that have the following changes in the world:

- Move. When the P3DX moves from one cell to another its position changes. The position that the P3DX has moved to is calculated and updated in the model.
- Turn. When the P3DX turns its orientation changes and it faces another direction. This is also a change that is calculated and updated in the model
- Push a box. When the P3DX pushes a box it introduces changes into the environment. The box's position has changes and it could have been pushed to a goal destination or not. Also, the P3DX position is updated.

3.3.5 System operation

For the planning-execution process to start, all modules need to register with Monitoring. The modules communicate with each other using RMI. The planning and execution process starts when one of the coordinator robots request to Monitoring solve a planning task by sending a PDDL domain and problem. Then, the Monitoring module requests a plan to the Decision Support module. This plan is generated from scratch using the initial state and the domain. Next, the Monitoring module iteratively sends every action a_i in π to the corresponding module. The Execution modules execute the action and respond to Monitoring with their partial state. After every action execution, a new observed state is built using the partial states of the execution modules. Then, the Monitoring module checks if the next action can be executed according to the observed state. If the action cannot be executed a new plan is requested to Decision Support. This process is repeated until the goals are reached.

Figure 3.11 presents the sequence diagram for the system operation. Each object is represented with a colour.

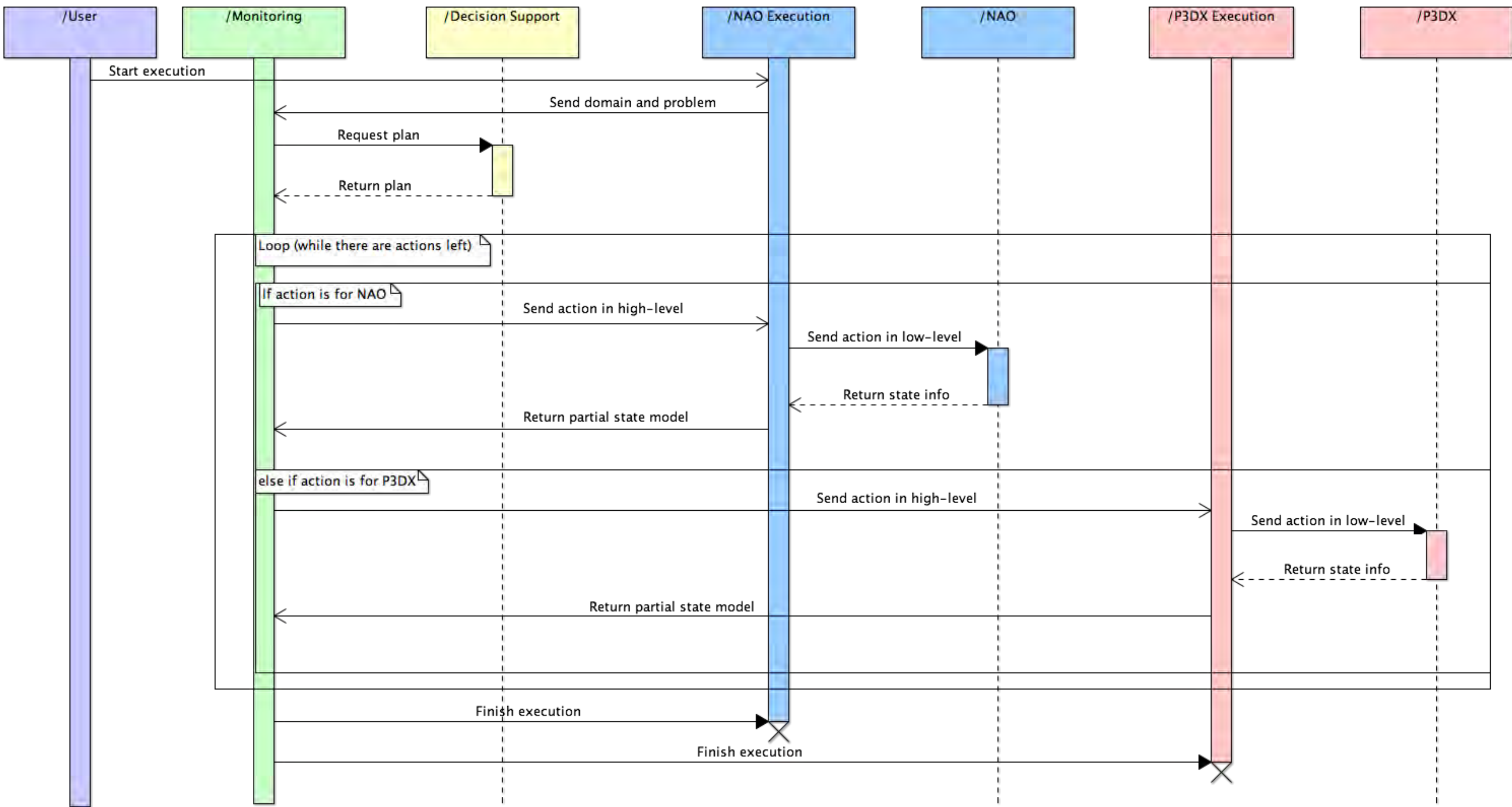


Figure 3.11: Sequence diagram for the system operation.

Chapter 4

Experiments

In this section it is explained in detail the experimentation that has been carried out. The experimentation is divided in two parts. In the first one, the functionality that has been developed for the NAO robot has been tested in unitary experiments. In the second one it is conducted the experimentation for the whole system.

The environment used is described in section 4.1. The unitary experimentation is detailed in section 4.2 and the system experimentation in section 4.3.

4.1 Experimentation environment

The experimentation has been conducted in the hallway 2.1.B in the Sabatini Building at Carlos III University. The characteristics of the environment are:

- Lighting. There is natural light as well as artificial light. There are big windows all along the corridor which makes natural light predominant. Only if there is no natural light then the lightness can be modified using the artificial light. However, this has not been the case for the experiments, where natural light was the main form of lighting.

- Floor. This environment has marble tiles that made the NAO robot slightly slip. In addition, the tiles are shiny and reflect the boxes, the QR and the NAO.

- Cells. The cells in the board are 0.9x0.9 cm. They have been marked on the floor with adhesive tape for the experiments since the floor tiles had different sizes.

For each experiment, the environment is graphically show in a picture. Figure 4.1 shows the legend used for this models.

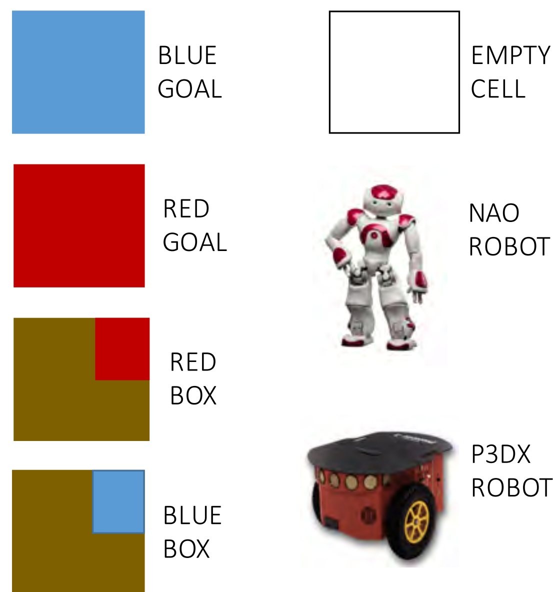


Figure 4.1: Environment diagrams legend.

4.2 NAO unitary experiments

The NAO unitary experiments have been conducted to test the functionality developed for the NAO robot. This functionality is composed of three actions: $get-colour$, $walk$ and $turn$. The experimentation is subdivided in two parts: Vision experiments and Movement skills experiments. The latter one groups both the $walk$ and $turn$ actions while the former is just for the $get-colour$.

This experimentation has used partial execution of the system using only the NAO robot in real environments. For each experiment is presented the description of the environment, the objectives and the results.

4.2.1 Vision experiments

Computer Vision techniques usually have limitations due to lighting conditions. In the real environment available for the experimentation in this work there is mainly natural light that comes from one source, therefore just from one direction. Given that natural light cannot be controlled, unitary experiments have been carried out to determine how it affects the QR detection. Also, the $get - colour$ functionality is tested.

The experiments have been conducted between 12 pm (noon) and 6 pm. Before 12 pm the sun light occupied the hallway and created backlighting.

4.2.1.1 Vision experiment 1

The environment for this experiment is presented in Figure 4.2. The NAO orientation is North. The goal of the problem consists on identifying the color of the box located in position (1,2). In this environment the NAO reads the QR on the side of the box that faces the window and receives natural light directly. Figure 4.3 shows the robot and the box during the planning and execution process. The objective of this experiment is to determine the light influence in the Computer Vision system for the box position and the NAO position to get the QR.

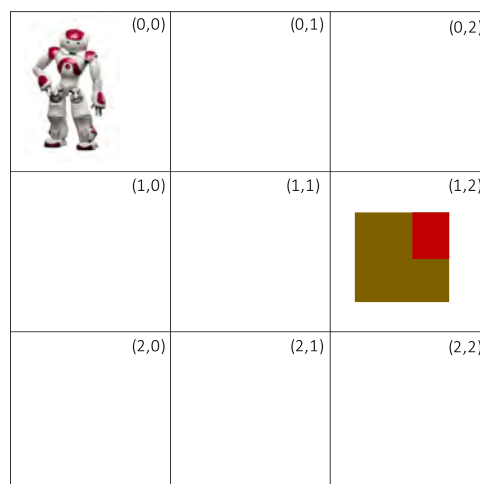


Figure 4.2: Experiment 1 environment diagram.

Figure 4.3: Experiment 1 real environment.

After the experiment execution, it could be seen that light incises directly in the box and increases the brightness of the picture. This made the detection of the QR unsuccessful since it was hard to distinguish. The QR was not detected in any of the three pictures taken. Figure 4.4 shows the pictures taken by the NAO robot. The picture on the left is the second picture taken and the picture on the right the third picture taken.

Figure 4.4: Experiment 1 NAO pictures.

4.2.1.2 Vision experiment 2

The environment for this experiment was changed to have the box facing the opposite direction as before. Figure 4.5 shows the environment diagram and Figure 4.6 the picture in the real environment. The goal of the problem consists on identifying the color of the box located in position (1,0). The objective of this experiment is to determine the light influence in the Computer Vision system for the new box position as well as the influence of NAO's new initial position.

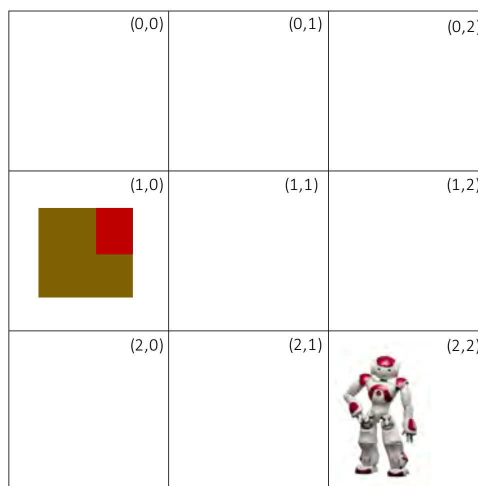


Figure 4.5: Experiment 2 environment diagram.

Figure 4.6: Experiment 2 real environment.

In this case, the lighting is even worse for the picture since it is very dark for the NAO to detect the QR in any of the pictures. This is due to backlighting. The pictures where the box was visible taken by the NAO robot to detect the QR are shown in Figure 4.7. The picture on the left is the second picture taken and the picture on the right the third picture taken.

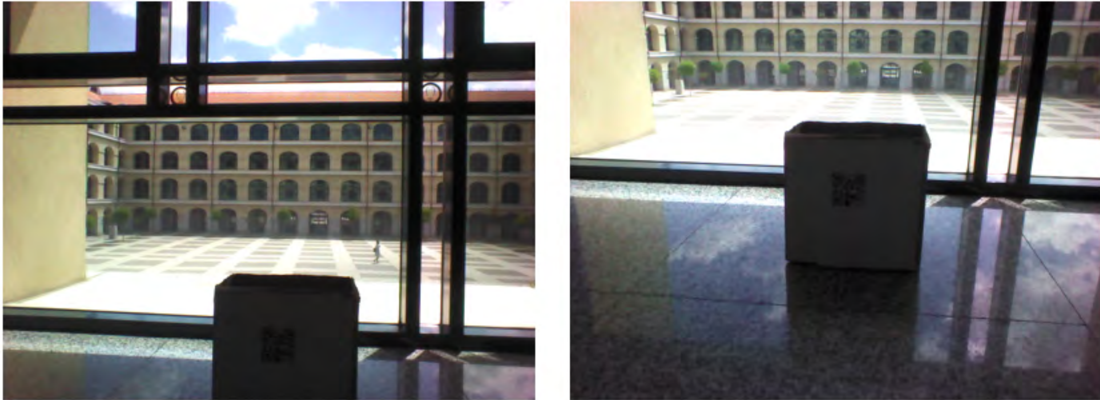


Figure 4.7: Experiment 2 NAO pictures.

4.2.1.3 Vision experiment 3

In this last experiment, the environment was changed to have the light incising in the box from the side when the NAO takes the picture. Figure 4.8 depicts the environment diagram and Figure 4.9 the real environment. The objective of this experiment is to determine the light influence in the Computer Vision system for the new box position and the new NAO position to get the QR.

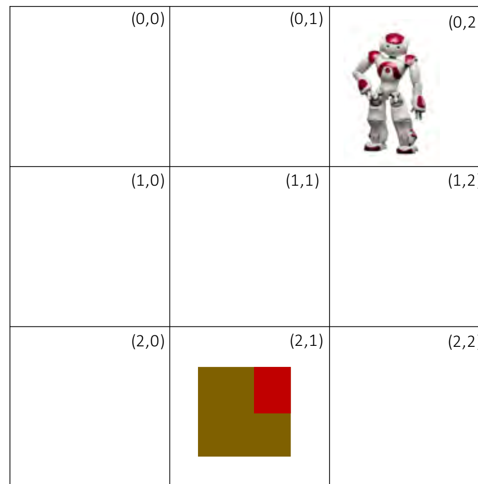


Figure 4.8: Experiment 3 environment diagram.

Figure 4.9: Experiment 3 real environment.

For this experiment the results were successful. The light allowed the QR to be detected in the second picture so there was no need to take a third picture. In addition, the detection of the QR was carried out as expected and the information from it could be retrieved (box name and colour). The pictures where the box was visible taken by the NAO robot to detect the QR are shown in Figure 4.10. In this case, the box was not visible in the first picture taken but it was visible in the second picture taken and the QR also detected in it.

Figure 4.10: Experiment 3 NAO pictures.

4.2.1.4 Vision experiment 4

This experiment was conducted to analyse if the QR size affects its detections. The two sizes that were tested are shown in Figure 4.11. No specific problem was used for this experimentation. For the experiment, the NAO had a box in front and executed the $get - colour$ action. The objective of this experimentation is to determine if the QR size is determinant for its detection.

(a)

(b)

Figure 4.11: Boxes with different QR size.

The experiment was executed 10 times, 5 for each size. The big QR had 4 successful detections and the small had 2 successful detections. It can be concluded that the larger QR is better than the smaller.

4.2.1.5 Vision experimentation conclusions

From the experimentation it can be concluded that light affects the QR detection. In situations when it strikes directly the box or when there is backlighting it is harder to detect the QR. In addition, the material of the floor tiles also affect since they reflect the light and the QR. Sometimes it happens for the third picture that there is in the picture the QR as well as its reflection, making it harder to detect. A solution to this is to execute the system in a controlled environment with artificial light. However, it was not possible to arrange such environment so the hallway was the best option.

4.2.2 Movement skills experiments

These experiments have been conducted in order to check whether the functions implemented to control the movements of the robot work in an acceptable way, that is, without stepping out of the cells. Since the cells are quite big for the NAO robot, this gives it some space for error, which is needed due to the inherent error that this robot has when walking or turning. More precisely, the experimentation objectives are:

- Check that the error correction in the turns works as expected in a single movement.
- Check that the error correction in the turns works as expected in concatenated movements.
- Check that the error correction when walking straight works as expected in a single movement.
- Check that the error correction when walking straight works as expected in concatenated movements.
- Check that the whole performance is acceptable.

Checking both corrections for single movements has been done while developing the code since feedback was needed to do so. Therefore, once they worked in single movements, concatenated movements need to be tried to see how the error was accumulated and if that influenced the whole performance. For that, two experiments are conducted changing the environment size.

4.2.2.1 Movement experiment 1

In this experiment the environment includes two boxes, one blue and one red. The goal of the problem is to discover the colours of boxes located at (1,1) and (1,2). Figure 4.12 depicts the environment. The experiment has been designed so that

the NAO performs two *turn* actions and two *walk* actions. When the NAO locates itself in a cell to then get a colour, its position has to be in front of the box so that the picture taken has the QR. The objective of the experiment is to determine if the *turn* and *walk* functionalities work in a small environment and to check that they work in order for the NAO to be centered in fron of a box to get the QR in the picture. The expected result is that the NAO turns or walks with errors but the correction error algorithm compensates them enough so that the NAO never steps out of the cells. Also that the NAO is centered in front of a box to take the picture.

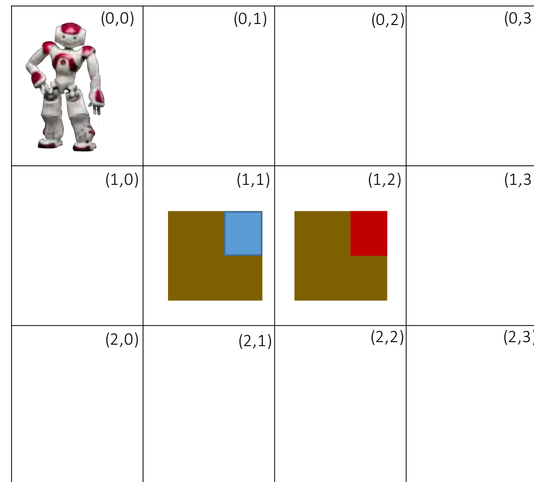


Figure 4.12: Experiment 1 environment diagram.

To solve the problem the NAO has performed 7 actions. The experiment has been executed six times. In two of them the NAO step out of cell (0,2) into (1,2) which made the execution unsuccessful. The other four times the execution was successful.

In the successful executions the NAO performance was acceptable. It traversed the board without stepping out of the cells and the goal of the problem was reached. The NAO did not walk straight but the correction implemented was enough to adjust the position and allow the NAO to have an acceptable trajectory, in this case almost perfect. The turn was corrected and performed with no remarkable error that affected the correct execution of the plan.

In the unsuccessful experiments the NAO deviate more than in the successful ones when walking. This is due to the floor tiles that are slippery for the NAO. It is also due to the way the NAO has been built as mentioned in chapter 3. Both are restrictions of this thesis problem. Figure 4.13 shows the deviation in the successful executions (1) and the one of the unsuccessful executions (2). Pictures (a) are the NAO position before the *walk* action and pictures (b) after the action. For (1) the deviation is not very large and the error propagates less than in (2) where the deviation is extremely large.

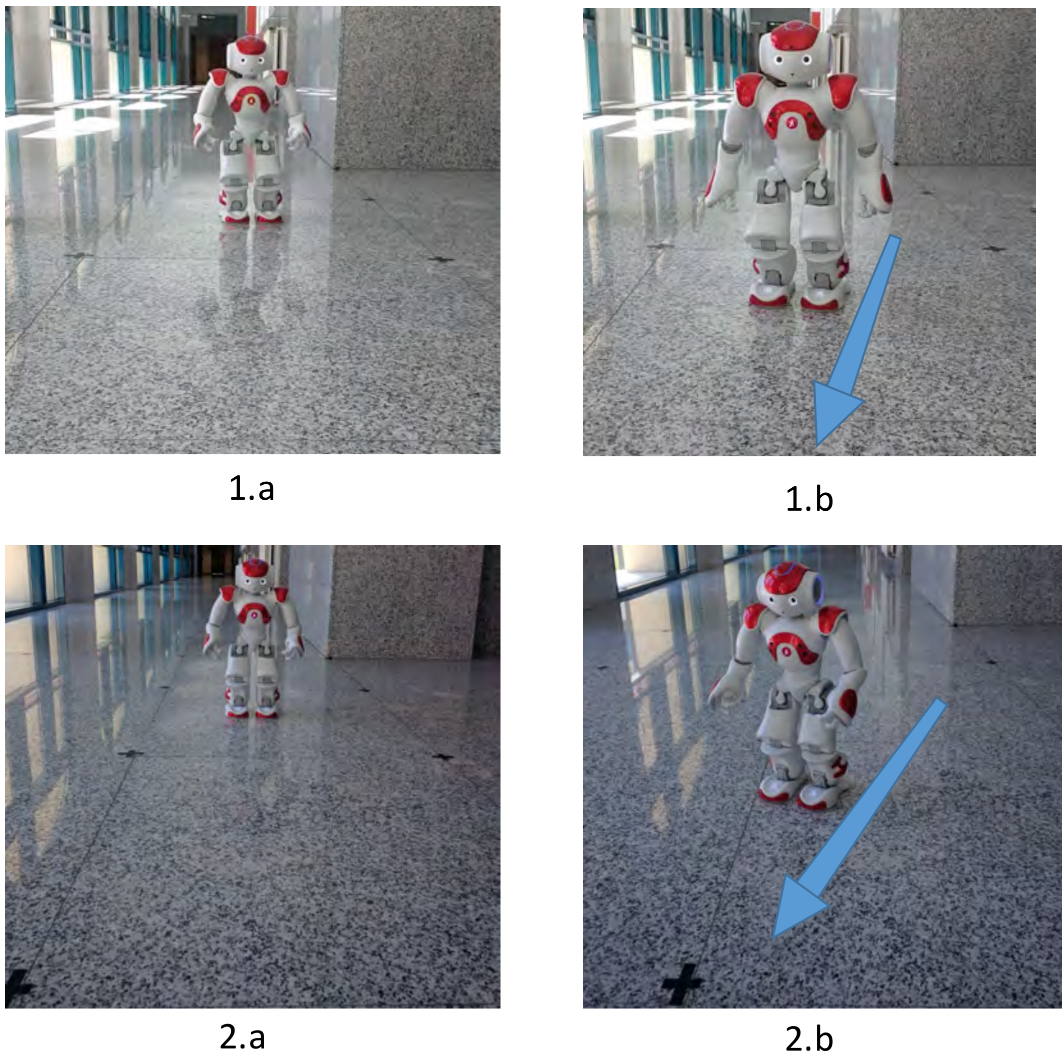


Figure 4.13: Walk action error in situations (1) and (2).

4.2.2.2 Movement experiment 2

For this experiment a larger environment has been used. It is adapted to the hallway's shape in which the experiments take place. Figure 4.14 depicts the environment. It has again two boxes, one blue and one red, and a NAO robot. The goal for this problem is to discover the box colours. The experiment has been designed so that the NAO has to concatenate several *walk* actions and some turns. The objectives of the experiment are the same as in experiment 1 but in a larger domain.

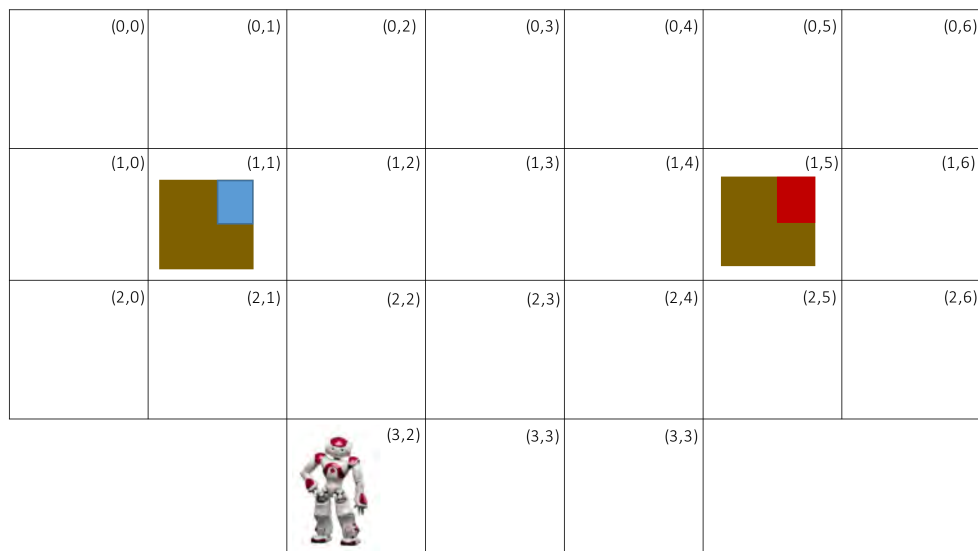


Figure 4.14: Experiment 2 environment diagram.

The experiment has been executed five times and the NAO executes 7 actions. Three of the executions were unsuccessful and two of them were successful.

In some of the unsuccessful executions the trajectory that the NAO follows is very close to the cell edges but does not step out of them. However, when it gets to the red box its position is not centered and the QR cannot be detected. Figure 4.15 shows the NAO position not centered in front of the box. This is due to the error when it turns 180 degrees in cell (1,2) from left to right. In others, it deviates and steps out of cell (1,4).

Figure 4.15: NAO position in front of the box in an unsuccessful execution.

The environment was modified as seen in Figure 4.16 to experiment with the same objectives as the previous ones. The experiment was executed five times and four of them were successful. This is because there was no 180 degrees turn, only 90 degrees turn which the NAO performs with more precision so less error is accumulated. In addition, factors mentioned before like the deviation in the *walk* action due to the slippery floor affect each execution in different ways.

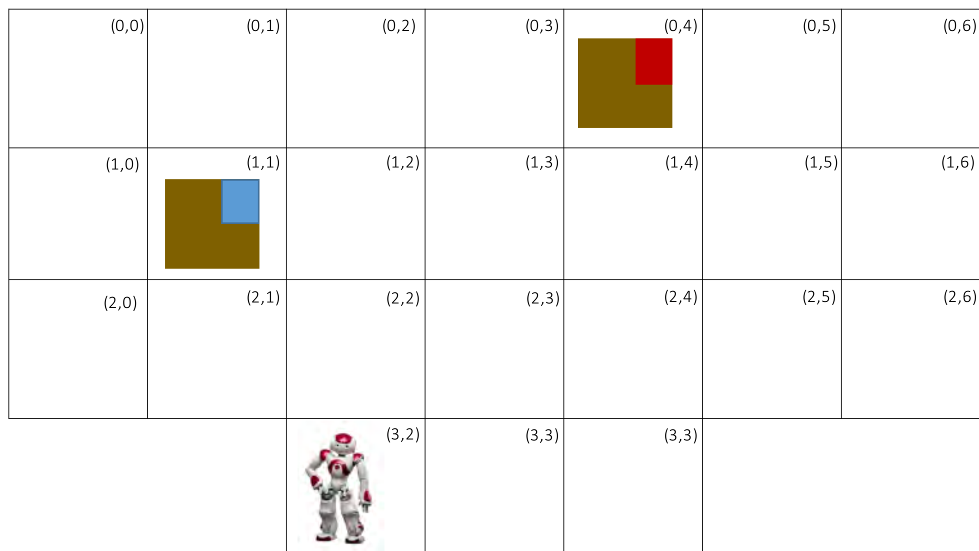


Figure 4.16: Experiment 2 modified diagram.

4.2.2.3 Movement experiments conclusions

The NAO performs successful executions without stepping out of the cells, keeping a reasonable trajectory corrected by the algorithms that allows to solve the problem.

Nonetheless, two conclusions can be drawn from this experimentation. Firstly, the error propagation due to the *walk* actions starts to be noticeable with 8 or more actions. It is more noticeable if there are several consecutive *walk* actions than when *turn* and *walk* actions are intercalated. This is because the correction algorithm corrects the final position of the NAO after the *walk* action but does not recover the distance lost in the deviation. However, in the turn it does save the distance but there is a threshold. The threshold is needed because of the NAO feet size that makes very small turns (around 10 degrees) hard to do. The error smaller than the threshold is accumulated and will be corrected later in the next turn, but if there is no turn then it is kept.

The second conclusion is that the the floor texture affects the NAO walk and turn as well as the way it is been built. The NAO is not a robot designed to perform walk with extreme precision. While this can be achieved with good results for small distances, as seen in this dissertation, it is a hard and tedious work. It can still be improved as it will be outlined in future work.

4.3 System experiments

Four experiments have been conducted to test the whole system once it was integrated.

4.3.1 Experiment 1: Problem 1

The problem proposed for this experiment simulates a small warehouse environment with one red box in location (1,2). The goal is to move the box to its destination location (1,3). However, the destination location is unknown and has

to be first discover by reading the QR. The problem has one NAO robot and one P3DX robot. Figure 4.17 shows the environment. The complexity of the problem is fairly easy because there is only one box and the domain is small so the space of states for the search is small. The objective of this experiment is to check that the system is able to solve a problem in the domain and that the system is correctly integrated and works as expected.

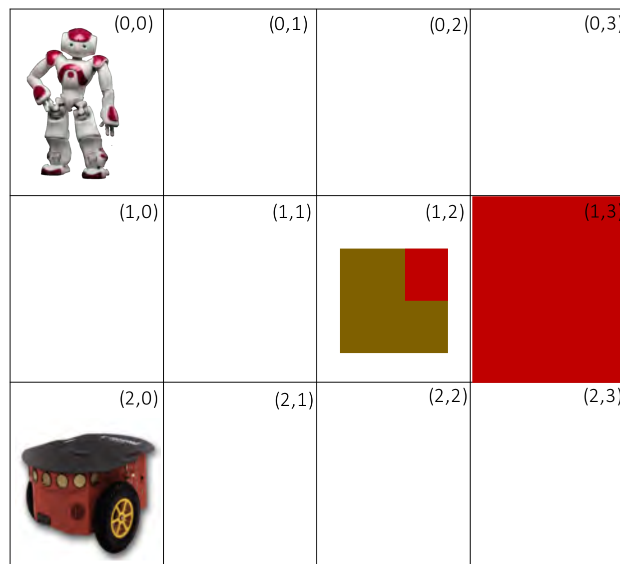


Figure 4.17: Problem 1 environment diagram.

The execution of the system was successful and it was able to solve the problem. Figure 4.18 shows the plan executed. With the completion of this experiment it is shown that the system integration is correct and that the system is able to solve problems with one box. During the experiment the NAO robot corrects its walking. The errors when walking are not very big in this experiment and the corrections are small. For the turn movement, the NAO turns less than 90 degrees and it is successfully corrected. During the experimentation the corrections help the NAO to keep a centered trajectory. The P3DX pushes the box straight without deviation and also traverses the environment with a centered trajectory.

The experiment was repeated several times. In some of them the NAO

robot was not able to get the QR. This might be due to the light conditions or the complexity of the picture to detect the QR. Given the distance between the NAO and the box, other elements surrounding the box are included in the picture. More specifically, the floor, which has black color, is also included in the picture and makes it harder to detect the QR. In other experiments, the P3DX robot did not push the box straight but the box still got to the destination cell without getting into other cells.

```
 0: MOVE F2-0F F2-1F RIGHT P3DX1
 1: MOVE F2-1F F2-2F RIGHT P3DX1
 2: MOVE F0-0F F0-1F RIGHT NAO1
 3: MOVE F0-1F F0-2F RIGHT NAO1
 4: TURN NAO1 RIGHT DOWN
 5: MOVE F0-2F F1-2F DOWN NAO1
 6: TURN NAO1 DOWN RIGHT
 7: GET-COLOUR NAO1 BOX1 F1-2F F1-3F RIGHT
 8: MOVE F2-2F F2-3F RIGHT P3DX1
 9: TURN P3DX1 RIGHT UP
10: TURN NAO1 RIGHT LEFT
11: TURN P3DX1 UP LEFT
12: MOVE F1-2F F1-1F LEFT NAO1
13: MOVE F2-3F F2-2F LEFT P3DX1
14: TURN P3DX1 LEFT UP
15: MOVE F2-2F F1-2F UP P3DX1
16: MOVE F1-2F F0-2F UP P3DX1
17: TURN P3DX1 UP RIGHT
18: MOVE F0-2F F0-3F RIGHT P3DX1
19: TURN P3DX1 RIGHT DOWN
20: PUSH-GOAL F0-3F F1-3F F2-3F DOWN BOX1 P3DX1 BLUE
```

Figure 4.18: Problem 1 plan.

After several executions of the experiment this video was obtained with a successful execution:

https://www.youtube.com/watch?v=QcoKGA2lm8o

4.3.1.1 Experiment 2: Problem 1.1

It is interesting to make the robots return to their original position after the task has been completed. For this, the goals of problem 1 have been modified to include that the robots must return to their original position, which are (0,0) and (2,0). This slightly increments the level of complexity because the number of actions increases and the NAO has to walk more. The objective of this experimentation is to test the system with a more complex problem with an interesting goal.

The experiment was repeated several times and most of them were successful. Figure 4.19 shows the plan executed with 23 actions. The NAO robot was able to return to its original position without deviating too much of its trajectory. The unsuccessful executions were due to the NAO being unable to detect the QR. A video of the execution can be found in:

https://www.youtube.com/watch?v=clanKaScye0

```
 0: MOVE F0-0F F0-1F RIGHT NAO1
 1: MOVE F0-1F F0-2F RIGHT NAO1
 2: MOVE F0-2F F0-3F RIGHT NAO1
 3: TURN NAO1 RIGHT DOWN
 4: GET-COLOUR NAO1 BOX1 F0-3F F1-3F DOWN
 5: TURN NAO1 DOWN LEFT
 6: MOVE F0-3F F0-2F LEFT NAO1
 7: MOVE F0-2F F0-1F LEFT NAO1
 8: MOVE F0-1F F0-0F LEFT NAO1
 9: MOVE F2-0F F2-1F RIGHT P3DX1
10: MOVE F2-1F F2-2F RIGHT P3DX1
11: TURN P3DX1 RIGHT UP
12: MOVE F2-2F F1-2F UP P3DX1
13: MOVE F1-2F F0-2F UP P3DX1
14: TURN P3DX1 UP RIGHT
15: MOVE F0-2F F0-3F RIGHT P3DX1
16: TURN P3DX1 RIGHT DOWN
17: PUSH-GOAL F0-3F F1-3F F2-3F DOWN BOX1 P3DX1 BLUE
18: TURN P3DX1 DOWN LEFT
19: MOVE F1-3F F1-2F LEFT P3DX1
20: MOVE F1-2F F1-1F LEFT P3DX1
21: MOVE F1-1F F1-0F LEFT P3DX1
22: TURN P3DX1 LEFT DOWN
23: MOVE F1-0F F2-0F DOWN P3DX1
```

Figure 4.19: Problem 1.1 plan.

4.3.2 Experiment 3: Problem 2

In this second experiment is presented a larger problem. In this one, there are two boxes, one blue and one red, a NAO robot and a P3DX robot. The goal of the problem is to get the blue box to the blue location and the red to the red one. Figure 4.20 shows the environment. The complexity of this problem increases because there are two boxes so the search space is bigger and the plan needs more actions. For this experiment the boxes were changed to bigger boxes created with the combination of two small ones, one on top of the other. The reason for this is that the QR will be at the NAO's height and it may be easier to detect it, given that QR detection was sometimes a problem in some executions in the previous experiment. The objective of the experiment is to check how the system works in a more complex problem than the one before with two boxes. It is also to check how the QR detection works with bigger boxes.

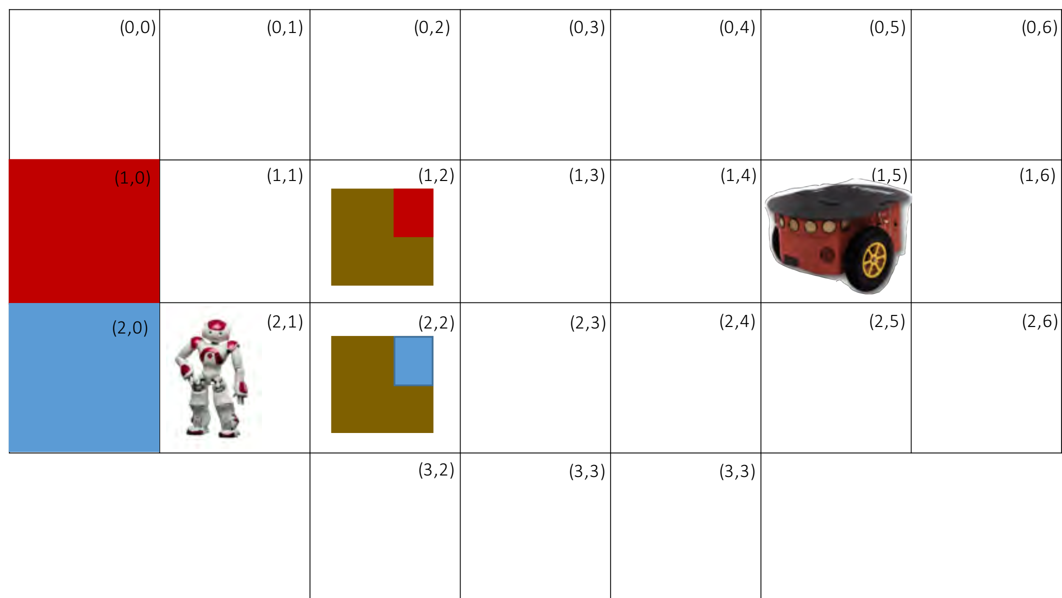


Figure 4.20: Problem 2 environment diagram.

Figure 4.21 shows the plan to solve this problem.

```
 0: MOVE F1-6F F1-5F LEFT P3DX1
 1: MOVE F2-1F F2-2F RIGHT NAO1
 2: MOVE F1-5F F1-4F LEFT P3DX1
 3: GET-COLOUR NAO1 BOX1 F2-2F F2-3F RIGHT
 4: TURN P3DX1 LEFT DOWN
 5: TURN NAO1 RIGHT UP
 6: MOVE F2-2F F1-2F UP NAO1
 7: MOVE F1-2F F0-2F UP NAO1
 8: MOVE F1-4F F2-4F DOWN P3DX1
 9: TURN P3DX1 DOWN LEFT
10: PUSH-NOT-GOAL F2-4F F2-3F F2-2F LEFT BOX1 P3DX1
11: PUSH-NOT-GOAL F2-3F F2-2F F2-1F LEFT BOX1 P3DX1
12: PUSH-GOAL F2-2F F2-1F F2-0F LEFT BOX1 P3DX1 BLUE
13: TURN NAO1 UP RIGHT
14: MOVE F0-2F F0-3F RIGHT NAO1
15: TURN NAO1 RIGHT DOWN
16: TURN P3DX1 LEFT RIGHT
17: MOVE F2-1F F2-2F RIGHT P3DX1
18: MOVE F2-2F F2-3F RIGHT P3DX1
19: MOVE F2-3F F2-4F RIGHT P3DX1
20: TURN P3DX1 RIGHT UP
21: MOVE F2-4F F1-4F UP P3DX1
22: TURN P3DX1 UP LEFT
23: GET-COLOUR NAO1 BOX0 F0-3F F1-3F DOWN
24: PUSH-NOT-GOAL F1-4F F1-3F F1-2F LEFT BOX0 P3DX1
25: PUSH-NOT-GOAL F1-3F F1-2F F1-1F LEFT BOX0 P3DX1
26: PUSH-GOAL F1-2F F1-1F F1-0F LEFT BOX0 P3DX1 RED
```

Figure 4.21: Problem 2 plan.

This experiment reported very interesting results. First, the QR detection improved considerably. It was detected in all the executions. The box size change was a perfectly reasonable approach since the NAO was prepared for any box size because it takes 3 pictures, each one at a different height. Figure 4.22 shows a picture taken from a big box (a) and another one from a small box (b). It can be seen how the QR is more centered in (a) than in (b) which makes it easier for the detector. The NAO QR detector is just like phones or other devices detectors where centering the QR in the image improves the detection performance. For the small boxes, the second picture that the NAO takes does not always include the whole QR. In the third picture it is included but it is not in the center or close to, it is usually at the bottom part of the picture and close to the corner. This explains why for the small boxes it is usually detected in the third picture but for the big boxes it is detected in the first or second.

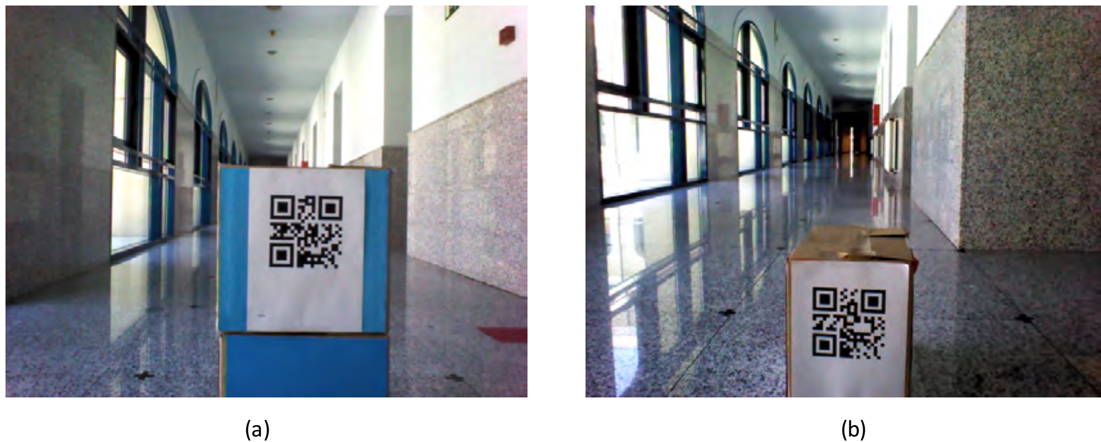


Figure 4.22: NAO pictures for different size boxes.

The experimentation gave away a limitation of the QR detection system that must be taken into account for future work. In this case, there was no problem since big boxes could be used. However, for problems that require just small boxes this limitation could be a problem. To solve it linear transformation algorithms could be applied iteratively to the image until the QR is detected.

Finally, another interesting aspect of this experiment is the collaboration between the robots. As it can be seen in the video, the NAO has to move so that

the P3DX can push the box. This shows the successful coordination between the robots.

The video with the execution can be found in:

`https://www.youtube.com/watch?v=_mL77D-2NWs`

4.3.3 Experiment 4: Problem 3

The problem proposed in this experimentation was designed to show the coordination and cooperation between both robots. Since the environment is not too big the robots have less room to move. Therefore, the NAO robot has to set aside in order to let the P3DX push the boxes. Then the P3DX waits for the NAO to finish its task and then moves to the location it needs to push the box. This fact makes the problem more complex for the planner algorithm. The objective of the experimentation is to test the system in an environment with this type of complexity and to show the coordination between the robots. Figure 4.23 shows the environment.

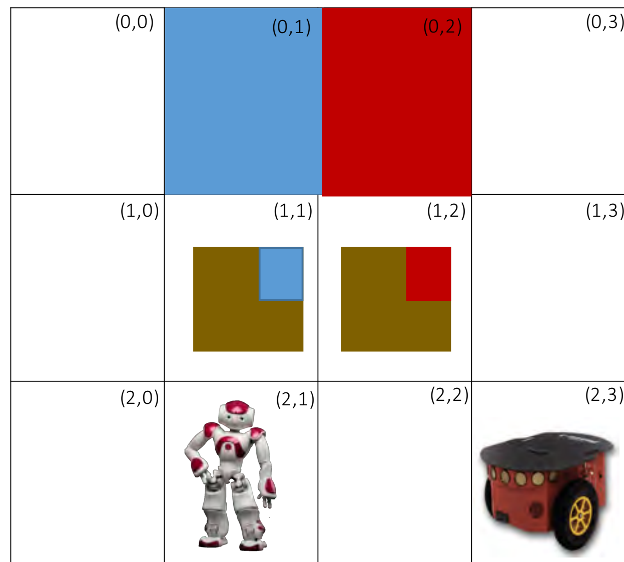


Figure 4.23: Problem 3 environment diagram.

Figure 4.24 shows the plan that solves the problem with 21 actions. Action 1 is a dummy action that the planner has included as a result of its search. However, this action does not provide anything to the plan and is due to the fact that the actions have no cost and there is no cost optimization.

```
 0: MOVE F2-3F F2-2F LEFT P3DX1
 1: TURN NAO1 RIGHT LEFT
 2: TURN NAO1 LEFT UP
 3: GET-COLOUR NAO1 BOX1 F2-1F F1-1F UP
 4: TURN NAO1 UP LEFT
 5: MOVE F2-1F F2-0F LEFT NAO1
 6: MOVE F2-2F F2-1F LEFT P3DX1
 7: TURN P3DX1 LEFT UP
 8: PUSH-GOAL F2-1F F1-1F F0-1F UP BOX1 P3DX1 BLUE
 9: TURN P3DX1 UP DOWN
10: TURN NAO1 LEFT RIGHT
11: MOVE F2-0F F2-1F RIGHT NAO1
12: MOVE F2-1F F2-2F RIGHT NAO1
13: MOVE F1-1F F2-1F DOWN P3DX1
14: TURN P3DX1 DOWN RIGHT
15: TURN NAO1 RIGHT UP
16: GET-COLOUR NAO1 BOX0 F2-2F F1-2F UP
17: TURN NAO1 UP RIGHT
18: MOVE F2-2F F2-3F RIGHT NAO1
19: MOVE F2-1F F2-2F RIGHT P3DX1
20: TURN P3DX1 RIGHT UP
21: PUSH-GOAL F2-2F F1-2F F0-2F UP BOX0 P3DX1 RED
```

Figure 4.24: Problem 3 plan.

The result of the experiment was positive because the task was solve. In some executions the NAO robot greatly deviated while walking but most of them were performed correctly. Again, in this case it was because of the floor tiles. The

execution can be seen in this video:

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

As seen in the video, the NAO robot sets aside after detecting the colour of the first box so that the P3DX can push the blue box. Then, the P3DX stays in location (1,1) without being in the way of the NAO robot that has to go to location (2,2). Then, again, once the box colour is discovered the NAO moves so that the P3DX can push the box.

4.3.4 System experimentation conclusions

After conducting the system experimentation, it can be concluded that the system is robust and solves problems correctly with some conditions:

- The QR has to be located at the NAO's height so that it is centered in the picture. If this is fulfilled the lighting conditions do not affect that much the detection because once the QR is centered the detection is very easy, as seen in the experiments.
- The floor tiles should not be slippery for the NAO robot so that it has the least deviation possible.

If any of the conditions is not fulfilled then the system performance is reduced as well as its success ratio. This analysis has outlined some interesting future work directions that would improve the system and help it evolve. This future work is described in section 6.3.

Chapter 5

Project management

In this chapter it is described how the project has been organized and planned with a development model and its phases. These phases are described in section 5.1 and the tasks for each of them with their duration are described in section 5.2. Finally, a list with the resources needed will be provided to calculate the final budget for this dissertation.

5.1 Phases description

The development process of the project will follow the Waterfall model. This model was formally described in 1970 by Winston W. Royce [33]. It is a sequential model where a phase is not started until the previous one has been completed. This could be adapted very well to this project since it has several components very differentiated and independent. A Spiral model [34], which was also considered, would require to have a prototype in the first iteration. However, obtaining a functional prototype of this system would take 4 to 6, given its size and number of components. Instead, a Waterfall model organized the project in independent phases as seen in Figure 5.1.

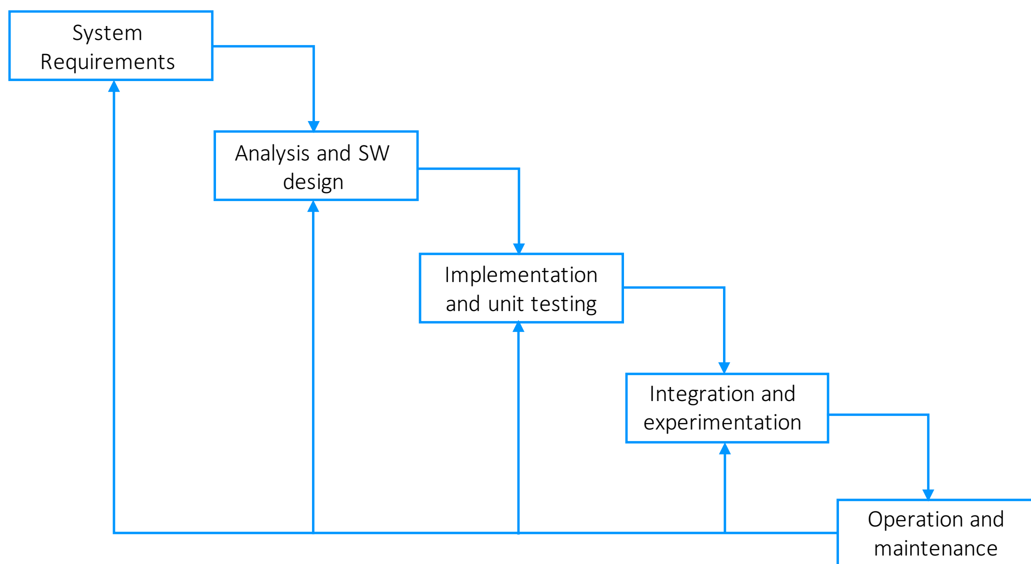


Figure 5.1: Waterfall development process.

The phases are here described:

- System requirements. This phase involves analysing the problem statement to determine the features and characteristics that the system must have. For that, use cases are defined to obtain specific requirements and functionality. In addition, and once the problem demands are clear, the project is organized, proposing a development process model and designing the tasks in each phase.

- Analysis and software design. In this stage the software that will be used is studied to become familiar with it and determine the best options. Then, using the previous requirements and functionality that the system must have the components are designed.

- Implementation and unit tests. This is the phase were the code is developed. Since there are three very differentiated components in the system (NAO Execution module, P3DX Execution module and planning subsytem) each of them will be developed independently and tested on its own.

- System integration and experimentation. In this phase the whole system is

built by integrating all the components. Then, experimentation is conducted to determine the correctness of the product.

- Operation and maintenance. This phase would involve the tasks needed to install, deploy, support and maintain the system. In this case, the documentation has been finished.

5.2 Planning

The different tasks required for each phase have been broken down in table 5.1. In addition, the tasks have also been depicted in chronological order with a Gantt diagram in Figure 5.2.

The project started on the 30th of November of 2015 and was finished the 21st of June of 2016, being in total 6 months. Only week days are counted excluding weekend days and with an average work of 3 hours a day. All tasks include in their duration the creation of the corresponding documentation and follow-up meetings with the tutors.

Task	Duration	Start date	End date
System requirements	12 days	30/11/2015	15/12/2015
Functionality definition	3 days	30/11/2015	02/12/2015
Use Cases definition	5 days	03/12/2015	09/12/2015
Requirements definition	4 days	10/12/2015	15/12/2015
Analysis and software design	37 days	16/12/2015	03/02/2016
Planning techniques and PDDL Analysis	3 days	16/12/2015	18/12/2015
NAO development environment (JNAOqi,Webots, Choregraph) Analysis	6 days	21/12/2015	28/12/2015
Computer Vision techniques Analysis	5 days	29/12/2015	04/01/2016
P3DX development environment (ROS) Analysis	4 days	05/01/2016	08/01/2016

PELEA Analysis	4 days	11/01/2016	14/01/2016
Domain design	7 days	15/01/2016	25/01/2016
Modules design	8 days	26/01/2016	04/02/2016
Implementation and unit tests	46 days	05/02/2016	08/04/2016
Domain and problems PDDL implementation	6 days	05/02/2016	12/02/2016
NAOExecution module implementation	31 days	15/02/2016	28/03/2016
NAOExecution module tests	6 days	29/03/2016	05/04/2016
P3DXExecution module experimentation	3 days	06/04/2016	08/04/2016
System integration and experimentation	32 days	11/04/2016	24/05/2016
Integration NAOExec-PELEA	7 days	11/04/2016	19/04/2016
Integration NAOExec-PELEA test	2 days	20/04/2016	21/04/2016
Integration P3DXExec-PELEA	3 days	22/04/2016	26/04/2016
Integration P3DXExec-PELEA test	2 days	27/04/2016	28/04/2016
Integration all components test	5 days	29/04/2016	05/05/2016
System experimentation	13 days	06/05/2016	24/05/2016
Operation and documentation	20 days	25/05/2016	21/06/2016
Use and documentation	16 days	25/05/2016	15/06/2016
Presentation	4 days	16/06/2016	21/06/2016

Table 5.1: Date and task definition of the project development.

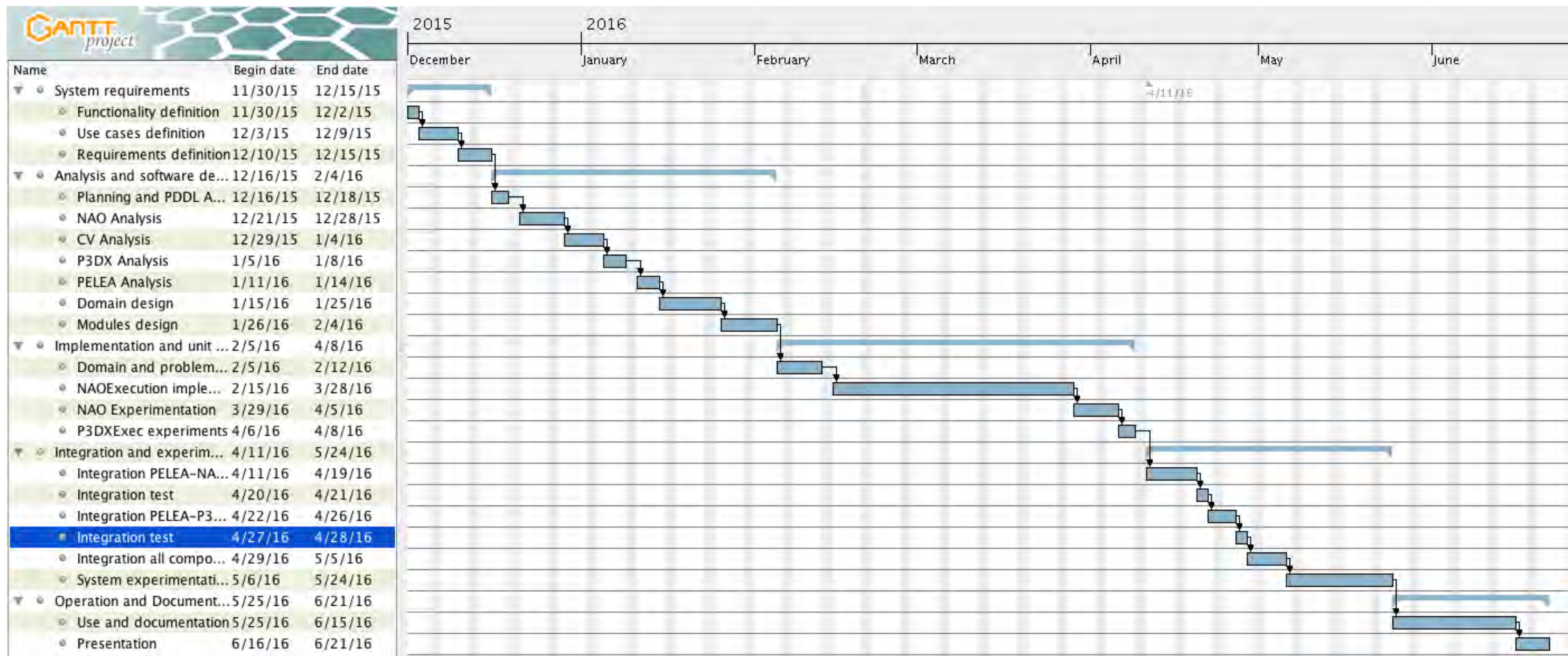


Figure 5.2: Project Gantt Chart. Dates are in MM/DD/YYYY format.

5.3 Budgeting

In this section, the project costs are broken down into detailed categories to estimate the total budget for it.

In table 5.2 the estimated costs for human resources are presented. It is added to the base salary the Social Security cost, which is a 30% of the salary [1]. The base salary have been obtained from the *Boletin Oficial del Estado from the 22nd of January of 2014, Number 19* [2] since it is the last one specified for this sector.

Position	Monthly base salary	Number of months	Final cost	Final cost with SS
Programmer	1000€	1.96	1966€	2585€
Tester	1000€	0.83	833€	1082€
Analyst	1333€	1.33	1777€	2310€
Project Director	2083€	1.3	2708€	3520€
TOTAL				9497€

Table 5.2: Estimated staff direct costs.

In table 5.3 there are listed costs related to the equipment used. Each of them has a unitary cost (C) and depreciation period in months (D) from which the chargeable cost or amortization (A) will be calculated for the period that has been used (P):

$$A = (P/D) * C$$

[1] http://www.seg-social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecotiza36537/index.htm, last visit 26 May 2016.

[2] https://www.boe.es/boe/dias/2014/01/22/pdfs/BOE-A-2014-639.pdf, last visit 26 May 2016.

Concept	Unitary cost	Depreciation period	Chargeable cost
Macbook Air 13.1”	1099 €	48 months	137.38 €
Netbook Samsung NC210	349 €	48 months	43.6 €
NAO Robot	12,000 €	72 months	999 €
P3DX Robot	5,777 €	72 months	481 €
Router	20 €	72 months	1.66 €
Wires	20 €	72 months	1.66 €
Fungibles	20 €	48 months	2.5 €
TOTAL			1,666.8 €

Table 5.3: Estimated equipment direct cost for a use of 6 months.

Assuming indirect cost to be a 20% of the direct costs then these are 2,232.76 €. The final budget comprises the staff, the equipment cost and the indirect costs and results in **13,396.56 €** (thirteen thousand three hundred and ninety-six euros with fifty-six cents).

Chapter 6

Conclusions

This chapter provides an overview of the conclusions after the completion of this project. First the general conclusions after the completion of this work are presented. Second, the conclusions concerning to the objectives proposed in the first chapter are presented. Finally some future work directions are described.

6.1 General conclusions

The main objective of this dissertation consist on building a multi-agent system that uses Automated Planning and Computer Vision in warehouse environments to solve a given task. The task that the system would carry out is warehouse organization with sensing, control and deliberation involved. The environment model is similar to a small warehouse.

The work has resulted in a system that integrates Automated Planning and Computer Vision to solve tasks using autonomous and heterogeneous agents. The deliberation process for the robots has been implemented with Automated Planning; for the control part a controller has been developed and for the sensing the robots were used to obtain the information of the real environment. Later, the controller process the information. This set of stages accomplishes and verifies the main objective.

Developing this kind of system faces several challenges. Working with robots

that interact with the real environment and gather information from it is complex. The sensing information is not always precise and the error propagates along the system. Computer vision algorithms are extremely sensitive to the environmental conditions, such as light, which makes it necessary to often use controlled environments. Finally, heterogeneous multi-agent system allows collaboration and therefore solving more complex tasks. However, it also requires the coordination and control of two different agents, in this case, with their different specifications and requirements. This has been accomplished by using PELEA. For this system, the NAO is required to perform very precise movements along the board: walk straight and turn 90 degrees. It may seem as something very basic but the NAO robot is not specifically designed for those tasks and the way it is built does not facilitate it. Consequently, the error correction was one of the main challenges.

A wide variety of techniques have been evaluated and tried during the design and development phases. Several Computer Vision techniques have been reviewed and used. From a first proposal where boxes would be coloured to a QR detection system, histograms and blobs recognition mechanisms have been also considered and implemented, but discarded. Also, several ideas for the error correction algorithm were proposed and developed until tests demonstrated they were not good enough. Among them, initially, the NAO error correction was done using the gyroscope inside the NAO to measure its velocity at every instant to determine the real distance travelled. Finally, the internal reference system in angles was used.

This work has required knowledge in Automated Planning, Computer Vision, object-oriented programming, robotics software and software engineering. Developing this dissertation has supposed acquiring some of this knowledge that was new or expand other that had already been acquired during the four years of degree.

The result has been satisfactory and acceptable, with some future work outlined that time did not allow to implement. In addition, I have collaborated with my advisors in a paper they have published in the International Joint Conference on Artificial Intelligence (IJCAI) 2016 Workshop on Autonomous Mobile Service

Robots. This dissertation is based in the work presented in that paper which proposes a problem that faces more complex challenges in the same environment with an evolved and more complex solution.

6.2 Conclusions concerning the objectives

In this section, a conclusion for each of the objectives proposed in section 1.3 will be presented:

- **Carry out an initial study of the technologies that will be used to become familiar with them and learn how to use them.** This objective has been completed and as a result of this, it has been necessary to become familiar and learn new technologies and paradigms such as: hybrid control systems, NAO software, Computer Vision algorithms and Automated Planning. This is an always enriching experience that has been crucial in order to develop the solution.

- **Design a model to represent the problems environment based on the Sokoban and implement it.** To reach this objective a study of the problem and the modifications needed was carried out to modify a previous domain in order to include new functionality. For the implementation, the PDDL language was used because is the languange used by the most common algorithms. Everything was tested with a planner to validate the design.

- **Develop the NAO robot functionality that must enable the robot to: move and turn.** This objective was achieved using the NAOqi API and designing and implementing error correction algorithms. This has shown to be a tedious task that requires working with the robot while developing it. It demonstrates that robot interaction with the real world and sensing is more complex that it seems and it is still not accurate enough.

- **Develop a module that uses Computer Vision techniques to identify the colours of the boxes and integrate it with the NAO func-**

tionality. This objective was reached with a QR detector and a QR reader. This was, in the end, a more reasonable approach since in a warehouse a QR can contain a lot of information from the box and not just only one characteristic, such as the colour. Achieving this objective involved testing several techniques in Computer Vision, getting a general idea of this field, and demonstrated how light and the environment can affect the robot's vision.

- **Coordinate the NAO and P3DX robots to solve tasks in the domain by using a hybrid control system.** To achieve this objective, PELEA has been used for the control system. This has required to integrate the developed NAO Execution module with the modules in PELEA.

- **Evaluate the correctness of the system with experiments.** This goal has been achieved through a planned and extensive experimentation. It has allowed to not only test the system but also outline future work that could not be included in this dissertation due to time and workload restrictions.

- **Generate the corresponding documentation.** To fulfill this objective, this document has been created with detailed descriptions of every aspect of the project.

6.3 Future work

On the developed system it can be added various improvements that would make its operation richer and would add new features to its execution. Some of these improvements are the following:

- Improve the robot NAO precision when walking. This work would include modifying the existing correction algorithm to include external guides. These guides would help the NAO robot determine its final position after moving with regard to external references rather than its internal reference

system. These guides could be for instance floor tiles lines or calculating tile centers. The improvement of this would be that these guides are external from the NAO robot and therefore always fixed and more reliable than its internal reference system.

- Improve QR detection. This work would be carried out to deal with the limitations found with QR detectors. First, finding algorithms or image processing techniques to eliminate the light effects in the image. For instance, highlight the QR black and white areas or modify the colour spaces and other characteristics of the image. Second, develop mechanisms to avoid problems with the location of the QR in the image. For example, apply to the image linear transformation algorithms to improve the detection in spite of the conditions of the picture.

- Increase the number or robots in the system. This work would involve adding an Execution module per robot included in the environment. Both new coordinators and cargo robots can be added.

- Create more complex problems. Experiments can be conducted to test the performance constraints in the system when the problem complexity is increased. For instance, use bigger environments, environments with obstacles or more boxes.

- Have the robots execute actions in parallel. This would require to deploy the PELEA technology that supports parallelization and the use of a planner that generates parallel plans.

- Include action cost. Modify the domain in PDDL to assign to each action a cost This way, the planner can plan in terms of cost optimization and dummy actions are avoided.

Appendix A

Installation

In this section, it will be explained the installation process for each of the libraries and dependencies needed for this project. The project has been developed and tested using two operating systems: OS X EL Capitan 10.11.4 and Ubuntu 12.04 LTS.

A.1 JDK

The Java Development Kit (JDK) last version can be downloaded from this link `http://www.oracle.com/technetwork/es/java/javase/downloads/index.html`. It is better to install the JDK and not only the Java SE because the JDK includes Java SE and also very useful developing tools. Once installed, the installer will open a pop-up browser window to check if it has been installed correctly.

For the installation in Ubuntu execute the following commands in the terminal:

```
$ sudo apt-add-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
```

To develop with Java, the Eclipse IDE has been used. The installer with the Eclipse last version (Mars) can be downloaded in this link `https://eclipse.org/downloads/`.

A.2 JNAOqi and NAO environment

The Java NAOqi (JNAOqi) library can be downloaded from the Aldebaran Community website `https://community.aldebaran.com/en` under Resources/Software. However, an account needs to be first created to access this content.

The JNAOqi library is a .jar that can be included in the Eclipse project in the following way: right-click on the project, then *build path*, then *add external archive*, then select the corresponding JNAOqi jar.

The latest version for Choregraphe and Webots can also be found in the Aldebaran Community website. Choregraphe can be downloaded if a graphical interface to control the robot is required. Webots is a robot simulator that can be used for the NAO robot .

A.3 ZXing

To use the ZXing library in the Java project in Eclipse the javase-x.x.jar and the zxing-x.x.jar, where the "x" stands for the version, are needed. They can be obtained here `http://repo1.maven.org/maven2/com/google/zxing/`.

To include them in the build path of the project do right-click on the project, then *build path*, then *add external archive*, then select the both .jar.

A.4 ROS

To install ROS Hydro in Ubuntu there are several steps to follow. If any other distribution is required, it can be selected from `http://wiki.ros.org/Distributions` and check the installation instructions provided.

For Hydro, first it is required to setup the source.list with this command:

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

And set up a key:

```
$ wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key
-O - | sudo apt-key add -
```

For the installation it can be used apt-get. For that, check first that it is up to date with:

```
$ sudo apt-get update
```

Then, install ROS Hydro (a prompt about hddtemp can pop-up to install it, but is not necessary for this project):

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

After it has finished, initialize rosdep to install dependencies when compiling:

```
$ sudo rosdep init
$ rosdep update
```

Configure the ROS environment variables so that they are added automatically every time a terminal is opened:

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

All these instructions can be found in the following link as well, `http://wiki.ros.org/hydro/Installation/Ubuntu`.

To manage the P3DX it is also needed the P2OS Driver. To install it, run the following command:

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

A.5 PELEA

To use PELEA it is needed: the PELEA modules, a modified MDPSim to simulate the environment and check states, the planner, in this case MetricFF and the domain and problems in PDDL. Everything can be found in `http://www.plg.inf.uc3m.es/pelea/tutorial-installation.php`.

Then, PELEA is used in our Java project as any other library in .jar format. It has to be compiled first to .jar. This can be done in Eclipse following these steps:

1. Create a new project in Eclipse with all the content downloaded.
2. Add any necessary library in the build path (as explained in previous sections in this appendix)
3. Set org.pleame.main as the main class in the proejct properties.
4. Compile the project to generate a .jar (Click on *Export*, then *Build runnable JAR*).

For MDPSim go to its directory and execute *./configure* and then run the command *make*.

For MetricFF go to its directory and run *make*.

Then, to run PELEA execute the following commands, each one in a different terminal, to run each module:

```
$ java -jar ./dist/Warehouse.jar -c ./config/FF-1.xml -n DS1
$ java -jar ./dist/Warehouse.jar -c ./config/FF-1.xml -n M1
$ java -jar ./dist/Warehouse.jar -c ./config/FF-1.xml -n EXE
```

DS1 is the name of the Decision Support module, M1 is the name of the Monitoring and EXE the name of the Execution module. An example of a configuration file can be found in Appendix C. This file contains parameters to configure the execution of PELEA.

Appendix B

User guide

In this section is explained how to use the system. First, the problem must be modelled in PDDL. As an example, the Problem 1 used in the experimentation is provided in Appendix C.

Then the configuration file has to be created to indicate the number of modules and their parameters to PELEA. It is composed of terms with a value and a name. The configuration file for Problem 1 can be found in Appendix C. It is divided into nodes, one for each module and a generic one at the beginning of the file. The important terms that must be modified when changing the problem are explained here:

- *IP* and *PORT* in the first group of terms indicates the IP address of the Monitoring module. It is necessary in order for the other modules to connect.
- *DOMAIN* and *PROBLEM* indicate the path to the domain and problem files.
- The *DISTANCE*, *MOVESPEED*, *TURNSPEED*, *PUSHSPEED* and *BOXSIZE* are parameters to configure the P3DX robot actions. The *DISTANCE* is the cell size and *BOXSIZE* is the box size. The other ones configure the speed of the P3DX when doing the actions.
- *INITEXECUTION* indicates if the node starts the execution, takes value *true*, or not, takes value $false$.

- *IP* and *PORT* in the NAO module indicate the IP address and port of the NAO in order to connect to it through WiFi.

- Finally, the term *NETWORK* in the Monitoring module shows the number of expected modules connected. The value *value="1,2,0,0,0,0,1,0"* indicates thate there is 1 Decisin Support, 2 Execution modules and 1 Monitoring.

After the problem and configuration files are done, the environment can be prepared (boxes, robots, etc). The Wi-Fi network must work and the robots must be turned on. Then, the system can be launched from the terminal with a script start.sh. Indicate the problem number (for Problem 1, use 1)as the command option:

```
$ ./start.sh -p 1
```

This script starts the execution of the system and the user does not have to intervene any more.

Appendix C

PDDL source code and scripts

Domain in PDDL designed and used in this dissertation.

```
(define (domain sokoban-colours-turn)
(:requirements :typing :fluents)
(:types LOC DIR thing - object
        BOX robot - thing
        P3DX NAO - robot
        colour)
(:predicates
             (at ?t - thing ?l - LOC)
             (robot-dir ?robot - robot ?d - DIR)
             (at-goal ?b - BOX)
             (is-goal ?l - LOC)
             (adjacent ?l1 - LOC ?l2 - LOC ?d - DIR)
             (clear ?l - LOC)
             (colour-cell ?l - LOC ?c - colour)
             (has-colour ?b - BOX ?c - colour)
             (known-colour ?b - BOX)
             (unknown-colour ?b - BOX)
             (no-storage ?b - BOX))
```

```
(:action move
    :parameters (?from - LOC ?to - LOC ?dir - DIR ?robot - robot)
    :precondition   (and (clear ?to)
                         (at ?robot ?from)
                         (adjacent ?from ?to ?dir)
                         (robot-dir ?robot ?dir)
                    )
    :effect         (and (at ?robot ?to)
                         (not (at ?robot ?from))
                         (clear ?from)
                         (not (clear ?to))
                    )
)
(:action push-goal
    :parameters  (?rloc - LOC ?bloc - LOC ?floc - LOC ?dir - DIR
                 ?b - BOX ?robot - P3DX ?c - colour)
    :precondition (and  (at ?robot ?rloc)
                        (at ?b ?bloc)
                        (clear ?floc)
                        (robot-dir ?robot ?dir)
                        (adjacent ?rloc ?bloc ?dir)
                        (adjacent ?bloc ?floc ?dir)
                        (is-goal ?floc)
                        (known-colour ?b)
                        (has-colour ?b ?c)
                        (colour-cell ?floc ?c)
                        (no-storage ?b)
              )
    :effect      (and   (at ?robot ?bloc)
                        (at ?b ?floc)
```

```
                        (clear ?rloc)
                        (at-goal ?b)
                        (not (at ?robot ?rloc))
                        (not (at ?b ?bloc))
                        (not (clear ?floc))
                        (not (no-storage ?b))
                 )
    )
(:action push-not-goal
    :parameters  (?rloc - LOC ?bloc - LOC ?floc - LOC ?dir - DIR
                 ?b - BOX ?robot - P3DX)
    :precondition (and  (at ?robot ?rloc)
                        (at ?b ?bloc)
                        (clear ?floc)
                        (robot-dir ?robot ?dir)
                        (adjacent ?rloc ?bloc ?dir)
                        (adjacent ?bloc ?floc ?dir)
                        (known-colour ?b) (no-storage ?b)
                  )
    :effect      (and   (at ?robot ?bloc)
                        (at ?b ?floc)
                        (clear ?rloc)
                        (not (at ?robot ?rloc))
                        (not (at ?b ?bloc))
                        (not (clear ?floc))
                )
)
(:action get-colour
    :parameters  (?robot - NAO ?b - BOX ?rloc - LOC ?bloc - LOC ?dir - DIR
    :precondition (and  (at ?robot ?rloc)
```

```
                        (at ?b ?bloc)
                        (adjacent ?rloc ?bloc ?dir)
                        (robot-dir ?robot ?dir)
                        (unknown-colour ?b)
                  )
    :effect       (and  (known-colour ?b)
                     (not (unknown-colour ?b))
    )
)
(:action turn
    :parameters (?robot - robot ?dr - DIR ?db - DIR)
    :precondition (and (robot-dir ?robot ?dr))
    :effect       (and (robot-dir ?robot ?db)
                       (not (robot-dir ?robot ?dr))
                   )
)
)
```

PDDL file for Problem 1 used in the experimentation:

```
(define (problem sokoban1)
(:domain sokoban-colours-turn)
(:objects
        P3DX1 - P3DX
        NAO1 - NAO
        box1 - BOX
        up down left right - DIR
        f0-0f f0-1f f0-2f f0-3f
        f1-0f f1-1f f1-2f f1-3f
        f2-0f f2-1f f2-2f f2-3f - LOC
        red blue grey - colour
```

```
)
(:init
    (adjacent f0-0f f0-1f right)
    (adjacent f0-0f f1-0f down)
    (adjacent f0-1f f0-0f left)
    (adjacent f0-1f f0-2f right)
    (adjacent f0-1f f1-1f down)
    (adjacent f0-2f f0-1f left)
    (adjacent f0-2f f0-3f right)
    (adjacent f0-2f f1-2f down)
    (adjacent f0-3f f0-2f left)
    (adjacent f0-3f f1-3f down)
    (adjacent f1-0f f1-1f right)
    (adjacent f1-0f f0-0f up)
    (adjacent f1-0f f2-0f down)
    (adjacent f1-1f f1-0f left)
    (adjacent f1-1f f1-2f right)
    (adjacent f1-1f f0-1f up)
    (adjacent f1-1f f2-1f down)
    (adjacent f1-2f f1-1f left)
    (adjacent f1-2f f1-3f right)
    (adjacent f1-2f f0-2f up)
    (adjacent f1-2f f2-2f down)
    (adjacent f1-3f f1-2f left)
    (adjacent f1-3f f2-3f down)
    (adjacent f1-3f f0-3f up)
    (adjacent f2-0f f2-1f right)
    (adjacent f2-0f f1-0f up)
    (adjacent f2-1f f2-0f left)
    (adjacent f2-1f f2-2f right)
```

```
(adjacent f2-1f f1-1f up)
(adjacent f2-2f f2-1f left)
(adjacent f2-2f f2-3f right)
(adjacent f2-2f f1-2f up)
(adjacent f2-3f f2-2f left)
(adjacent f2-3f f1-3f up)
(at box1 f1-3f)
(no-storage box1)
(at P3DX1 f2-0f)
(at NAO1 f0-0f)
(robot-dir P3DX1 right)
(robot-dir NAO1 right)
(clear f0-1f)
(clear f0-2f)
(clear f0-3f)
(clear f1-0f)
(clear f1-1f)
(clear f1-2f)
(clear f2-1f)
(clear f2-2f)
(clear f2-3f)
(is-goal f2-3f)
(colour-cell f0-0f grey)
(colour-cell f0-1f grey)
(colour-cell f0-2f grey)
(colour-cell f0-3f grey)
(colour-cell f1-0f grey)
(colour-cell f1-1f grey)
(colour-cell f1-2f grey)
(colour-cell f1-3f grey)
```

```
    (colour-cell f2-0f grey)
    (colour-cell f2-1f grey)
    (colour-cell f2-2f grey)
    (colour-cell f2-3f blue)
    (has-colour box1 blue)
    (unknown-colour box1)
)
(:goal
    (and
        (at-goal box0)
        (at-goal box1)
    )
)
)
```

Configuration file for Problem 1 used in the experimentation:

```
<?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=";" name="SEPARATOR"/>
<term value="./problems/warehouse/domain.pddl" name="DOMAIN"/>
<term value="./problems/warehouse/p11.pddl" name="PROBLEM"/>
<term value="./experiment/" name="OUTPUT_DIR"/>
<term value="./temp/" name="TEMP_DIR"/>
<term value="FF" name="NAME"/>
<term value="1" name="ROUNDS"/>
<term value="PARTIAL" name="STATE"/>
<term value="ON" name="DEBUG"/>
<nodes>
<node type="planner" id="FD">
<term value="FD" name="PLANNER_NAME"/>
<term value="./planners/ff/" name="PLANNER_DIR"/>
<term value="org.pelea.planners.MetricFF" name="PLANNER_CLASS"/>
<term value="0" name="PLANNER_MODE"/>
<term value="1000" name="MAX_PLANNING_TIME"/>
</node>
<node type="planner" id="RFD">
<term value="FDR" name="PLANNER_NAME"/>
<term value="./planners/ff/" name="PLANNER_DIR"/>
<term value="org.pelea.planners.MetricFF" name="PLANNER_CLASS"/>
<term value="1" name="PLANNER_MODE"/>
<term value="1000" name="MAX_PLANNING_TIME"/>
</node>
```

```
<node type="decisionSupport" id="DS1">
<term value="1" name="type"/>
<term value="2" name="mode"/>
<term value=
"org.pelea.core.module.basic.DecissionSupportBasic" name="CLASS"/>
<term value="FD,RFD" name="PLANNERS"/>
<term value="false" name="DS_ALLWAYS_REPLANN"/>
<term value="true" name="DELETE_TEMP_FILES" />
</node>
<node type="execution" id="P3DX1">
<term value="2" name="type"/>
<term value="2" name="mode"/>
<term value="org.pelea.core.module.robotics.P3DX" name="CLASS"/>
<term value="127.0.0.1" name="IP"/>
<term value="11311" name="PORT"/>
<term value="false" name="INITEXECUTION"/>
<term value="0.9" name="DISTANCE"/>
<term value="0.3" name="MOVESPEED"/>
<term value="0.15" name="TURNSPEED"/>
<term value="0.3" name="PUSHSPEED"/>
<term value="0.225" name="BOXSIZE"/>
</node>
<node type="execution" id="NAO1">
<term value="2" name="type"/>
<term value="2" name="mode"/>
<term value="org.pelea.module.robotics.NAOExecution" name="CLASS"/>
<term value="192.168.1.179" name="IP"/>
<term value="9559" name="PORT"/>
<term value="true" name="INITEXECUTION"/>
</node>
```

```
<node type="monitoring" id="M1">
<term value="7" name="type"/>
<term value="1" name="mode"/>
<term value="org.pelea.core.module.basic.MonitoringBasic"
name="CLASS"/>
<term value="false" name="VALIDATE_STATE"/>
<term value="true" name="EXECUTION_MODE"/>
<term value="1,2,0,0,0,0,1,0" name="NETWORK"/>
<term value="false" name="INITEXECUTION"/>
<term value="robot,nao,p3dx" name="EXECUTION_CODE"/>
<term value="true" name="PDDL_IDENTIFICATION"/>
<term value=
"org.pelea.languages.pddl.comparison.EffectsPDDLComparison"
name="COMPARISON_CLASS"/>
</node>
</nodes>
</configuration>
```

Script start.sh to start the execution of the system:

```
#!/bin/bash

sudo chmod 777 /dev/ttyUSB0
roslaunch ../../p2os/p2os_driver/launch/p2os_driver.launch &

while getopts ":p:" opt; do
  case $opt in
    p)
      PROBLEM_NUMBER=$OPTARG
      ;;
  esac
done

gnome-terminal \
--tab -e "bash -c \"java -jar ./dist/Warehouse.jar -c
./config/FF-$PROBLEM_NUMBER.xml -n M1; exec bash\""
--title "Monitoring - Pelea" \
--tab -e "bash -c \"java -jar ./dist/Warehouse.jar -c
./config/FF-$PROBLEM_NUMBER.xml -n DS1; exec bash\""
--title "Decision support - Pelea" \
--tab -e "bash -c \"java -jar ./dist/Warehouse.jar -c
./config/FF-$PROBLEM_NUMBER.xml -n P3DX1; exec bash\""
--title "Execution P3DX - Pelea" \
--tab -e "bash -c \"java -jar ./dist/Warehouse.jar -c
./config/FF-$PROBLEM_NUMBER.xml -n NAO1; exec bash\""
--title "Execution NAO - Pelea" \
&>/dev/null
```

Bibliography

[1] T. Fukuda, Y. Hasegawa, K. Kosuge, K. Komoriya, F. Kitagawa, and T. Ikegami, "Environment-adaptive antipersonnel mine detection system-advanced mine sweeper," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 3618–3623, IEEE, 2006.

[2] I. W. Hunter, T. D. Doukoglou, S. R. Lafontaine, P. G. Charette, L. A. Jones, M. A. Sagar, G. D. Mallinson, and P. J. Hunter, "A teleoperated microsurgical robot and associated virtual environment for eye surgery," *Presence: Teleoperators & Virtual Environments*, vol. 2, no. 4, pp. 265–280, 1993.

[3] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, p. 9, 2008.

[4] M. Ai-Chang, J. L. Bresina, L. Charest, A. Chase, J. C. jung Hsu, A. K. Jónsson, B. Kanefsky, P. H. Morris, K. Rajan, J. Yglesias, B. G. Chafin, W. C. Dias, and P. F. Maldague, "Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission.," *IEEE Intelligent Systems*, vol. 19, no. 1, pp. 8–12, 2004.

[5] K. Rajan, C. McGann, F. Py, and H. Thomas, "Robust mission planning using deliberative autonomy for autonomous underwater vehicles," in *Proceedings of the Workshop on Robotics in Challenging and Hazardous Environments, ICRA*, (Rome, Italy), 2007.

[6] R.-L. Hsu, M. Abdel-Mottaleb, and A. K. Jain, "Face detection in color images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 5, pp. 696–706, 2002.

[7] H. S. Nwana, "Software agents: An overview," *The knowledge engineering review*, vol. 11, no. 03, pp. 205–244, 1996.

[8] C. Sierra, M. Wooldridge, and N. Sadeh, "Agent research and development in europe," *Internet Computing, IEEE*, vol. 4, no. 5, pp. 81–83, 2000.

[9] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," in *Proceedings of the 3rd international joint conference on Artificial intelligence*, pp. 235–245, Morgan Kaufmann Publishers Inc., 1973.

[10] M. Wooldridge, *An introduction to multiagent systems.* John Wiley & Sons, 2009.

[11] S. Russell, P. Norvig, and A. Intelligence, "A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, p. 27, 1995.

[12] J. Ferber, *Multi-agent systems: an introduction to distributed artificial intelligence*, vol. 1. Addison-Wesley Reading, 1999.

[13] Y. Kubera, P. Mathieu, and S. Picault, "Everything can be agent!," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 1547–1548, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[14] K. Capek, *Rossum's Universal Robots.* Science-fiction story, 1921.

[15] D. R. Yates, C. Vaessen, and M. Roupret, "From leonardo to da vinci: the history of robot-assisted surgery in urology," *BJU international*, vol. 108, no. 11, pp. 1708–1713, 2011.

[16] J. C. González, J. C. Pulido, F. Fernández, and C. Suárez-Mejías, "Planning, execution and monitoring of physical rehabilitation therapies with a robotic architecture," in *Proceedings of the 26th Medical Informatics Europe conference (MIE). Studies in Health Technology and Informatics*, vol. 210, pp. 339–343, 2015.

[17] D. Nakhaeinia, S. H. Tang, S. M. Noor, and O. Motlagh, "A review of control architectures for autonomous navigation of mobile robots," *International Journal of the Physical Sciences*, vol. 6, no. 2, pp. 169–174, 2011.

[18] R. A. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[19] A. A. Medeiros, "A survey of control architectures for autonomous mobile robots," *Journal of the Brazilian Computer Society*, vol. 4, no. 3, 1998.

[20] C. Guzmán, V. Alcázar, D. Prior, E. Onaindia, D. Borrajo, J. Fdez-Olivares, and E. Quintero, "Pelea: a domain-independent architecture for planning, execution and learning," in *Proc. ICAPS*, vol. 12, pp. 38–45, 2012.

[21] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[22] A. Gerevini and D. Long, "Plan constraints and preferences in pddl3," in *Proceedings of ICAPS'06 Workshop on Soft Constraints and Preferences in Planning*, (The English Lake Districk, Cumbria, UK), 2006.

[23] D. V. McDermott, "A heuristic estimator for means-ends analysis in planning.," in *AIPS*, vol. 96, pp. 142–149, 1996.

[24] B. Bonnet and H. Geffner, "Hsp: Heuristic search planner," 1998.

[25] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, pp. 253–302, 2001.

[26] S. Richter and M. Westphal, "The lama planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, no. 1, pp. 127–177, 2010.

[27] M. Helmert, "The fast downward planning system.," *J. Artif. Intell. Res.(JAIR)*, vol. 26, pp. 191–246, 2006.

[28] T. B. Moeslund and E. Granum, "A survey of computer vision-based human motion capture," *Computer vision and image understanding*, vol. 81, no. 3, pp. 231–268, 2001.

[29] M. Tkalcic, J. F. Tasic, *et al.*, "Colour spaces: perceptual, historical and applicational background," in *Eurocon*, 2003.

[30] M. Mason and Z. Duric, "Using histograms to detect and track objects in color video," in *Applied Imagery Pattern Recognition Workshop, AIPR 2001 30th*, pp. 154–159, IEEE, 2001.

[31] M. B. Dillencourt, H. Samet, and M. Tamminen, "A general approach to connected-component labeling for arbitrary image representations," *Journal of the ACM (JACM)*, vol. 39, no. 2, pp. 253–280, 1992.

[32] L. Mihaylova, P. Brasnett, N. Canagarajah, and D. Bull, "Object tracking by particle filtering techniques in video sequences," *Advances and Challenges in Multisensor Data and Information Processing*, vol. 8, pp. 260–268, 2007.

[33] W. W. Royce, "Managing the development of large software systems," in *proceedings of IEEE WESCON*, vol. 26, pp. 1–9, Los Angeles, 1970.

[34] B. Boehm, "A spiral model of software development and enhancement," *ACM SIGSOFT Software Engineering Notes*, vol. 11, no. 4, pp. 14–24, 1986.