Enhancing Maritime Defence and Security Through Persistently Autonomous Operations and Situation Awareness Systems

Georgios Papadimitriou

Ocean Systems Laboratory

School of Engineering and Physical Sciences

Heriot-Watt University

A thesis submitted for the degree of

Doctor of Philosophy

August 2018

Abstract

This thesis is concerned with autonomous operations with Autonomous Underwater Vehicles (AUVs) and maritime situation awareness in the context of enhancing maritime defence and security. The problem of autonomous operations with AUVs is one of persistence. That is, AUVs get stuck due to a lack of cognitive ability to deal with a situation and require intervention from a human operator. This thesis focuses on addressing vehicle subsystem failures and changes in high level mission priorities in a manner that preserves autonomy during Mine Countermeasures (MCM) operations in unknown environments. This is not a trivial task. The approach followed utilizes ontologies for representing knowledge about the operational environment, the vehicle as well as mission planning and execution. Reasoning about the vehicle capabilities and consequently the actions it can execute is continuous and occurs in real time. Vehicle component faults are incorporated into the reasoning process as a means of driving adaptive planning and execution. Adaptive planning is based on a Planning Domain Definition Language (PDDL) planner. Adaptive execution is prioritized over adaptive planning as mission planning can be very demanding in terms of computational resources. Changes in high level mission priorities are also addressed as part of the adaptive planning behaviour of the system. The main contribution of this thesis regarding persistently autonomous operations is an ontological framework that drives an adaptive behaviour for increasing persistent autonomy of AUVs in unexpected situations. That is, when vehicle component faults threaten to put the mission at risk and changes in high level mission priorities should be incorporated as part of decision making.

Building maritime situation awareness for maritime security is a very difficult task. High volumes of information gathered from various sources as well as their efficient fusion taking into consideration any contradictions and the requirement for reliable decision making and (re)action under potentially multiple interpretations of a situation are the most prominent challenges. To address those challenges and help alleviate the burden from humans which usually undertake such tasks, this thesis is concerned with maritime situation awareness built with Markov Logic Networks (MLNs) that support humans in their decision making. However, commonly maritime situation awareness systems rely on human experts to transfer their knowledge into the system before it can be deployed. In that respect, a promising alternative for training MLNs with data is presented. In addition, an in depth evaluation of their performance is provided during which the significance of interpreting an unfolding situation in context is demonstrated. To the best of the author's knowledge, it is the first time that MLNs are trained with data and evaluated using cross validation in the context of building maritime situation awareness for maritime security.

This thesis is dedicated to my parents, Konstantinos and Paraskevi, and to my sister, Vaia.

Acknowledgements

First, I would like to thank my supervisor prof. David Lane for giving me the opportunity to be part of the Ocean Systems Laboratory. His vision, experience and guidance helped me a lot throughout my PhD studies. A special thank you goes to Dr. Zeyn Saigol for his constructive advice and guidance during our weekly progress meetings without which studying to acquire a PhD degree would be a much harder task. I would also like to thank my colleagues in the Ocean Systems laboratory for all the fruitful conversations and the wonderful time spent together. I am extremely grateful to my family for being unconditionally supportive and believing in me. Last but not least, I would like to thank Despoina for her support all these years and for reminding me that work is not everything in life.

The work presented in this thesis received funding from the Defence Science and Technology Laboratory (DSTL) of the UK's Ministry of Defence under contract number DSTLX-1000064104.

ACADEMIC REGISTRY

Research Thesis Submission

Name:	Georgios Papadimitriou		
School:	Engineering and Physical Sciences		
Version: *(i.e. First, Resubmission, Final)*	Final	Degree Sought:	Doctor of Philosophy Electrical Engineering

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1) the thesis embodies the results of my own work and has been composed by myself
2) where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons
3) the thesis is the correct version of the thesis for submission and is the same version as any electronic versions submitted*.
4) my thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5) I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.
6) I confirm that the thesis has been verified against plagiarism via an approved plagiarism detection application e.g. Turnitin.

* *Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.*

Signature of Candidate:	Georgios Papadimitriou	Date:	10/08/2018

Submission

Submitted By *(name in capitals):*	
Signature of Individual Submitting:	
Date Submitted:	

For Completion in the Student Service Centre (SSC)

Received in the SSC by *(name in capitals):*			
Method of Submission *(Handed in to SSC; posted through internal/external mail):*			
*E-thesis Submitted (**mandatory for final theses**)*			
Signature:		Date:	

Please note this form should be bound into the submitted thesis.

Contents

List of Figures

List of Tables

Part I

Preliminaries

Chapter 1

Introduction

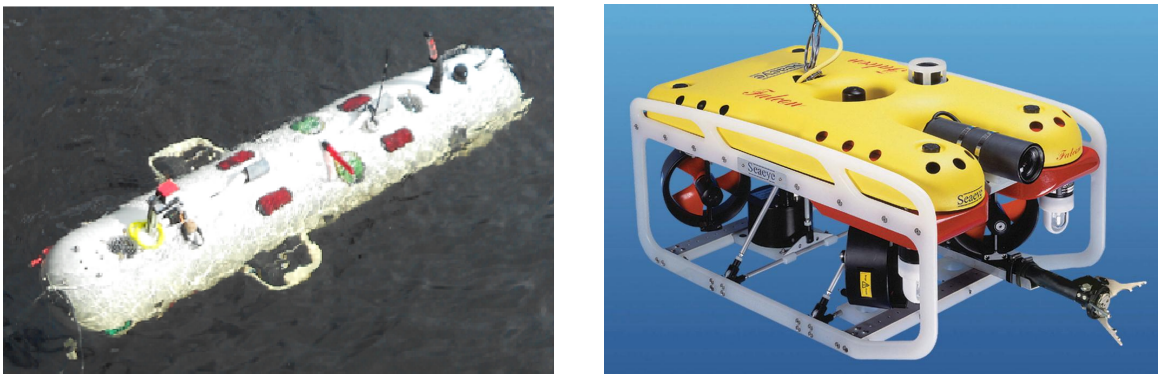
1.1 Problem Statement

Maritime defence and security is concerned with surveillance, monitoring, and threat neutralization operations both on the sea surface and underwater as well as in harbours. Ensuring maritime defence and security can be a very challenging task. More specifically, with respect to defence operations, reducing or eliminating personnel field risk exposure, acting in unknown and dynamic environments and rapidly responding to mission objective and priority changes while maintaining efficiency are some of the greatest challenges. Regarding maritime security, incomplete information and difficulties in efficiently fusing observations from various sources to identify abnormal behaviours and/or potentially illegal activities can heavily impact the level of security offered. Traditionally, the heaviest burden is placed on trained personnel which are tasked with conducting highly challenging, complex and in many cases dangerous operations often with limited resources and support. Over the years it became clear that in order for such operations to remain effective and keep up with the ever increasing challenges, changes to the aforementioned traditional approach had to be made. The direction followed was the introduction of sophisticated sensors, Maritime Situation Awareness (MSA) systems as well as Unmanned Underwater Vehicles (UUVs) and Unmanned Surface Vehicles (USVs) in an attempt to ease the burden on humans while at the same time providing more effective and robust solutions.

1.1.1 Autonomous Underwater Operations

Back in 2010 the UK Ministry of Defence (MoD) had expressed their intent to increase the use of a Autonomous Underwater Vehicles (AUVs) for underwater operations [6]. AUVs are one of the two major categories of UUVs (the other being ROVs which stands for Remotely Operated Vehicles). Figure 1.1 illustrates examples of the two major UUV categories. As opposed to ROVs, AUVs are not tethered to a surface vessel or ground station for virtually unlimited power supply communication and piloting from some operator. They are tetherless, autonomous and run on dedicated power supply such as batteries. AUVs are

potentially more cost effective to deploy, reduce risk to human personnel in the field, provide enhanced effectiveness through quality of sensor data and length of deployment, can be deployed organically and with greater agility around a theatre of operation. Current generations of AUVs, however, are only capable of autonomous operation under human supervision, or with preprogrammed scripts for behaviours. Many over the horizon or extended endurance operations required to achieve the benefits above will need greater sophistication in automatically responding to unknown and uncertain environments, failure of vehicle subsystems, and changes in objectives. A specific example of this in the un-



(a) Nessie V AUV (b) SeaEye Falcon ROV

Figure 1.1. Examples of the two major UUV categories

derwater domain is the current Mine Countermeasures, Hydrography and Patrol Capability (MHPC) programme [6], where off board sensors (i.e. AUVs) will be deployed from metal hull minesweepers conducting Mine Countermeasures (MCM) operations by the end of the decade. For the Concept of Operations (ConOps) currently under development, longer endurance with greater flexibility, robustness, adaptability and persistence in autonomy than are available in current commercial AUVs (e.g. REMUS) will be required.

In the future, AUVs must be more persistent. In practice, this means they must make less requests for assistance from an operator when they get stuck due to a lack of cognitive ability to deal with a situation. This requires that they are capable of adapting their mission plans on the fly in response to changes observed in the environment around them and in themselves (i.e. failures), using sensors and by communication. Environments will be partially known a priori, and must therefore be observed and modelled internally as a basis for decision making in planning. Goal sequences that produce a priori mission plans must be capable of modification and task sequences adapted while maintaining coherence with entry and exit states required by other parts of the plan. Consequently, this requires adaptability in world modelling, planning and execution in both space and time, in the presence of vehicle and sensor constraints and limitations (e.g. endurance, speed, range and accuracy).

1.1.2 Maritime Situation Awareness

Maritime Situation Awareness (MSA) is concerned with building and maintaining awareness of an unfolding situation that involves environmental elements on the sea surface or

underwater. The purpose of MSA is to identify incidents that need to be dealt with to avoid or constrain undesirable effects that can threaten safety and security in the area where such incidents are observed and/or potentially "linked" areas that can be also affected. Regarding environmental elements, these can vary from surface vessels such as ships to underwater ones such as submarines. We call these dynamic elements since they can move. In addition, environmental elements can be static such as oil rigs and harbours, to name but a few, while situations, and consequently incidents, can involve various combinations thereof. Incidents can be either intended or unintended. Unintended incidents are related to safety, e.g. a cargo vessel being on a collision course with an oil extraction platform. On the other hand, intended incidents are related to intended illegal activities such as smuggling and terrorism [7]. In an increasingly uncertain and dangerous global setting, the threat of people and organizations involved in illegal activities at sea is greater than ever. Small vessels can be used to board other bigger vessels in order to loot them and/or hold their crew as hostage for ransom in acts of piracy and terrorism. Moreover, all sorts of smugglers and traffickers (arm, drug, human, oil etc.) commonly use naval routes to promote their activities and distribute their cargo. Irrespective of the situation, building and maintaining awareness of it correctly and in a robust manner is a non trivial task. The three most prominent challenges are: i) the high volumes of information gathered from various sources, ii) their efficient fusion taking into account any contradictions (observations uncertainty) and iii) the requirement for reliable decision making and (re)action in the presence of potentially multiple interpretations of a situation (situation uncertainty).

Currently maritime situation awareness is mostly built around human operators following in this manner a human centric approach, meaning that humans are heavily engaged in the assessment of an unfolding situation. More specifically, under such a setting, humans have to evaluate vast amounts of information some of which may be contradicting making its fusion extremely difficult even for operators with years of experience. Other factors such as fatigue can lead to poor judgement and decisions which at best delay the identification of incidents while at worst can be catastrophic. That is, complete oversight of major incidents that could put human lives at risk and pose a great threat to maritime security. The identification of the aforementioned problems with the human centric approach led to a gradual shift towards systems centric maritime situation awareness over the last few years. That is, the development and deployment of situation awareness systems whose purpose is to ease the burden on humans and support them. Sometimes system centric situation awareness is referred to as next generation situation awareness [8]. Currently, such systems commonly rely on human domain experts to transfer and encode their knowledge into the system so that it can be later deployed. This is mostly a repetitive, trial and error, time consuming process. There are also studies suggesting that there is loss of information since only a small portion of knowledge is actually transferred (e.g. see Nilsson et al. [9]). As such, knowledge acquisition should not only be automated but also come directly from data.

1.2 Thesis Objectives

In this thesis we aim to present a framework for persistently autonomous underwater operations with AUVs, significantly reducing operator requests for assistance in environments of increased complexity. Three core, interlinked areas of science will be addressed: i) ontology-based (semantic) knowledge representation and reasoning, ii) planning and execution of missions, iii) plan and execution adaptation on the fly, in response to new information acquired through sensors while the vehicle is acting in its environment as well in response to changes in mission requirements and internal vehicle state, working under resource constraints. In the context of maritime defence, the main focus is on applying the framework for conducting persistently autonomous MCM. In that respect, one of the main objectives of the thesis is to demonstrate the benefits of an ontology-based approach that can be used by AUVs intended for MCM to express knowledge and reason about their state, the state of the environment in which they are deployed, mission goals and high level mission priorities. That is, on the one hand we aim to demonstrate how this knowledge and reasoning are efficiently combined to plan and execute MCM missions successfully. On the other hand, we aim to demonstrate that in the presence of real time high level mission priority changes and vehicle component faults the vehicle employs an adaptive planning and execution behaviour. In the first case, the purpose is the facilitation of the need for adaptation due to real time mission requirement changes while in the latter, recovery and satisfaction of mission goals to the maximum extent possible. Thus persistence in autonomy can be achieved and maritime defence be enhanced. Experiments are conducted in a simulated MCM environment.

Outwith the context of persistently autonomous operations and maritime defence, this thesis aims to provide an insight into maritime situation awareness systems that support human operators in decision making. The main focus is on maritime situation awareness built with Markov logic networks [10], for which, one of the objectives is to to present an approach for training them with data. Another objective is to provide an in depth evaluation of their performance and how this is affected by the presence or absence of contextual information effectively extending the work of Snidaro et al. [5]. The investigated scenario is concerned with the identification of vessels rendezvousing in order to get involved in smuggling activities.

1.3 Thesis Contributions

One of the main contributions of this thesis is the development of a framework that can be used by AUVs for persistently autonomous underwater operations and can be found in Chapter 7. This will be of interest to the autonomous underwater operations community since it offers a novel combination of ontology-based knowledge representation and reasoning, adaptive planning and adaptive execution for promoting persistence in autonomy. In particular, the framework offers mitigation of AUV subsystems failures through a robust

ontological representation upon which reasoning procedures can reassess the AUV's internal state dynamically and in real time and enable the vehicle to adapt and recover so that mission goals are satisfied to the maximum extent possible. Another contribution is that the framework enables AUV's to respond to high level mission priority changes and adapt in real time. The application of the framework to MCM which is presented in Chapter 8 is of particular interest to the maritime defence community not only due to the aforementioned features but also due to the very nature of the high level mission priorities. That is, further to the standard MCM missions where energy consumption and/or execution time govern the mission, we have provided an alternative perspective in which criteria such as probability and entropy are also considered during mine reacquisition and inspection. Finally in Chapter 9, to the best of the author's knowledge, it is the first time that Markov logic networks are trained with data and evaluated using cross validation for building maritime situation awareness in the context of maritime security. This thesis has resulted in the following publications:

1. G. Papadimitriou and D. Lane,"Semantic based knowledge representation and adaptive mission planning for MCM missions using AUVs," in *OCEANS 2014-TAIPEI*, pp. 1-8, IEEE, 2014.

2. G. Papadimitriou, Z. Saigol, and D. M. Lane,"Enabling fault recovery and adaptation in mine-countermeasures missions using ontologies," in *OCEANS 2015- Genova*, pp. 1-7, IEEE, 2015.

1.4 Thesis Structure

Chapter 2 presents an overview of the research concerned with knowledge representation and reasoning. The focus is on logic-based, probabilistic and hybrid approaches which are critically analysed and for which various reasoning approaches are presented. Chapter 3 is concerned with the field of knowledge engineering with ontologies. In particular, various languages for authoring ontologies are presented while an account of our language choice is given and the components of ontologies based on it are described. In addition, a background of how ontologies are used in robotics is provided and the knowledge processing system upon which our persistently autonomous operations framework is based is introduced. Chapter 4 is concerned with presenting an overview of the field of Artificial Intelligence (AI) planning along with various AI planning approaches such as temporal planning. In addition, it offers a useful background of temporal planning with the Planning Domain Definition Language (PDDL). Chapter 5 provides an overview of the maritime situation awareness field through: i) a presentation of common sources of information utilized in building maritime situation awareness, ii) an analysis of the role of context, uncertainty and prior knowledge of the field, iii) data fusion models commonly used in the field and iv) a critical analysis of the various maritime situation awareness systems. Chapter 6 is a rather brief chapter that provides a general thesis background so that concepts and

mechanisms described within Chapter 7 are better understood. Having said that, Chapter 7 describes the aforementioned persistently autonomous operations framework where the framework's architecture is presented and a detailed analysis of the framework's ontology-based knowledge representation and reasoning is provided. Finally, Chapter 7 describes the adaptive mission planning and execution approach followed within the framework. Chapter 8 presents the application of the framework to MCM. The chapter starts with an introduction to MCM and the phases which it comprises and continues with the few modifications made to the framework in order to be applicable to the MCM setting. Moreover, experimental results with respect to conducting MCM under different high level mission priorities are presented as are experimental results that demonstrate the framework's adaptive behaviour and recovery capabilities in the presence of high level mission priority changes and component faults respectively. Chapter 9 is concerned with maritime situation awareness in the context of maritime security like Chapter 5 but from a different perspective. More specifically, Chapter 9 is concerned with training and evaluating Markov Logic networks for maritime situation awareness building on the background knowledge provided in Chapter 5. Generative, likelihood-based weight learning approaches are presented within the chapter. Experiments and experimental results are provided and analysed in the context of identifying vessels rendezvousing in order to get involved in illegal activities. Finally, in Chapter 10, we present our overall conclusions and suggestions for future work.

Part II

Background

Chapter 2

Knowledge Representation and Reasoning

This chapter presents a broad field of the Artificial Intelligence (AI) which is concerned with knowledge representation and reasoning. Conceptually we divided the chapter into three main sections in order to present logic-based, probabilistic and hybrid approaches (see Sections 2.1, 2.2 and 2.3 respectively). Logic-based approaches are fundamental in understanding both ontologies which are part of Chapters 3, 7, 8 and the logic part of hybrid approaches in Section 2.3. In particular, the description logics section (Section 2.1.3), is relevant for the OWL DL ontologies that lie in the center of autonomous underwater operations while first-order logic in Section 2.1.2 is relevant to Markov logic networks in Section 2.3.3. Markov logic networks are used in Chapter 9 for maritime situation awareness. In addition, probabilistic approaches in Section 2.2 are useful in understanding the probabilistic aspect of hybrid approaches in Section 2.3.

2.1 Logic-Based Approaches

In this section we present the fundamental building blocks of logic-based knowledge representation and reasoning systems in a way that is not specific to the type of logic used. Therefore, we aim to produce a reference point that will guide the reader through the remainder sections (Sections 2.1.1 - 2.1.3) where the fundamental concepts will be specialized and expanded accordingly.

The core structure of a logic-based system is its Knowledge Base (KB). A KB consists of a set formulas. Depending on the logical representation (language) used, formulas have a syntax and semantics. Syntax refers to how formulas can be expressed while semantics refer to the meaning of the formulas. Through semantics we compute the truth values of formulas with respect to each possible world. Every formula in logic has to be either true or false with the exception of fuzzy logic where we have intermediate levels of truth which are also called degrees of truth. Possible worlds are possible environments[1] that a logic-

[1] Environments can be real or simulated.

based agent might or might not be in [11]. The term model can be used instead of possible world when we strictly refer to mathematical abstractions that are used to calculate the truth values of formulas. In order for this to be possible formulas must not contain occurrences of free variables or no variables at all. We will clarify this in Sections 2.1.1 and 2.1.2 where we talk about the syntax and its elements as well as the semantics of formulas. Now, if a formula a is true in a model m, then we say that m is a model of a or that a satisfies m [11]. The notation used for denoting all models of a formula a is $M(a)$.

Next, we will talk about reasoning and associated concepts. Reasoning is the act of drawing/deriving conclusions. A fundamental concept in logical reasoning is entailment which is also known as logical consequence, that is, a formula follows from another formula [11]. Therefore, entailment describes the relation between the meanings of formulas. We will say that a formula a entails formula b ($a \models b$) iff (if and only if) in every model in which a is true, b is also true [11]:

$$a \models b \Leftrightarrow M(a) \subseteq M(b) \tag{2.1}$$

Now, if we expand the concept of entailment to be applicable to knowledge bases by treating a KB as a set of formulas or alternatively as a single formula that asserts all formulas in the KB, we can use entailment to draw/derive conclusions based on the KB [11]. That is, perform inference via model checking in which we check if some formula b is true whenever the KB is true. Substituting a with KB in Equation 2.1 we get:

$$KB \models b \Leftrightarrow M(KB) \subseteq M(b) \tag{2.2}$$

To disambiguate between entailment and inference, consider a treasure hunting game. If we think of all the logical consequences of a KB as a set of all possible locations of the treasure and b as the treasure then: i) entailment is the treasure being in one of the locations and ii) inference is actually finding the treasure. Inference can be formally represented as follows:

$$KB \vdash_i b \tag{2.3}$$

where, i is an inference algorithm and the expression is read as b is derived from KB by i.

Many people use the terms inference and reasoning interchangeably for referring to the same thing: drawing conclusions. Formally though, the term inference refers to the means used for drawing conclusions while reasoning refers to drawing conclusions. That is, the means for achieving the goal and the goal itself respectively.

When talking about inference we effectively refer to algorithms that are applied to a KB and attempt to yield an outcome. Consequently, inference can be analysed in terms of soundness, completeness and decidability. An inference algorithm is sound iff its outcome provides proof only for KB formulas that do not violate the semantics of the type of logic used. More formally, an inference algorithm is sound when it “derives only formulas that

are entailed in the KB" [11]. This is also known as the correctness property. Complete is any inference algorithm that can derive all valid outcomes. More formally, an inference algorithm is called complete when it can derive any formula that is entailed in the KB. Additionally, decidable is any inference algorithm that derives a formula that is entailed in the KB provided that one exists.

So far, we have introduced and analysed several fundamental concepts of logical systems both in terms of knowledge representation and reasoning. This has hopefully created a solid basis in which the reader can return to for clarification when needed.

2.1.1 Propositional Logic

In this section we present a very simple logical language with limited expressive power, that is, propositional logic or propositional calculus [11]. We analyse propositional logic with respect to its syntax and semantics. Then we look into inference algorithms applicable in this simple type of logic.

2.1.1.1 Syntax

As evidenced by its name, the syntax of propositional logic refers to the set of rules defining the structure of allowable formulas as well as allowable logical connectives or operators. Formulas in propositional logic can be either atomic or complex. Atomic formulas or literals are any formulas that consist of a single proposition. Single propositions are also known as propositional variables and can be either true or or false and the naming convention is that they are denoted with an uppercase letter. A, N are some examples of single propositions or atomic formulas or propositional variables. Note that since propositional variables can be either true or false they are bound to a set of values (true/false). Now, complex formulas on the contrary consist of atomic formulas connected by logical operators and parentheses. We present the logical operators following their order of precedence when appearing in formulas:

1. $\neg$: Read as *negation* or *not*. The negation of a formula (a negated formula) is the opposite of a that formula and vice versa. For instance, N is the opposite of $\neg N$ and vice versa. N is also called a literal and $\neg N$ a negative literal.

2. $\wedge$: Read as *conjunction* or *and*. The conjunction between two or more literals produce a complex formula. For instance, $L \wedge M \wedge N$ is a conjunction with the conjuncts being L, M, N.

3. $\vee$: Read as *disjunction* or *or*. The disjunction between two or more literals produce a complex formula. For instance, $L \vee N$ is a disjunction with the disjuncts being L, N.

4. $\Rightarrow$: Read as *implies*. The implication – as its name suggests – connects two formulas to denote that some formula implies some other formula or that some implied formula

follows from some other formula. For instance, $L \wedge M \vee N \Rightarrow Q$ is an implication in which $L \wedge M \vee N$ is called the antecedent and Q is called the consequent.

5. $\Leftrightarrow$: Read as $if\ and\ only\ if$ or *biconditional* or $equivalent\ to$. The equivalence connects two formulas that are equivalent. For instance, $N \wedge L \Leftrightarrow \neg M$ denotes that $N \wedge L$ is equivalent to $\neg M$.

2.1.1.2 Semantics

The semantics in propositional logic are a set of rules based on which the truth values (*true* or *false*) of formulas are determined given some model (see Section 2.1). To determine the truth values of formulas one needs to know the truth values of propositions. For n propositions in a KB the number of possible models is 2^n. Table 2.1 illustrates the possible models for a KB whose formulas consist of two propositions.

Propositions	**Possible Models** (m_i)
L, M	$m_1 = \{L = true, M = false\}$ $m_2 = \{L = true, M = true\}$ $m_3 = \{L = false, M = false\}$ $m_4 = \{L = false, M = true\}$

Table 2.1. Possible models m_i for a KB whose formulas consist of two propositions L, M.

Let us now introduce two special propositions, the *True* and the *False* proposition. *True* and *False* are propositions and they should not be confused with *true* and *false* which are truth values. To distinguish between the two, notice that *True* and *False* begin with an uppercase letter which denotes that they are propositions while *true* and *false* begin with a lowercase letter. *True* is special proposition that is always *true* and *False* is a special proposition that is always *false*. To complete this section, we present Table 2.2 which is a *truth table* that illustrates the semantic rules in propositional logic.

Propositions		**Negation**	**Conjunction**	**Disjunction**	**Implication**	**Equivalence**
L	M	$\neg L$	$L \wedge M$	$L \vee M$	$L \Rightarrow M$	$L \Leftrightarrow M$
$true$	$true$	$false$	$true$	$true$	$true$	$true$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$false$	$false$	$true$	$false$	$false$	$true$	$true$

Table 2.2. Truth table of propositional logic semantics associating logical connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$) with propositions L, M.

2.1.1.3 Inference

Inference in propositional logic can be achieved through theorem proving or model checking. In theorem proving we apply inference rules to KB formulas trying to derive a proof

of a desired goal. In contrast, in model checking we enumerate all possible models trying to derive a proof that the desired goal holds in all possible models [11].

Let us begin with theorem proving and inference rules. Inference rules are used by theorem proving inference algorithms and guarantee sound inference. Recall from Section 2.1 that an inference algorithm is sound when it derives only formulas that are entailed in the KB. Table 2.3 illustrates some of the most famous inference rules in propositional logic for a complete list of rules the interested reader should refer to [11].

Inference Rule Name	**Formulation**	**Meaning**
Modus Ponens	$\frac{a \Rightarrow b, \quad a}{b}$	Given any formulas of the form $a \Rightarrow b$ and a, then formula b can be inferred.
Modus Tollens	$\frac{a \Rightarrow b, \quad \neg b}{\neg a}$	Given any formulas of the form $a \Rightarrow b$ and $\neg b$, then $\neg a$ can be inferred.
Hypothetical Syllogism	$\frac{a \Rightarrow b, \quad b \Rightarrow c}{a \Rightarrow c}$	Given any formulas of the form $a \Rightarrow b$ and $b \Rightarrow c$, then $a \Rightarrow c$ can be inferred.
Disjunctive Syllogism	$\frac{\neg a, \quad a \vee b}{b}$	Given any formulas of the form $\neg a$ and $a \vee b$, then b can be inferred.
And-Elimination	$\frac{a \wedge b}{a}$	Given a conjunction of any formulas $a \wedge b$, then any of the conjuncts can be inferred.
Resolution	$\frac{\neg a \vee b, \quad a \vee c}{c \vee b}$	Given any formulas of the form $\neg a \vee b$ and $a \vee c$, then $c \vee b$ can be inferred.

Table 2.3. Sample table of proposition logic inference rules. a, b, c are formulas.

Except for the property of soundness another desirable property in inference algorithms is completeness. Recall from Section 2.1 that an inference algorithm is complete when it can derive any formula that is entailed in the KB. Soundness can be expressed as follows: $KB \vdash_i a \implies KB \models a$, where KB is a knowledge base, a is a formula and i an inference algorithm. That is, if a is derived from KB by some i, then KB entails a. In other words, if we can prove a from KB given some i, then a is true given KB. Now, completeness can be expressed as follows: $KB \models a \implies KB \vdash_i a$. That is, if KB entails a then a can be derived from KB by some i. In other words, if a is true given KB, then we can prove a from KB given some i.

Algorithms that couple the resolution inference rule (Table 2.3) with any complete search algorithm are also complete. To achieve this the KB has to be expressed in Conjunctive Normal Form (CNF). A KB in CNF consists exclusively of conjunctions of clauses. A clause is a disjunction of literals. Luckily, any formula in propositional logic can be written as a conjunction of clauses and be logically equivalent to the original. Additional theorem proving inference algorithms are the forward and backward chaining algorithms.

In order for forward and backward chaining to be applicable, the propositional KB has to be in definite clausal form. A definite clause is a disjunction of literals with exactly one positive (i.e. non-negated) literal while a disjunction of literals with no positive literals is called a goal clause. Both definite and goal clauses are Horn clauses. A Horn clause is a disjunction of literals with at most one positive literal. Formulas that consist of only

a single positive literal are called facts. A Horn clause can be written as an implication whose antecedent is a conjunction of positive literals and its consequent a positive literal (for definite clauses) or *False* (for goal clauses). For the case of a fact, the antecedent is the *True* proposition and the consequent is the single positive literal. For example, $\neg A \vee \neg B \vee C$, $\neg A \vee \neg B$ and C can be written as $A \wedge B \Rightarrow C$, $A \wedge B \Rightarrow False$ and $True \Rightarrow C$ respectively. Having a Horn KB of definite clauses written as a series of implications enables both the forward and backward chaining algorithms to exploit the Modus Ponens inference rule (see Table 2.3). Algorithm 1 illustrates the forward chaining inference algorithm in propositional logic for a Horn KB that consists of definite clauses [11].

Algorithm *ForwardChainingPL(KB, query)* **returns** *true* or *false*
Input: *KB*, a propositional knowledge base with definite clauses
query, a propositional symbol (proposition)
counter ← a table, where $counter[c]$ is the number of propositional symbols in c's antecedent
inferred ← a table, where $inferred[s]$ is initially *false* for all propositional symbols
agenda ← a queue of propositional symbols, initially symbols known to be *true* in the *KB*

while *agenda* is not empty **do**
  $p \leftarrow$ Pop(*agenda*)
  if $p = query$ **then**
    return *true*
  end
  if $inferred[p] = false$ **then**
    $inferred[p] \leftarrow true$
    for each clause c in *KB* where p is in c.ANTECEDENT **do**
      decrease $counter[c]$
      if $counter[c] = 0$ **then**
        add c.CONSEQUENT to *agenda*
      end
    end
  end
end
return *false*

Algorithm 1: The forward chaining algorithm in propositional logic for a knowledge base that consists of definite clauses [11].

The algorithm takes as input a *KB* consisting of definite clauses and a *query* which is a propositional symbol (proposition). The *agenda* is a queue that contains propositional symbols that are known to be *true* before the algorithm starts processing the KB. The *counter* is a table that contains the number of propositional symbols whose value is unknown for each antecedent of every implication. *Inferred* is another table that contains information (*true*, *false* values) about whether a propositional symbol was processed by the algorithm. Initially all values in *inferred* are false. The algorithm starts by getting a proposition symbol

from the *agenda*. If the proposition symbol matches the *query* then the algorithm terminates returning *true*. If not, it checks whether that symbol has already been processed before and if not then it changes the respective value of *inferred* to true. The algorithm then continues by decreasing the *counter* for every antecedent in which that symbol is present and if the counter of an antecedent reaches zero then its consequent is added to the *agenda*. This process is repeated until the *agenda* is empty or until a symbol from the *agenda* matches the *query*. Forward chaining is sound and complete and runs in linear time $O(n)$ in the size of the KB. Also, it is classified as a data-driven inference algorithm because it starts processing based on known data (*true* propositional symbols).

The backward chaining algorithm operates differently from the forward chaining algorithm in the sense that it starts from the *query* working backwards. That is, it does not start searching the implications in which the initial *true* propositions are part of the antecedents, trying in this way to infer the consequents and by extension the query. Backward chaining instead starts from the *query*, searching for implications whose consequents are the *query*, trying to prove the antecedents. If one of them is proved to be true then the *query* is *true*. If none is *true*, the *query* is *false*. Backward chaining is sound and complete and runs in linear time $O(n)$ in the size of the KB. In addition, it is classified as a goal-driven inference algorithm because it starts processing based on known goals.

So far we have presented the theorem proving reasoning approach. Let us now discuss about two algorithmic families that lie within the model checking reasoning approach. The first one is backtracking and the second is local search. Both families are used for solving the satisfiability problem also known as the SAT problem: checking the satisfiability of a formula. A formula is satisfiable if it is true in at least one model.

Regarding backtracking in logic[2], the general idea behind it is enumeration of partial possible models in a tree structure given a formula to be checked for satisfiability. Backtracking starts with a node that is a partial possible model and at each stage it expands the node with a more refined children node. That is, refining the partial possible model by assigning a value to an unknown propositional symbol. If the refined model can no longer be a candidate for satisfying the formula, then the algorithm backtracks to the parent node and chooses assignment of a different value for the symbol. This operation is executed recursively until all solutions (possible models) are enumerated if any exists. Figure 2.1 illustrates how backtracking works. One of the most famous algorithms that utilise backtracking is the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [12, 13]. The input of the DPLL algorithm is any formula expressed in CNF and the output models that satisfy the formula (if any). DPLL is both sound and complete and with a time complexity of $O(2^n)$ in the size of the CNF formula.

In contrast to backtracking algorithms, local search algorithms only provide sound inference. The rationale behind local search is to iteratively change the truth values of propositional symbols in formula clauses that are expressed in CNF so that all clauses are satisfied. By doing so local search algorithms return a satisfying model or failure. Before the

[2]Not to be confused with backtracking processes in general

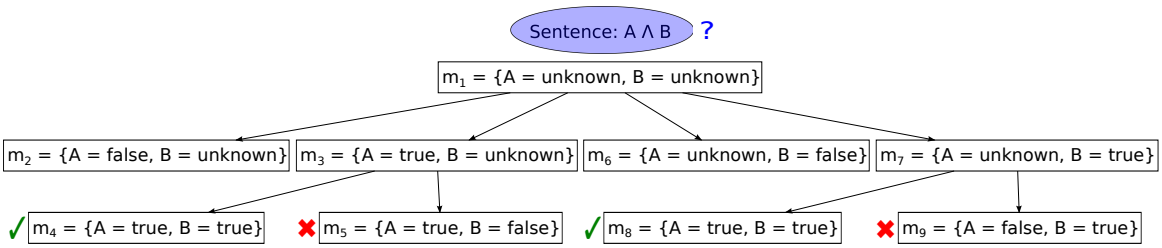


Figure 2.1. Backtracking as a search tree. Given a formula A ∧ B, the algorithm tries to enumerate all possible models (solutions) which satisfy the formula. First it creates model m_1 which is a partial possible model. Then expanding in a depth-first manner it generates m_2 which is also a partial possible model. Since m_2 can not satisfy the formula the algorithm backtracks to m_1 and it then generates m_3. m_3 is a partial candidate and it is expanded to m_4 which is a solution (possible model). The algorithm continues by generating m_5 and then backtracks to m_1, it generates m_6 but backtracks again since m_6 is not a partial possible model. Finally m_7 is generated and it is expanded to m_8 (a solution) and then m_9 (rejected model). The solution set is $\{m_4, m_8\}$.

iteration process commences all propositional symbols are given a truth value. The first iteration only differs from the initial truth value assignment by one truth value. So does every successive iteration form the previous one. Local search algorithms require evaluation functions that operate as terminating criteria. The number of unsatisfied clauses or the maximum number of truth value flips that a local search algorithm is allowed to perform are common evaluation functions in propositional logic [11]. The search space can suffer from local minima therefore local search algorithms employ various random assignment strategies to escape such minima. One of the most used algorithms in the family of local search is WALKSAT [14, 15]. WALKSAT initializes by performing a truth assignment to all propositional symbols in a CNF formula. Then, for every iteration it chooses an unsatisfied clause at random (typically according to a uniform distribution) and randomly flips the value of one of its propositional symbols. The algorithm terminates when the maximum number of flips is reached and either returns a model that satisfies the formula or failure. Returning failure though does not necessarily mean that the formula is unsatisfiable. It might also be the case that the maximum number of flips was not enough.

2.1.2 First-Order Logic

In this section we present and discuss about a more expressive language than propositional logic (see Section 2.1.1) known as first-order logic. As a language, first-order logic bears a resemblance to natural human languages in that it is structured around objects and relations among objects (some of which are functions, i.e. relations that yield only one outcome for a given input) which are expressed following a specific syntax and semantics. Moreover, as in the case of natural language, reasoning is an important part of first-order logic which enables first-order logic-based systems infer new knowledge and perform decision making [11].

2.1.2.1 Syntax

In this section we describe the syntax of first-order logic. Since first-order logic is structured around objects, relations and functions, its core syntactic symbols represent exactly these (objects, relations and functions).

Constant symbols represent objects, predicate symbols represent relations while *function* symbols represent functions [11]. By convention, constant, predicate and function symbols begin with an uppercase letter. When compared to propositional logic, what different is that having relations and functions creates the need for *arity* (i.e. the number of arguments in a predicate or function). For example *George* and *David* are constants, $StudentOf(George, David)$ is a predicate with an arity of two denoting that *George* is a student of *David*. Whereas, $UniversityOffice(David)$ is a function[3] with an arity of one.

In first-order logic we also have the notion of *terms*. A *term* is a logical expression that refers to a function or an object. Therefore *David* (a constant) and $UniversityOffice(David)$ (a function) are terms. Objects can also be represented by *variables* instead of constants when the object we refer to is unknown. Thus variables are also terms. By convention variables begin with a lowercase letter. For example, $StudentOf(x, y)$ is a predicate with two variables x, y. In this form the predicate indicates that some person x is a student of some other person y in contrast to $StudentOf(George, David)$ where the two variables have been replaced by two constants (*George* and *David*). The process of replacing variables with constants is called *instantiation*, also known as *grounding*. Finally, predicate symbols applied to a tuple of terms are known as *atoms*.

Alike propositional logic, first-order logic has formulas that can either be *atomic* or *complex*. Atomic formulas in first-order logic are single predicates:

- $StudentOf(x, David)$
- $SisterOf(Despoina, x)$
- $BrotherOf(FatherOf(Despoina), x)$

or an *equality* (denoted as $=$) between two terms:

- $UniversityOffice(David) = 2.17$
- $MotherOf(John) = Alexandra$

At this point, before we continue with the syntax of first-order logic, it is important that we clarify something about atomic formulas that contain non-free, also known as bound, variables or no variables at all as in the case of equality that we just presented. Such atomic formulas are known as *closed* atomic formulas and *ground* atomic formulas respectively or simply atomic *sentences*[4]. Now, complex formulas consist of combinations of atomic

[3] It is a function because a professor has normally only one office

[4] Can be used instead of closed or ground formulas. However we will **avoid** using the term sentence as it may cause confusion. For instance, the occurrence of term formula may be interpreted as an *open* (i.e. containing occurrences of free variables) formula which is not necessarily the case. It is like saying that when we use the term triangle we do not refer to equilateral triangles.

formulas, logical connectives (see Section 2.1.1), functions and symbols known as *logical quantifiers*. Quantifiers allow us to express information about collections of objects [11]. First-order logic supports two quantifiers: the *universal* denoted as $\forall$ and read as "for all" and the *existential* quantifier denoted as $\exists$ and read as "exists". Examples of complex *open* formulas are the following:

- $\neg StudentOf(x, John)$
- $StudentOf(x, David) \wedge StudentOf(George, David)$
- $\forall\, x \quad Friends(x, y) \Rightarrow SimilarHabits(x, y)$
- $\exists\, x \quad Student(x) \wedge StudentOf(x, y)$

As for complex closed formulas, they are complex formulas that contain non-free variables or no variables at all. For instance, consider the following complex (open) formula from the above set of complex formulas:

- $\forall\, x \quad Friends(x, y) \Rightarrow SimilarHabits(x, y)$

To form a complex closed formula from this complex formula we need to either substitute y with a constant, say $Peter$, or bind it as in the case of x with some quantifier, e.g.:

- $\forall\, x, y \quad Friends(x, y) \Rightarrow SimilarHabits(x, y)$

To conclude, a formula is closed if all of its variables are bound with quantifiers, also, quantifiers together with logical connectives form the *logical operators* of first-order logic. The precedence of operators is: $\neg > = > \wedge > \vee > \Rightarrow > \Leftrightarrow > \forall = \exists$.

2.1.2.2 Semantics

Recall from Section 2.1.1 that the semantics of propositional logic define the set of rules based on which the truth values of propositional formulas are determined given some model. In first-order logic models are richer than the ones in propositional logic.

Each model has a *domain*, denoted as D, which is the set of objects it contains [11]. A domain cannot be empty which means that every possible world must contain at least one object. In addition, each model consists of an *interpretation*, denoted as I, that "specifies exactly which objects, relations and functions are referred to by the constant, predicate and function symbols" [11]. An interpretation is itself a function:

- A function F of some arity n is a function $I(F)$ from D^n to D.
- A constant C is a function $I(C)$ from D^0 to D.
- A predicate P of some arity n is a function from D^n to $\{true, false\}$.

Now, in order to assign truth values to formulas we need to perform a series of actions:

1. Each variable x must be assigned/replaced with a constant C. That is, $\mu(x) = C$.

2. Every term $\{t_1, t_2, \ldots, t_n\}$ must be evaluated to an object from the domain $\{d_1, d_2, \ldots, d_n\}$. That is, $I(f(t_1, t_2, \ldots, t_n))(d_1, d_2, \ldots, d_n)$

3. Every atomic formula $P(t_1, t_2, \ldots, t_n)$ is assigned a truth value depending on whether $\langle u_1, u_2, \ldots, u_n \rangle \in I(P)$ (true) or $\langle u_1, u_2, \ldots, u_n \rangle \notin I(P)$ (false), where $u_1, u_2, \ldots, u_n$ are evaluations of the terms $t_1, t_2, \ldots, t_n$. A formula of the form $t_1 = t_2$ is evaluated to *true* if both t_1 and t_2 evaluate to the same object in D.

4. A complex formula with logical connectives is evaluated to *true* or *false* based on the truth table of propositional logic semantics (see Table 2.2). A formula of the form $\exists\, x \;\; \phi(x)$ is evaluated to *true* if and only if there is an assignment for x that satisfies $\phi(x)$. A formula of the form $\forall\, x \;\; \phi(x)$ is evaluated to *true* if and only if all assignments for x satisfy $\phi(x)$.

Except for the standard semantics that we presented above there are also alternative ones [11]. We will demonstrate those by providing one example. Consider the natural language expression "David has two PhD students, George and Hashim". By writing:

$$StudentOf(George, David) \wedge StudentOf(Hashim, David)$$

we do not capture what is really reflected by the natural language expression. First of all we would need to add that George and Hashim are not the same person and that David has no other students. The precise expression in first-order logic would then be:

$$StudentOf(George, David) \wedge StudentOf(Hashim, David) \wedge George \neq Hashim$$
$$\wedge \forall\, x \;\; StudentOf(x, David) \Rightarrow (x = George \vee x = Hashim)$$

Judging from the above logical expression one quickly identifies that translation from natural language is not as straightforward as one might think. Consequently this can lead to insufficient translations of natural situations in first-order logic and by extension to incomprehensible behaviours from first-order logic-based agents.

In the scope of alternative semantics we find the *unique names* assumption which dictates that every constant symbol must refer to a different object. In our example this assumption relinquishes the need to include $George \neq Hashim$. Another alternative semantics assumption is the *closed world* one which dictates that any atomic formula that is not known to be *true* is treated as *false*. In our example this assumption relinquishes the need to include:

$$\forall\, x \;\; StudentOf(x, David) \Rightarrow (x = George \vee x = Hashim)$$

Yet another assumption that can be made is *domain closure* which dictates that the only objects that a model contains are the ones that are referred to by constant symbols [11]. The

unique names, closed world and domain closure assumptions are also known as *database semantics*. One final point to be made is that there are no correct and incorrect semantics in general, only correct and incorrect semantics according to the conventions that we follow [11].

2.1.2.3 Inference

Inference in first-order logic is a well researched scientific topic for which many inference approaches have been developed over the years. In this section we will initially present inference rules for quantifiers and then move to the more complete approaches of first-order inference algorithms. Before continuing, remembering that in order for inference to take place, formulas in a KB must not contain free variables.

Now, One of the first approaches for performing inference in first-order logic was applying a two-stage process in which we first propositionalize the first-order KB and then reduce first-order to propositional inference [11]. The first stage is realised by applying inference rules that operate on quantified formulas. Regarding the universally quantified formulas, we apply the *universal instantiation* inference rule while in the case of existentially quantified formulas we apply the *existential instantiation* inference rule (see Table 2.4). Universal instantiation can be applied many times to produce new formulas while existential instantiation can be applied only once to a formula. In the case of universal instantiation the new formulas are *logically equivalent* to the original one while in the case of existential instantiation the new formula is only *inferentially equivalent*[5] to the original one.

Inference Rule Name	**Formulation**	**Meaning**
Universal Instantiation	$\frac{\forall\, x \quad a}{\text{SUB}(\{x/g\}, a)}$	For any formula a that is universally quantified on variable x, substitute every x in a with a ground term g.
Existential Instantiation	$\frac{\exists\, x \quad a}{\text{SUB}(\{x/l\}, a)}$	For any formula a that is existentially quantified on variable x, substitute every x in a with a constant symbol l that does not appear anywhere else in the KB.

Table 2.4. Universal and Existential inference rules in first-order logic.

Two examples (one for each rule) will help us understand the effect of the rules on quantified formulas. First consider the following KB:

$$\exists\, x \;\; StudentOf(x, David)$$
$$\exists\, y \;\; Friends(y, David)$$

[5]Inferentially equivalent are two or more formulas or KBs that are satisfied exactly and only when the originals are satisfied.

Applying existential instantiation on the above quantified formulas based on the following substitutions:

$$\text{SUB}(\{x/C_1\}, \exists\, x \;\; StudentOf(x, David))$$
$$\text{SUB}(\{y/C_2\}, \exists\, y \;\; Friends(y, David))$$

we get the following KB:

$$StudentOf(C_1, David)$$
$$Friends(C_2, David)$$

where, C_1 and C_2 are new constant symbols that do not appear anywhere else in the KB. Such constants are called *Skolem constants* [11]. Now consider the following KB:

$$\forall\, x \;\; Smokes(x) \wedge Alcoholic(x) \Rightarrow Cancer(x)$$
$$Smokes(George)$$
$$Alcoholic(George)$$
$$Friends(Mary, George)$$

Applying universal instantiation based on the ground terms of the above KB we get the following KB:

$$Smokes(George) \wedge Alcoholic(George) \Rightarrow Cancer(George)$$
$$Smokes(Mary) \wedge Alcoholic(Mary) \Rightarrow Cancer(Mary)$$
$$Smokes(George)$$
$$Alcoholic(George)$$
$$Friends(Mary, George)$$

The above KBs can be treated as a propositional KBs if every ground atomic formula is treated as a propositional symbol. Consequently, the inference algorithms that were described in Section 2.1.1 can be applied.

Reasoning by reducing first-order logic knowledge bases to propositional ones and then applying propositional inference is sound and depending on the propositional inference algorithm used it can be complete as well (see Section 2.1.1). This reasoning approach can be slow in large domains since it can generate many irrelevant instantiations such as $Smokes(Mary) \wedge Alcoholic(Mary) \Rightarrow Cancer(Mary)$[6].

Another, much faster approach, is *unification* and *lifting* [11]. Unification [11, 16] searches for suitable variable substitutions that unify different logical formulas, i.e. make

[6]It is obvious that the one getting cancer is George and not Mary given the remainder of the KB.

them look identical:

$$\text{UNIFY}(k,l) = \theta$$

where, k and l are formulas and θ a unifier (if one exists) such that $\text{SUB}(\theta,k) = \text{SUB}(\theta,l)$. For example, $\text{UNIFY}(StudentOf(George,x), StudentOf(y,David)) = \{x/David, y/George\}$ but what about $\text{UNIFY}(StudentOf(George, x), StudentOf(y, z))$? In the latter case there are more than one unifiers. That is, $\theta = \{x/George, y/George, z/George\}$ or $\theta = \{y/George, x/z\}$. In such cases we choose the Most General Unifier (MGU), i.e. the one that imposes the least restrictions on the variable values. Every other unifier is an instantiation of the MGU [16]. In the above example the MGU is $\theta = \{y/George, x/z\}$. The complete algorithm for computing MGUs can be found in [11]. By unification we eliminate the need for unnecessary instantiations but this creates the need for first-order inference algorithms since we now also have to deal with variables and by extension with first-order constructs. At this point we perform lifting which is extending propositional algorithms to do just this, operate on first-order KBs. This includes lifting the inference rules that were presented in Section 2.1.1. Lifted inference is more efficient than propositionalizing a KB and then applying propositional inference since through unification it performs only necessary substitutions.

Alike propositional logic, inference based on resolution is applicable in first-order logic if it is first lifted. Lifted resolution is also called generalized resolution and it is given by:

$$\frac{k_1 \vee k_2 \vee \ldots \vee k_m, \quad l_1 \vee l_2 \vee \ldots \vee l_n}{\text{SUB}(\theta, k_1 \vee k_2 \vee \ldots \vee k_m \vee l_1 \vee l_2 \vee \ldots \vee l_n)}$$

where, k_i and l_j are first-order complementary literals[7]. Two first-order literals are complementary if one unifies with the negation of the other: $\text{UNIFY}(k_i, \neg l_j) = \theta$. θ is the MGU of all k_i and l_j. In order for generalized resolution to be applicable, the first-order KB needs to be expressed in CNF. As in the case of propositional logic, every formula in first-order logic can be expressed in the CNF form [11]. Generalised resolution yields both sound and complete inference results.

After describing unification and lifting and presenting the lifted version of resolution it should not come as a surprise that both forward and backward chaining algorithms are also applicable in first-order logic if lifted. That is, they are converted to utilize a lifted version of Modus Ponens which is called Generalized Modus Ponens (GMP) and operates on first-order KBs of definite clauses. For atomic formulas p_i, p_i'–whose variables are assumed to be universally quantified–and a consequent q where there is a substitution θ such that $\forall\, i$ $\text{SUB}(\theta, p_i') = \text{SUB}(\theta, p_i)$–we can unify p_i' and p_i for all i–the formulation of GMP is the following [11]:

$$\frac{p_1', p_2', \ldots, p_n', \quad (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUB}(\theta, q)}$$

[7]Recall from propositional logic (Section 2.1.1) that a literal is an atomic formula or its negation.

which means that given the above conditions we can infer $\text{SUB}(\theta, q)$. GMP offers sound and complete inference. Although logical consequence (entailment) in first-order logic is semidecidable[8], for KBs that are function free entailment is decidable. Algorithm 2 illustrates a simple forward chaining algorithm for first-order KBs with definite clauses while Algorithm 3 illustrates simple backward chaining one.

Forward chaining takes as input a first-order *KB* consisting of definite clauses and a *query* which is a first-order atomic formula. The *new* variable is a set of new formulas inferred in every iteration of the algorithms main loop. Initially *new* is empty and it is initialized to empty after every iteration. In each iteration the algorithm standardizes-apart (i.e. eliminates clashes in variable names) a formula in the KB and applies the GMP rule. If the outcome q' does not unify with any formula already in the KB or *new* is not empty then it adds q' to *new* and attempts to unify q' with the *query*. If unification succeeds it then returns the outcome of this unification. The next step is to add *new* to the existing KB. This process is repeated until new is empty. After exiting the repeat-until loop the algorithm returns *false*. Consequently, the algorithm returns *substitutions* followed by *false* or just *false*. Since forward chaining utilizes the GMP rule it offers both sound and complete inference. In the case that the first-order KB does not contain any functions forward chaining runs in polynomial time. That is, given a KB with p predicates with a maximum arity of k and n constant symbols, forward chaining will "converge" after pk^n iterations.

Regarding backward chaining, The algorithm takes as input a first-order definite clause *KB*, *goals* which is a list of conjuncts that constitute a query and θ which is the current substitution initialized with $\theta = \{\}$. The algorithm starts by checking if *goals* is empty and if so it returns $\{\theta\}$. Then it applies θ to the first element of the *goals* conjunct list and the outcome is assigned to q'. Then for each formula in the *KB* the algorithm standardizes it apart. Given that the standardized apart formula equals $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$ and q unifies with q', the algorithm appends the rest of the *goals* conjuncts at the end of a list consisting of $[p_1, p_2, \ldots, p_n]$. The algorithm then assigns the new list to a variable called *new_goals* which is actually an updated list of *goal* conjuncts. Finally the algorithm is recursively called with new inputs which are: the *KB*, *new_goals* and the composition of unifiers θ and θ' which practically accumulates the unification substitutions together. The algorithm's outcome is assigned to *answers* along with *answers* from previous calls. *Answers* is a set of substitutions and it is returned when the algorithm terminates. Backward chaining is also used in Logic Programming (LP) which we are going to describe next.

As its name suggests, LP [17, 18] is a programming paradigm based on logic and is a subcategory of declarative programming. Declarative programming is a programming paradigm which dictates that systems should be given: i) the means to solve problems of interest and ii) the requirements for achieving a solution, iii) without being given a control flow on how to achieve a solution. There are many LP languages including ABSYS [19],

[8]Semidecidability in logic is a form of "decidability" in which given an arbitrary formula and a KB, there exists an algorithm which says always yes to the formula if it is entailed in the KB but there exists no algorithm which says no to the formula if it is not entailed in the KB.

Algorithm *ForwardChainingFOL(KB, query)* **returns** a *substitution* or *false*
Input: *KB*, a first-order definite clause knowledge base
query, an atomic formula
new ← a set of new formulas inferred in every iteration

repeat
  new ← {}
  for each *formula* in *KB* **do**
    $(p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(*formula*)
    for each θ such that SUB$(\theta, p_1 \wedge p_2 \wedge \ldots \wedge p_n) =$
    SUB$(\theta, p'_1 \wedge p'_2 \wedge \ldots \wedge p'_n)$ for some $p'_1, p'_2, \ldots, p'_n$ in *KB* **do**
      $q' \leftarrow$ SUB(θ, q)
      if q' does not unify with some formula already in *KB* or *new* **then**
        add q' to *new*
        $\phi \leftarrow$ UNIFY$(q', query)$
        if ϕ is not *fail* **then**
          return ϕ
        end
      end
    end
  end
  add *new* to KB
until *new* is empty;
return *false*

Algorithm 2: The forward chaining algorithm for a first-order knowledge base that consists of definite clauses [11].

Algorithm *BackwardChainingFOL(KB, goals, θ)* **returns** a set of *substitutions*
Input: *KB*, a first-order definite clause knowledge base
goals, a list of conjuncts that constitute a query
θ, the current substitution, the initialization is $\theta = \{\}$
answers ← a set of substitutions, the initialization is *answers* $= \{\}$

if *goals* is empty **then**
  return $\{\theta\}$
end
$q' \leftarrow$ SUB$(\theta,$ FIRST$(goals))$
for each *formula* in *KB* where STANDARDIZE-APART*(formula)* $=$
$p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow$ UNIFY(q, q') succeeds **do**
  new_goals $\leftarrow [p_1, p_2, \ldots, p_n |$ REST$(goals)]$
  answers $\leftarrow$ BackwardChainingFOL$(KB,$ *new_goals*, COMPOSE$(\theta, \theta')) \cup$
  answers
end
return *answers*

Algorithm 3: The backward chaining algorithm for a first-order knowledge base that consists of definite clauses [11].

Datalog [20, 21], HiLog [22], Prolog [23, 24] and Mercury [25] among others. In the remainder of this section we will focus on Prolog as it is one of the most widely used

languages for LP.

Prolog is a dynamically typed language i.e. type checking occurs during runtime. Prolog programs consist of *terms* which are either *atoms*, *numbers*, *variables* or *compound terms*. An *atom* is any sequence of characters, special symbols and numbers enclosed in single quotes, or any sequence thereof starting with a lower case character. For example, 'Anything', 'hello', i_am_an_atom, */s1k# and [] are all atoms with the last one being a special atom denoting a list. *Numbers* can be either integers or floats. Atoms and numbers can be also referred to as *constants*. A *variable* is any sequence of characters and the underscore symbol (_) that starts either with the underscore or an uppercase character. For example, I_aM_a_VARIAble, _i_AM_a_variable are variables. The underscore alone denotes a special type of variable called the *anonymous variable*. As opposed to ordinary variables, anonymous variables are used when we want to instruct Prolog that we are not interested in an instantiation for that particular variable. For example:

$$[X,X] = [1,2].$$

will yield false because X cannot have the value of 1 and 2 at the same time while:

$$[\_,\_] = [1,2].$$

will yield true because the anonymous variables are not instantiated. A *compound term* is a term followed by arguments which are also terms. For example, $student(X, brother\_of(a, b))$ is a compound term with *student* being the *functor* and X, $brother\_of(a, b)$[9] being the arguments.

A Prolog program–also called Prolog KB–is a set of terms forming *definite clauses* which are also known as Prolog *rules*. In order to understand the analogy to first-order logic let us provide the following example:

$$\neg Smokes(x) \vee \neg Alcoholic(x) \vee Cancer(x) \tag{2.4}$$

is a first-order formula in definite clausal form. Recall that a definite clause can be written as an implication whose antecedent is a conjunction of positive literals and its consequent a positive literal. Consequently formula 2.4 can be written as:

$$Smokes(x) \wedge Alcoholic(x) \Rightarrow Cancer(x) \tag{2.5}$$

In Prolog, formula 2.5 is represented by the following rule:

```
cancer(X) :- smokes(X), alcoholic(X).
```

where conjuncts $Smokes(x)$ and $Alcoholic(x)$ are separated by a comma (,) which in Prolog represents the $\wedge$ operator when used between terms. The $\vee$ operator is represented by a semicolon (;). Rules in Prolog must end with a full stop as should facts:

[9] $brother\_of(a, b)$ is also a compound term with functor $brother\_of$ and arguments a,b.

```
cancer(X) :- smokes(X), alcoholic(X).
smokes(george).  % a fact
alcoholic(george). /* another
                                    fact */
```

where, % and /*...*/ are used for single and multi-line comments respectively.

When performing queries Prolog starts an inference process which is based on depth-first backward chaining where rules and facts are evaluated in the order of appearance in the Prolog KB. Compared to first-order logic, Prolog utilizes database semantics (see Semantics in Section 2.1.2). Due to the depth-first backward chaining approach, inference in Prolog can in some cases suffer from redundancy and infinite loops [11] which makes inference incomplete. While redundancy is not a severe problem, infinite loops are. In order to reduce the occurrences of such phenomena we can use *memoization* [26–28]. Memoization in programming is essentially an optimization technique that allows us to cache results of expensive computations should they be required again in the future. In the context of Prolog and depth-first backward chaining memoization is realized as "caching solutions to subgoals as they are computed and reusing these solutions in the event that a subgoal recurs" [11].

2.1.3 Description Logics

Description Logics (DLs) [29, 30] also known as terminological logics and concept languages, are a family of knowledge representation languages that are subsets of first-order logic (some of these subsets are decidable) and can be used to represent the knowledge we hold about an application domain. DLs are an evolution of Semantic Networks and Frames and consist of different dialects that have different expressive power. In general, as a subset of first-order logic they are less expressive but retain the advantage of representing relational knowledge in a well defined and structured way. In the context of DLs, the knowledge we hold about an application domain can be distinguished into two categories. The first being general knowledge about the domain which is known as *intentional knowledge* and the second, specific knowledge of a particular configuration or instantiation of the domain which is known as *existential knowledge* [29]. In DLs knowledge resides in the so called DL KB. A DL KB comprises two components: the Terminological Box (TBox) which contains *intentional knowledge* and the Assertional Box (ABox) which contains *existential knowledge*.

In practice a TBox is realised as a set of domain concept definitions and their hierarchy the so called taxonomy. For clarification purposes let us provide an example. Consider a hypothetical underwater mine countermeasures domain where an underwater vehicle needs to locate and neutralize sea mines. In this context, domain *concepts* such as *PerceptualAgent*, *NavalMine*, *MooredMine*, *AUV* (Autonomous Underwater Vehicle), *NeutralisingSomething*, *NeutralisingAMooredMine*, *AnchorChain*, *Anchor* could be used to model the domain. In order to represent the domain more accurately we can organise the concepts in

a *taxonomy* using subconcept definitions such as an *AUV* is-a *PerceptualAgent*, a *MooredMine* is-a *NavalMine* and *NeutralisingAMooredMine* is-a *NeutralisingSomething*. Concepts are not limited to physical objects but can also represent non physical entities as in the case of *NeutralisingSomething* and *NeutralisingAMooredMine* which represent actions. Moreover, a TBox consists of *axioms* which are used for defining concepts by restricting their domain and range to individuals with certain attributes. For instance, the concept *NeutralisingAMooredMine* can be defined as a subconcept of *NeutralisingSomething* with the restriction *objectActedOn* that has to be some individual of a *MooredMine*:

$$NeutralisingAMooredMine \sqsubseteq NeutralisingSomething \sqcap \exists objectActedOn.MooredMine$$

Continuing with our running example, let us assume that we have set of *individual* entities that are instantiations of concepts: *AUV1*, an instantiation of *AUV*, *MooredMine1*, an instantiation of *MooredMine* and *AnchorChain1* an instantiation of *AnchorChain*. These instantiations, which are also referred to as concept assertions, are part of the ABox since an ABox, as its name suggests, contains assertional knowledge. Ergo, all *individuals* (concept assertions) present in a domain. Apart from *concepts* and *individuals* in DLs we have *roles*. *Roles* are used for concept definitions as well as for forming binary relations between individuals and for assigning attributes to individuals [2]. We already saw an example of a role used in a concept definition as part of an axiom, that is, the *objectActedOn* role acting as a restriction when defining the *NeutralisingAMooredMine* concept. With respect to binary relations between individuals we can have *MooredMine1 isTiedToChain AnchorChain1* with *isTiedToChain* being a role. Finally a role such as *hasWidth* can be used to assign the width dimension to our moored mine individual (e.g. *MooredMine1 hasWidth 4* meters). Role assertions concerning individuals, that is, excluding role assertions that are part of axioms, are also part of the ABox in addition to concept assertions.

The expressing power of a specific DL language can be roughly identified by its name since it is a very common convention to name a DL language so that its name is roughly indicative of the constructors it supports. In this context, the name of a DL language can be decomposed into two "parts": the basic logic used in the language and the extensions to that basic logic. Regarding the basic logic part, $\mathcal{AL}$ and $\mathcal{FL}$ are the most common and stand for attributive language and frame-based language respectively. Table 2.5 illustrates most common basic logic extensions in DLs as well as their meaning. In this context, $\mathcal{ALC}$ for example stands for attributive language with complements and supports concepts, concept intersections, concept unions, concept negations and universal as well as existential restrictions [31]. Additional naming conventions are also quite common in DLs. For example $\mathcal{S}$ is a naming convention used to indicate a language which is based on $\mathcal{ALC}$ but also supports transitive roles.

Basic Logic Extensions	Meaning
$\mathcal{U}$	concept disjunction
$\mathcal{C}$	complex concept negations (complements)
$\mathcal{E}$	full existential quantification
$\mathcal{F}$	functional roles
$\mathcal{H}$	role hierarchy
$\mathcal{I}$	inverse roles
$\mathcal{N}$	unqualified cardinality restrictions
$\mathcal{Q}$	qualified cardinality restrictions
$(\mathcal{D})$	data types
$\mathcal{O}$	nominals (enumerated classes of object value restrictions)
$\mathcal{R}$	limited complex role inclusion axioms, reflexivity and irreflexivity, role disjointness

Table 2.5. Most common basic logic extensions in DLs.

2.2 Probabilistic Approaches

In Section 2.1 we presented logic-based approaches for knowledge representation and reasoning which are the earliest in this vast area of research. Logic-based approaches assume a deterministic world and do not deal with uncertainty explicitly if at all. Many real world applications though require that acting agents handle uncertainty which stems from partial observability, partial knowledge, non determinism or a combination thereof. Therefore in this section we switch our focus on probabilistic approaches and more specifically to Probabilistic Graphical Models (PGMs) as a means of representing and reasoning about uncertain knowledge. Before diving into PGMs let us first provide some information about basic concepts in the field so that we build a *vocabulary* of terms.

The most fundamental building block of a PGM is its graphical representation. A graph is composed of *nodes*[10] which connect to each other via *edges*[11]. In the context of a PGM, nodes represent random variables while edges represent probabilistic relations among these variables. The graph as a whole represents the *joint* probability distribution over the variables that are part of the graph and contains information about how to decompose it in a product of *factors* through *factorization*. In order to compute the *joint* as well the *conditional* and the *marginal* probabilities and to perform *factorization* we use the following two fundamental rules of probability theory [32]:

$$\text{sum rule} \qquad P(X) = \sum_{Y} P(X,Y) \tag{2.6}$$

$$\text{product rule} \qquad P(X,Y) = P(Y|X)P(X) \tag{2.7}$$

where, X, Y are random variables, $P(X,Y)$ is a joint probability and is read as "the probability of X and Y", $P(Y|X)$ is a conditional probability and is read as "the probability of Y given X", $P(X)$ is a marginal probability read as "the probability of X". Based on the prod-

[10] Nodes are also known as *vertices*.

[11] Edges are also known as *arcs*.

uct rule (see Equation 2.7) and the symmetry property $P(X, Y) = P(Y, X)$ we can derive the *Bayes' theorem*:

$$\text{Bayes' theorem} \qquad P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}. \tag{2.8}$$

By applying the sum rule (see Equation 2.6) to the Bayes' theorem denominator we get:

$$\text{normalization constant} \qquad P(X) = \sum_{Y} P(X|Y)P(Y) \tag{2.9}$$

which serves as a normalization constant so that the sum of $P(Y|X)$ is equal to *one* which is a fundamental attribute of probability distributions.

Given two or more random variables, their joint probability distribution is the probability distribution expressing the probabilities of all possible combinations of their values. Like in any probability distribution, all these probabilities should sum up to *one*. Table 2.6 illustrates the joint probability distribution of three binary $(True, False)$ random variables A, B, C. The joint probability distribution for the above case can be expressed using the

A	B	C	Probability
True	True	True	0.1
True	True	False	0.2
True	False	True	0.1
True	False	False	0.05
False	True	True	0.1
False	True	False	0.1
False	False	True	0.05
False	False	False	0.3

Table 2.6. Exemplar joint probability distribution for binary random variables A, B, C.

notation $P(A, B, C)$. Based on the joint probability distribution we can compute both the *conditional* and the *marginal* probability distributions. A conditional probability distribution is a probability distribution that expresses the probabilities of some random variables given others. Continuing our running example with A, B, C, if we where to calculate the conditional probability distribution for A, B given $C = True$ we would get rid of any table entry in which $C \neq True$ (rows *two*, *four*, *six*, *eight*) and then divide each remaining entry with the sum of all remaining probabilities so that we normalize the probabilities to add up to *one* (see Table 2.7). The conditional probability distribution for this case is denoted as $P(A, B|C = True)$. Finally, a marginal probability distribution is a probability distribution that expresses the probabilities of a random variable irrespective of the values of the others. So if we were to compute the marginal probability of A denoted as $P(A)$ we would get Table 2.8.

There are two different families of PGMs, namely, *directed graphical models* and *undirected graphical models*. As their name suggests, directed graphical models comprise edges that have a direction. That is, there is an arrow and the end of each edge when connect-

A	B	C	Probability
True	True	True	0.286
True	False	True	0.286
False	True	True	0.286
False	False	True	0.142

Table 2.7. Exemplar conditional probability distribution for binary random variables A, B given $C = True$.

A	Probability
True	0.45
False	0.55

Table 2.8. Exemplar marginal probability distribution for binary random variable A.

ing two nodes illustrating the direction of the connection. Regarding directed approaches, the most commonly used one is Bayesian Networks (BNs) which are described in Section 2.2.1. In contrast, undirected graphical models comprise edges that have no direction. Markov Random Fields (MRFs), also known as Markov Networks (MNs), are the most commonly used undirected approach and are described in Section 2.2.2. Other types of PGMs exist but are omitted since they are outside the scope of this thesis.

Inference in PGMs can be primarily categorized based on the accuracy of its outcomes. In this context we can perform either approximate (e.g. sampling-based) or exact inference. The general idea behind approximate inference methods is to trade the accuracy of exact methods with efficiency so that inference can become tractable. Given the above categorization we can perform an additional one which is based on whether we are interested in the full posterior probability distribution (marginal, conditional, joint) or just interested in the most probable variable assignment of query variables. For example, in the first case, given a set of query random variables Q and a set of evidence (observations) E we can compute $P(Q|E)$. On the contrary, in the later case we compute only the most probable value assignment of Q i.e. $arg\ max_Q P(Q|E)$. This is also known as *maximum a-posteriori* (MAP) inference [33]. In the case where Q is the set of all the remaining non-evidence variables in the network (i.e. $Q \cup E = X$, where X if the set of all random variables in the network), then the process is known as the most probable explanation (MPE) inference [34, 35], a special case of MAP inference.

2.2.1 Bayesian Networks

A Bayesian network [36, 37] is a Directed Acyclic Graph (DAG) which represents probabilistic relations between random variables through directed edges and a joint probability distribution over all the random variables present in the graph. Depending on the domain of application random variables can be either observable or latent, continuous or discrete, or combinations thereof, e.g. a variable can be continuous and observable. A random variable corresponds to an attribute, feature, or hypothesis with respect to an entity for which there

can be uncertainty in the application domain. Bayesian networks are a well suited representation for expressing causal links between random variables, a feature stemming from their directed graphical representation nature.

The graph representation itself allow us to intuitively visualize random variables as well as their dependencies (relations) and probabilistic influence without having to necessarily interpret the underlying mathematical formulas. Figure 2.2 illustrates six small exemplar BNs and the joint probability distributions of their random variables. In Figure 2.2a we can

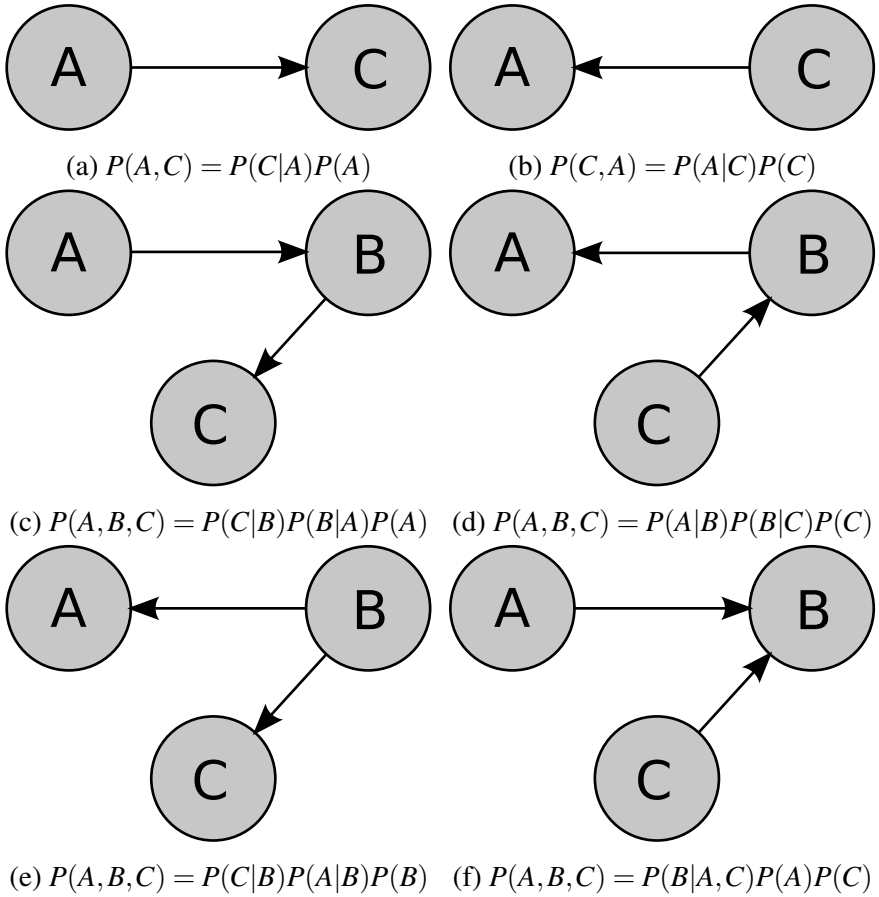


Figure 2.2. Exemplar Bayesian networks and the joint probability distributions of their random variables A, B, C.

clearly see that random variables A influences C i.e conditioning on A changes our belief about C. Similarly, it is clear by observing Figure 2.2b that C influences A. In general probabilistic inference is symmetrical. That is, if A can influence C then C can influence A. Now, Figures 2.2c – 2.2f are more "interesting" in the sense that there is an intermediate random variable B which affects probabilistic influence. In the case of Figure 2.2c, A influences C if B is not observed while if B is observed then A does not influence C. In addition, in the case of Figure 2.2d C influences A if B is not observed while C does not influence A if B is observed. Moreover, Figure 2.2e "tells" us that A influences C and vice versa if B is not observed while this is not the case if B is observed. Finally, Figure 2.2f represents what we call a V-structure. In this case A does not influence C if B and all of its descendants (if any) are not observed while we say that A influences C if either B or any of its descendants (if any) are observed. The above (exhaustive) cases of probabilistic

influence also demonstrate two concepts known as *d-separation* and *d-connectedness* [38], also called *directional-separation* and *directional-connectedness* respectively. Both concepts are interrelated. In general, we say that a random variable X is *d-separated* from a random variable Y if X does not influence Y or if Y does not influence X. In addition we say that X and Y are *d-connected* if X influences Y or if Y influences X. Now, given a third random variable Z, d-connectedness and d-separation are calculated in the same manner as probabilistic influence in cases of Figures 2.2c – 2.2f. Both concepts are applicable to sets of random variables in a very straightforward way. That is, two sets of random variables are d-separated iff (if and only if) each random variable in one set is d-separated from every over random variable in the other set and so on. In addition, both concepts are used in BN inference algorithms which is the topic of the next section.

2.2.1.1 Inference

As was mentioned earlier in this section, inference in PGMs can be primarily categorized as being exact or approximate based on its accuracy. In this section we will be describing both types of inference in BNs starting with exact methods and continuing with approximate ones.

The complexity of exact inference in BNs is related to the tree width of the graph [39, 11]. Exact inference in BNs is NP-hard. In the case of a Bayesian network being a polytree, which is also known as a singly connected network [40], time complexity of exact inference is linear in the size of the network. Clustering algorithms can be used in order to convert a Bayesian network into a polytree by clustering individual random variables (nodes) into hyper-nodes also called mega-nodes. One of the most common exact inference algorithms is variable elimination which is also known as bucket elimination [41] which is an improvement to the enumeration algorithm [11]. The rationale behind variable elimination is eliminating irrelevant to the inference random variables, i.e. non ancestors of query variables and evidence variables, and evaluating sums of products of conditional probabilities based on Equations 2.6 and 2.7. Despite achieving linear time complexity with polytrees exact inference still remains NP-hard which practically means that it is generally intractable. For this reason we need to consider approximate inference approaches.

One of the most common family of approaches for performing approximate inference in Bayesian networks is randomized sampling based methods also known as Monte Carlo (MC) methods. The accuracy of MC methods is dependent on the number of samples generated from a known probability distribution. In this family we find *direct sampling methods* which include sampling based on rejection. More specifically, *rejection sampling* generates samples from the prior distribution of the BN that is applied to. That is, for each random variable without a parent it performs a random assignment from its possible values and then carries on by computing conditional probabilities of the children random variables (nodes) given the parents until finishing processing the whole network. This process is performed for n times (algorithm parameter defined by the user). After n sampling iterations the al-

gorithm rejects those samples that are not consistent with the evidence and then estimates the posterior probability distribution given the evidence. Since rejection sampling first performs sampling and then rejection based on evidence it is rather inefficient since it wastes time to sample the network even with values that are contradicting to the evidence. To remedy this, we can utilize a different direct sampling method called *likelihood weighting*. Likelihood weighting generates samples by taking into consideration the evidence (observations) by only sampling the non evidence variables. The sampling of each non evidence variable is executed like in the rejection sampling approach but this time it is assigned a weight which is initially *one*. Every time the algorithm falls into a variable that is part of the evidence instead of sampling it, it fixes it to the value that it has in the evidence and multiplies the current weight with the conditional probability of the evidence variable given its parents. The outcome of this multiplication becomes the new weight. The process continues until the full network is processed in this manner yielding a full sampling iteration. At the end of each iteration we get a full sample of all non evidence variables and an associated weight. After n sampling iterations we will get weighted counts of each non evidence variable instantiation based on which we can compute an estimate of the joint posterior probability distribution and consequently be to able to estimate the conditional distribution given the evidence. However, likelihood weighting is prone to accuracy degradation as the number of evidence variable increases.

Another family of MC methods is the family of *Markov Chain* inference algorithms abbreviated as MCMC (Markov Chain Monte Carlo). In contrast to direct sampling methods MCMC methods do not sample the BN fully randomly each time. Instead they start with a random sampling of all non evidence variables (same as likelihood weighting without the weighting part) and then change the sample incrementally by sampling only one variable at a time at random. This means that each consecutive sample differs from the previous one by only one assignment. One of the most common and fundamental MCMC inference method is *Gibbs sampling*. The rationale behind Gibbs sampling is that it first samples all non evidence variables based on the values of the evidence variables which are kept fixed. This constitutes a sampling iteration which yields a complete sample (value assignment of all non evidence variable). For each consecutive sampling iteration the algorithm chooses one random non evidence variable at random and samples it based on its Markov blanket[12]. After n sampling iterations we will get counts of each non evidence variable instantiation based on which we can compute an estimate of the joint posterior probability distribution and consequently be to able to estimate the conditional distribution given the evidence. As in any sampling algorithm the accuracy of the estimate is related to the number of samples n.

Variational methods are also an alternative family of approximate inference approaches but are outside the scope of this thesis. For a description of variational methods the interested reader is referred to [32].

[12]The Markov blanket of a BN variable Q_i is the set of random variables (nodes) that are: the children and the parents of Q_i (C_i and P_i respectively) as well as all the remainder parents of C_i.

2.2.1.2 Applications

As was mentioned earlier, BNs are well suited for expressing causal links between random variables (entities) of an application domain. Therefore, they "have found their way" in many applications to diverse fields of the scientific spectrum.

One such field is utilization of BNs in the field of safety. In this area we find [42] where the authors propose a BN-based approach for safety risk assessment in steel construction projects. In [43] Bayesian networks are deployed for dynamic safety analysis of process systems while in [44] BNs are used for maritime safety management. Another area where BNs have been utilized is healthcare. An example of such application is the work presented in [45] where the authors take advantage of a Bayesian network approach for modelling medical problems for the purpose of personalized healthcare. Moreover, in [46] BNs are used as a decision making support method for diagnosing dementia. In the work presented in [47], BNs are used for meta-analysis tool for studies in patients with coronary artery disease. Other fields where BNs are applied are: fault diagnosis [48–50], meteorology [51–53] and neuroscience [54] among others.

Finally, of particular interest are the applications of BNs in the robotics field. In this area we find the work of [55] on context-aware home service robots. Furthermore, BNs are applied in sensor planning for mobile robot localization [55], robot mapping [56] and robot cognition [57] among others.

2.2.2 Markov Networks

A Markov Network (MN) [58], also known as a Markov Random Field (MRF), is an undirected graphical model which represents probabilistic relations between random variables (nodes) through undirected edges and a joint probability distribution over all the random variables present in the graph. Regarding the nature of random variables, in both MNs and BNs they express and represent the same things (see Section 2.2.1). In contrast to a BN though, a MN can be cyclic. This enables the encoding of cyclic dependencies between random variables, a feature not present in BNs. In addition, the undirected nature of the graphical representation of a MN makes it more suitable for expressing soft constraints between random variables when compared to a BN [32].

In BNs we saw how d-separation and d-connectedness represent the (conditional) independencies/dependencies of random variables in the graphical representation. In MNs (conditional) independencies/dependencies are much more straightforward to compute through simple graph separation, something that stems from the undirected nature of MNs. Figure 2.3 illustrates an exemplar MN. It allows us to visualise and the concept of conditional independencies and some additional concepts which are the *Markov blanket* of a node and a *clique* in a MN. By observing Figure 2.3 we can say that the nodes of set A (X_2, X_3) are conditionally independent of the nodes of set B (X_6, X_7) given nodes X_4, X_5, i.e given a separating set D. To identify two conditionally independent sets, e.g. A, B, one has to check whether there is another set of nodes, e.g. D, which by completely removing it along with

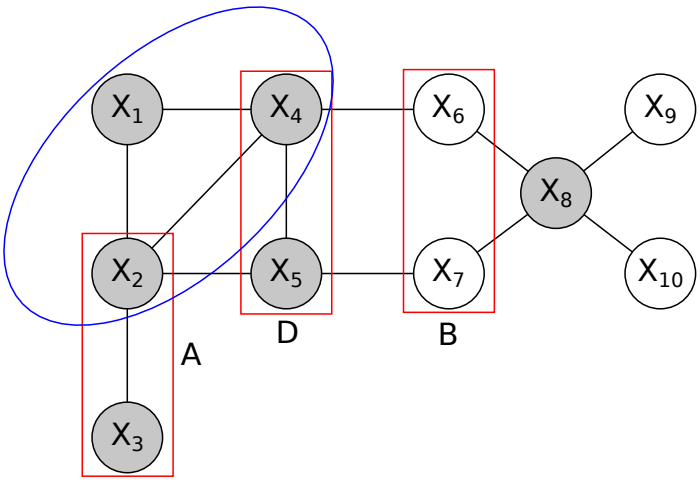


Figure 2.3. An exemplar Markov Network consisting of *ten* nodes ($X_1 - X_{10}$). Nodes of set A (X_2, X_3) are conditionally independent of nodes of set B (X_6, X_7) given a separating set of nodes D (X_4, X_5). Sets A, B, D are enclosed in red rectangles. Nodes X_6, X_7, X_9, X_{10} are the Markov blanket of node X_8. Finally, nodes who are enclosed in the blue ellipse (X_1, X_2, X_3) represent one of the cliques of the Markov network.

the edges connecting it to the rest of the network will cause the sets A, B to completely disconnect from each other. This is the case in our example and it is denoted as $A \perp\!\!\!\perp B \mid D$. This is also known as the global Markov property. In the event that A, B are not completely disconnected from each other then we say that A, B are conditionally dependent on each other given D. The same applies for single nodes. That is, any two non-adjacent (non directly connected) nodes are conditionally independent of each other given all other nodes and any two adjacent nodes are conditionally dependent on each other given all other nodes. A Markov blanket of a node X_i is the set of all neighbouring nodes as illustrated in Figure 2.3. One last concept to observe in Figure 2.3 is cliques. Nodes X_1, X_2, X_4 constitute a clique. A clique in a MN is a fully connected subgraph. Another clique in Figure 2.3 is formed by nodes X_2, X_4, X_5. Also, the node pair X_1, X_4 is also one of the cliques of the exemplar MN. The cliques defined by nodes X_2, X_4, X_5 and X_1, X_2, X_4 are also known as *maximal cliques*. A maximal clique is a clique in which if we include any other network node it will stop being a clique.

Let $X = X_1, X_2, \ldots, X_n$ be the set of all random variables in the network in a MN. The joint probability distribution of the random variables is given by:

$$P(X = x) = \frac{1}{Z} \prod_m \phi_{\mathrm{m}}(x_{\mathrm{m}}) \tag{2.10}$$

where, $P(X = x)$ is the joint probability distribution i.e the probability that the random variables in set X are in state (value configuration) x, the state of the random variables present in the mth clique is x_{m}, $\phi_{\mathrm{m}}(x_{\mathrm{m}})$ represents a potential function for the mth clique and Z is the partition function also called a normalization constant. The network has one potential function for each clique in the graph. A potential function is any non-negative function that takes a value for each state of the clique it represents. The choice for its form

should be such that it takes into consideration that its value for each clique state should reflect what we want to model in our network. Finally, the partition function Z is given by:

$$Z = \sum_{x} \prod_{m} \phi_{\mathrm{m}}(x_{\mathrm{m}}) \tag{2.11}$$

and it serves the purpose of normalizing the probabilities in the distribution so that they add up to *one*. A small example will help us understand the above concepts better.

Consider a MN with which we model the fact that speeding, fatigue and drinking can cause accidents while driving a car where random variables *Speeding*, *Fatigue*, *Drinking* and *Accident* are binary (*True*, *False*). Such a network is shown in Figure 2.4. To demon-

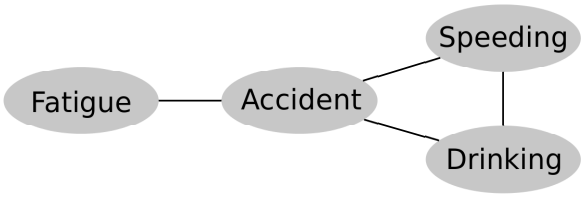


Figure 2.4. A Markov network representing the relations (dependencies) among random variables *Speeding*, *Fatigue*, *Drinking* and *Accident* in a car driving scenario.

strate how a potential function can be realized in the network of Figure 2.4 let us take into consideration the clique defined by nodes *Fatigue*, *Accident*. For this clique we are going to have one potential function with one value for each state of the clique (see Table 2.9). Higher values for the potential function in a clique state when compared to some other state

Fatigue	**Accident**	ϕ(**Fatigue**, **Accident**)
True	True	3
True	False	1
False	True	3
False	False	3

Table 2.9. States of the clique defined by nodes *Fatigue*, *Accident* and the values of its potential function for each state. Note that the form of the potential function is deliberately omitted to emphasize that the potential function can be any non-negative function.

of the same clique means that the higher valued state is more probable than the lower valued one. By observing Table 2.9 we can see that the state where someone drives under fatigue and does not have an accident is less probable than someone not driving under fatigue and having an accident. This is expected since driving tired is one of the major factors causing car accidents. Similarly we get the values for all states of the potential function for the clique defined by nodes *Speeding*, *Drinking* and *Accident*. Then from Equations 2.10, 2.11 we compute the joint probability distribution of the network.

One of the problems in MNs and indeed in Bayesian networks is the complexity of the graph. This is something that can heavily impact inference. In the case of a MN in particular, complexity is dependent on the sizes of the cliques in the graph. For small

cliques the problem is trivial but consider a case where in a MN we have many cliques comprising 15 nodes. Even in the binary case we have $2^{15} = 32768$ states for each such clique and consequently as many value calculations for the potential function. In order to mitigate the computational problem we can utilize a *log-linear* model to compute the joint probability distribution of a MN as follows [10]:

$$P(X = x) = \frac{1}{Z} \exp\Big(\sum_i w_i f_i(x)\Big) \tag{2.12}$$

where, $f_i(x)$ is called a state feature and can be any real-valued function of the state and w_i is the weight of the feature. In this manner we can have much fewer features than states. Let us now continue with our running example and replace the potential function ϕ with the following feature and weight:

$$f_1(Fatigue, Accident) = \begin{cases} 1, & \text{if Fatigue = False or Accident = True} \\ 0, & \text{otherwise} \end{cases} \tag{2.13}$$

$$w_1 = 1.1 \tag{2.14}$$

Now given the above we can reconstruct the same potential function values of Table 2.9 for all states of the clique using just one binary feature. By extension we can do the same for all cliques and finally calculate the joint probability distribution using Equation 2.12 instead of Equation 2.10. This concludes the description of MNs. Next we will be focusing on inference.

2.2.2.1 Inference

As in the case of BNs, inference in MNs can be either approximate or exact and can be either concerned with computing marginal,conditional and joint probability distributions of the random variables or with the most probable assignment of query variables. In the latter case we say that we perform MAP inference or MPE inference which is a special case of MAP inference (see Section 2.2).

Like in the case of BNs, the complexity of exact inference in MNs is related to the complexity of the graph structure of the network. Networks with many random variables and large cliques are the most challenging ones in terms of exact inference which is in the general case #P-complete. Algorithms such as enumeration and variable elimination that are applicable in BNs are also applicable in MNs. The junction tree algorithm [59] is also another alternative for exact inference in MNs. The intractability of exact inference in the general case makes it feasible only in the simplest of cases. Therefore, approximate inference approaches are more suited to more complex problems.

With respect to approximate methods, as in the case of BNs, MCMC is very popular in MNs. Within this spectrum, Gibbs sampling is one of the most famous algorithms for computing conditional and marginal probability distributions in MNs. Variational methods

[32] are an alternative to MC methods but are outside the scope of this thesis. Another approximate inference method is belief propagation, also called loopy belief propagation [60] due to the fact that MNs can contain cycles. The key concept in loopy belief propagation is the application of the sum-product algorithm [61] for calculating marginal probability distributions. Loopy belief propagation can suffer from convergence infeasibility or poor results [62] but for most cases it is a very efficient method. Regarding approximate MAP inference, iterated conditional modes [63] is one of the early approaches. A variation of loopy belief propagation based on the max-product algorithm can also be used for approximate MAP inference in MNs. Other approximate MAP inference methods are simulated annealing [64] and graph cuts [65] among others.

2.2.2.2 Applications

Markov networks are well suited models for expressing soft constraints between random variables. As such, they have been applied to many problems in a diverse group of scientific fields over the years.

One of the most prominent applications of MNs is in the field of computer vision. More specifically, MNs have been used for image segmentation [66, 67], image registration and matching [68, 69], optical flow analysis [70, 71], image denoising and reconstruction [72, 73] as well as for generating high resolution range images [74] among others. Data mining is another field in which MNs are applied to. Within this field we find the work presented in [75] where the authors model term dependencies and perform information retrieval in text via MNs. In [76], the authors present an MN-based approach for information retrieval via verbose queries. In [77], MNs are used as a modelling approach to mine what the authors call the (social) network value of customers which can be used for improved marketing of products or services. Social network analysis via MNs is also presented in [78, 79].

2.3 Hybrid Approaches

So far we have presented knowledge representation and reasoning approaches that are based on (pure) logic and probabilistic approaches (see Sections 2.1 and 2.2 respectively). Hybrid approaches on the other hand, merge both probabilistic and logic-based approaches into single knowledge representation and reasoning mechanisms aiming at benefiting from the advantages of both while abating their disadvantages. The scientific domain under which these approaches fall is known in the artificial intelligence and machine learning community as Statistical Relational Learning (SRL) [80].

Despite its name focusing on the learning aspect of such hybrid approaches, SRL is also concerned with reasoning and consequently decision making in application domains that exhibit both uncertainty and complex relational structure. The hybrid approaches of SRL can be mainly categorized with respect to the underlying probabilistic model that they utilise, i.e. Bayesian or Markovian, and secondly with respect to their underlying logic-

based model, i.e. first-order, propositional etc. Bayesian SRL approaches include Bayesian Logic Networks (BLNs) [81], Logical Bayesian Networks (LBNs) [82, 83], Probabilistic Relational Models (PRMs) [84, 85] and Multi-Entity Bayesian Networks (MEBNs) [86, 87, 1] among others. On the other hand, Markovian SRL approaches include Relational Markov Networks (RMNs) [88, 89] and Markov Logic Networks (MLNs) [10, 90–93] among others.

Statistical relational learning is a very active field of research within machine learning and artificial intelligence with a lot of real world applications. However, it is outside the scope of this section to provide a complete survey of the domain. This section rather focuses on MLNs which are presented and analysed in detail in Section 2.3.3. Multi-entity Bayesian networks and Bayesian logic networks (see Sections 2.3.1 and 2.3.2 respectively) are presented as two representative Bayesian SRL approaches but in less detail. The reason for doing so is that MLNs are the most commonly used and mature approach in the field and many other models such as MNs and BNs can be thought of as special cases of MLNs. An additional reason justifying this choice of ours is that as undirected models, MLNs are better suited for expressing soft constraints compared to directed models.

2.3.1 Multi-Entity Bayesian Networks

Multi-entity Bayesian Networks [86, 87, 1] are an extension of Bayesian networks that incorporates first-order logic into the probabilistic setting of Bayesian networks. Bayesian networks, despite being powerful enough to model a wide range of problems, often lack in expressiveness that would help tackle an even wider range of problems that require the relational expressive power of first-order logic. Furthermore, standard Bayesian networks are unable to be applied in problems where it is not possible to predefine the numbers, types and relationships among the entities that compose the problem [1]. In addition, a standard Bayesian network approach does not take into account that a random variable may need to depend on its value in the previous time-step. On the other hand, first-order logic has these abilities, but is not applicable in domains where uncertainty needs to be part of the representation and the reasoning process. MEBNs approaches benefit from the advantages of both worlds by merging them into a single mechanism in which logic and probability coexist and are supplementary to each other.

One of the most valuable features of Bayesian networks is the simplicity and comprehensiveness their DAG structure offers and MEBNs retain this advantage. The core structural block that is used by multi-entity Bayesian networks is called an MFrag. An MFrag, which stands for multi-entity Bayesian network fragment, contains a graph structure, as in a Bayesian network, that has random variables (nodes) and dependence relationships (directed edges) between them. An example of an MFrag modelling the level of danger in which a starship in a hypothetical science fiction scenario is exposed to is shown in Figure 2.5. Each MFrag expresses a conditional probability distribution over the instantiations of its *resident* nodes given the instantiations of *parent* and *context* nodes. More specifically,

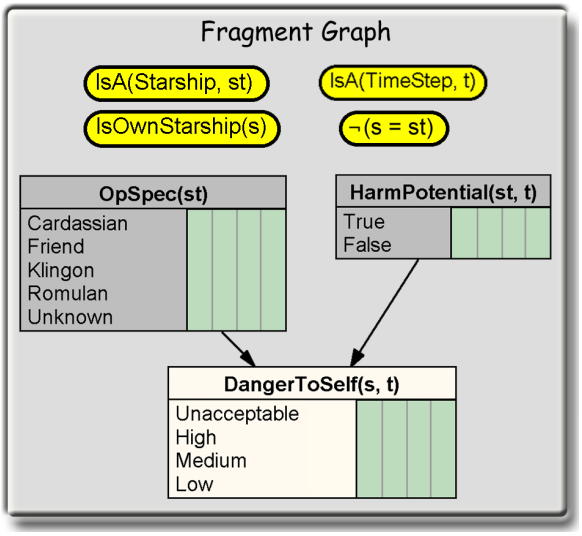


Figure 2.5. MFrag example modelling the level of danger in which a starship in a hypothetical a science fiction scenario is exposed to. Nodes in yellow represent context nodes. Nodes in dark grey are input nodes while the node in light grey is a resident node. The figure is taken from [1].

resident nodes represent conditional probability distributions given parent nodes while context nodes, which are logical boolean constraints, have the role of conditions that need to be satisfied for both dependencies and conditional probability distributions to be applicable. MFrags can be thought of as candidate templates which once instantiated, are combined in what is called MTheories. An MTheory is nothing more than a joint probability distribution over the random variables represented by the MFrags forming it. In this manner we can model different scenarios by having different MFrags forming different MTheories all in one network. In order to compute any query with respect to an entity (random variable) a Situation-Specific Bayesian Network (SSBN) is constructed. That is, a minimal BN for answering a query [94] using the information provided. A good introductory tutorial on MEBNs can be found in [1].

2.3.2 Bayesian Logic Networks

Bayesian Logic Networks [81] have emerged as an approach focused on practical applicability to real world problems. As such, they provide enhanced expressiveness when compared to MEBNs since they are able to encode global first-order logical constraints as opposed to MEBNs and local logical constraints in the form of MFrag context nodes. Like MEBNs, BLNs are templates for constructing ground networks but in the BLN case the ground network is a mixed network as opposed to a ground BN in the case of a MEBN. Mixed networks [95] are extensions of BNs containing deterministic (logical) constraints. In practice a mixed network comprises a probabilistic and a deterministic component.

More specifically, a BLN consists of a set of *fundamental* declarations, a set of *fragments* of conditional probability distributions and a set of *formulas* in first-order logic. The fundamental declarations of a BLN include the types of the abstract entities that are to

appear in the network, the actual entities and the signatures of functions present in the network. In this context, logical predicates are declared as logical boolean functions (e.g. *logical boolean takesPartIn*$(person, course)$ is a logical predicate indicating that an abstract entity of type person takes part in an abstract entity of type course). On the other hand, functions that are used for instantiation of random variables are declared as random (e.g. *random gradeDomain grade*$(student, course)$ is a function signature that maps from a student-course pair to the grade domain [81]). A fragment essentially defines a dependency of a random variable (e.g. $grade(student, course)$) on other random variables which are the parents while a first-order logical formula is created by a combination of logical boolean functions. Fragments can be thought of as templates for the probabilistic component of the ground mixed network while logical formulas as templates for the deterministic component. In order to perform inference in BLNs we can apply standard algorithms that are applicable in mixed networks [95, 96].

2.3.3 Markov Logic Networks

Markov Logic Networks [10] are powerful representations that unify the worlds of logic and probabilistic graphical models in a way that exploits the advantages of both. Namely, first-order logic provides the means for expressing knowledge and reasoning about structured information while Markov networks handle uncertainty in a probabilistic manner.

A Markov logic network is based on a first-order knowledge base. A first-order knowledge base describes a set of possible worlds for which none of the formulas in the knowledge base are violated [97]. All other worlds are impossible. This is effectively a set of hard constraints that define the set of possible worlds. The rationale behind Markov logic is to soften hard constraints. Consequently, when a world violates a formula then it becomes less probable instead of impossible. Suppose we wanted to express that an employee is not an employer with a first-order logic formula: $\forall x\, Employee(x) \Rightarrow \neg\, Employer(x)$. In first-order logic as soon as there is a person that is an employee and an employer at the same time[13] then the world that represents such a situation becomes impossible. First-order logic makes no distinction between there being one employee that is also an employer and all of the employees being also employers. This is of course not what holds in reality since the latter might be extremely less probable than the first but still possible. Markov logic networks address this problem by attaching real valued weights to first-order logic formulas. Formulas with negative weights can always be replaced by their negations as long as they are associated with the opposite of their initial negative weights, i.e. positive weights. As such, higher weights impose stronger constraints while lower weights impose weaker constraints. Therefore, worlds that violate formulas with high weights are penalised more and the more such formulas are violated the more such worlds are penalised. Weights that are positively infinite ($+\infty$) impose an unbreakable constraint, that is, formulas with such weights represent pure first-order logic formulas. Let us now define Markov logic networks

[13] Someone working for a company that he also owns makes him both an employer and an employee.

in a formal manner [10]:

Definition 2.1 *A Markov logic network L is a set of pairs (F_i, w_i), where F_i is a first-order logic formula and w_i is a real valued weight attached to it. Along with a finite set of constants $C = \{c_1, c_2, c_3, \ldots, c_{|C|}\}$, it defines a ground Markov network $M_{L,C}$ as follows:*

1. *$M_{L,C}$ contains one binary node for each possible grounding of each literal appearing in L. The node value is 1 if the ground literal is true, and 0 otherwise.*

2. *$M_{L,C}$ contains one binary feature for each possible grounding of each formula F_i in L. The value of the feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i which is attached to formula F_i in L.*

To demonstrate how a Markov logic network is realized and how a ground MN is created let us expand on the the employer-employee example from the beginning of this section. We add a first-order logical formula in Markov logic which expresses that an employer x does not work for someone else y: $Employer(x) \Rightarrow \neg\, WorksFor(x, y)$. Finally we complete our knowledge base by adding a formula that represents that if someone x, is part of a company c and that someone x works for someone else y then y is part of the same company c: $Company(c, x) \wedge WorksFor(x, y) \Rightarrow Company(c, y)$. All formulas are assigned some weight to express how strong the constraint they represent is as seen in Table 2.10. Let George, John and MicroGalaxy be three constants for the KB shown in Table 2.10,

Weights	**Formulas**
0.9	$Employee(x) \Rightarrow \neg\, Employer(x)$
1.7	$Employer(x) \Rightarrow \neg\, WorksFor(x,y)$
2.5	$Company(c,x) \wedge WorksFor(x,y) \Rightarrow Company(c,y)$

Table 2.10. Markov logic employer-employee example knowledge base. The higher the weight the stronger the constraint. All formulas are considered to be universally quantified.

where George, John are people and MicroGalaxy is a company. Now, the ground MN of the aforementioned constants and our KB is the one shown in Figure 2.6. The resulting

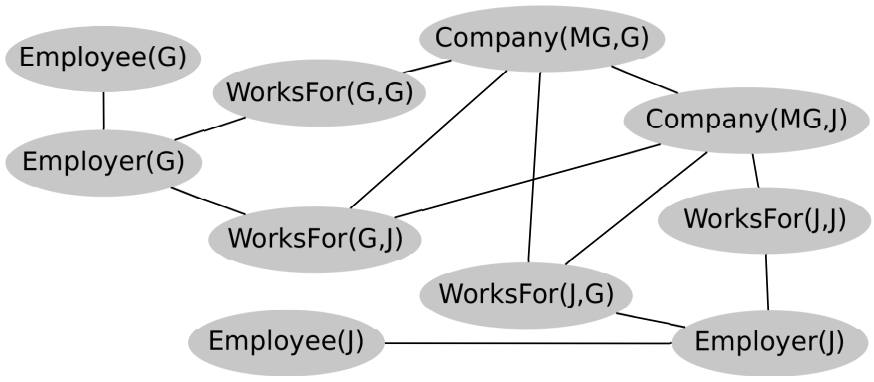


Figure 2.6. Ground Markov network of the employer-employee running example. Constants George, John and MicroGalaxy are abbreviated as G, J and MG respectively.

ground MN comprises all the possible ground literals in the domain which are the nodes in the resulting network. The edge connection between nodes is decided based on ground clauses. That is, ground literals (nodes) that are part of a clause grounding are connected by an edge. Formulas in MLNs follow the syntax of first-order logic (see Section 2.1.2). As in the case of MEBNs and BLNs, MLNs are templates, but for constructing ground MNs. Depending on the constants as well as their number (quantity) these templates can yield different ground MNs of different sizes and/or structure [10]. Now, since MLNs are templates for constructing ground MNs, from Equations 2.10, 2.12 and Definition 2.1 it follows that the probability distribution of possible worlds x represented by the ground Markov network $M_{L,C}$ is given by [10]:

$$P(X = x) = \frac{1}{Z} \exp\Big(\sum_i w_i n_i(x)\Big) = \frac{1}{Z} \prod_i \phi_{\mathrm{i}}(x_{\mathrm{i}})^{n_i(x)} \tag{2.15}$$

where, $n_i(x)$ is the number of true groundings of F_i present in x, x_i are the truth values, which is practically the state, of the atoms in F_i and $\phi_{\mathrm{i}}(x_{\mathrm{i}}) = e^{w_i}$. Recall that we talked about possible worlds in Section 2.1.

Finally with respect to semantics, MLNs follow the semantics of first-order logic but make the *unique names* and *domain closure* assumptions as well as the *known functions* assumption which dictates that for each function in the MLN, the value of the function applied to every possible tuple of arguments is known and is a an element of the finite set of constants. The first two assumptions are the alternative semantics assumptions that we described in Semantics of Section 2.1.2. This comes as no surprise since MLNs are a combination of Markov networks and first-order logic.

2.3.3.1 Inference

Inference in MLNs can be described in the same terms as in the case of PGMs and first-order logic. Again, this comes as no surprise since as we mentioned in the previous section, MLNs combine both worlds benefiting from their advantages. This practically creates the necessity of combining probabilistic inference algorithms with the ones used for first-order inference so that they can be applicable in this new (combined) setting. With respect to computing the probability that some first-order formula F_1 holds given some other first-order formula F_2, a Markov logic network L and a set of constants C is given by [98]:

$$P(F_1|F_2, M_{L,C}) = \frac{\sum_{x \in X_{F_1} \cap X_{F_2}} P(X = x|M_{L,C})}{\sum_{x \in X_{F_2}} P(X = x|M_{L,C})} \tag{2.16}$$

where $M_{L,C}$ is the ground MN, X_{F_1} is the set of worlds where F_1 holds, X_{F_2} is the set of worlds where F_2 holds and $P(X = x|M_{L,C})$ is calculated based on Equation 2.10. Now, in order to compute the most probable state of a world y given some evidence x (MAP/MPE

inference) [98] we use the following:

$$arg\ max_y P(y|x) = arg\ max_y \sum_i w_i n_i(x,y) \tag{2.17}$$

which follows from Equation 2.15.

Recall that exact probabilistic inference in MNs is #P-complete while exact inference in first-order logic is NP-complete. Therefore computing Equation 2.16 exactly is generally intractable but for the simplest of cases. An algorithm that does so is enumeration-ask. With respect to approximate inference recall that Gibbs sampling is one of the most common MCMC algorithms in PGMs. In the combined setting of probabilistic and first-order inference though we would face challenges if we were to apply Gibbs sampling as is. First, deterministic dependencies that are present in logical representations would make Gibbs sampling and any MCMC method to output the wrong probabilities. This is due to the fact that a deterministic dependency would affect the sampling process in a way that it would get stuck in a portion of the state space and yield wrong estimations. Even in the case of near deterministic dependencies MCMC methods demonstrate reduced performance and are very slow. To address these problems, Poon and Domingos in [99], combined Gibbs sampling with ideas from SAT solvers (see Inference in Section 2.1.1) lifted for first-order logic (see Inference in Section 2.1.2) to create MC-SAT. More specifically, MC-SAT is a combination of Gibbs sampling and the SAMPLESAT algorithm [100]. Algorithm 4 illustrates MC-SAT in pseudocode. MC-SAT operates on a ground MN where, $x^{(i)}$ is the

Algorithm *MC-SAT(clauses, weights, num_samples)*
Input: *clauses*, first-order clauses from a first-order KB in CNF
weights, weights of the clauses
num_samples, number of times the algorithm will perform sampling

$x^{(0)} \leftarrow$ Satisfy(hard *clauses*)
for $i \leftarrow 1$ to *num_samples* **do**
  $M \leftarrow \emptyset$
  for all $c_j \in$ *clauses* satisfied by $x^{(i-1)}$ **do**
    With probability $1 - e^{-w_j}$ add c_j to M
  end
  Sample $x^{(i+1)} \sim U_{SAT(M)}$
end

Algorithm 4: The MC-SAT algorithm for approximate probabilistic inference in Markov logic networks [99].

current state of the network. The initial state $x^{(0)}$ of the network is the outcome of applying a SAT solver to all hard clauses. That is, clauses with an infinite weight which, if you recall, represent deterministic constraints. If at this step the hard clauses are not satisfiable then the algorithm's output is undefined [99]. M is a random subset of currently satisfied clauses that need also to be satisfied in the next network state. In the beginning of each sampling

iteration i we set M to $\emptyset$. In addition, in every sampling iteration i, we perform a nested iteration over all ground clauses c_j that belong to the set of clauses that were satisfied in the previous network state[14] $x^{(i-1)}$ and with probability $1-e^{-w_j}$ we add c_j to M. After the nested iteration is complete we sample the next state $x^{(i+1)}$ as a uniform sample from $SAT(M)$ which is a set of states that satisfy M. This completes a full sampling iteration. The algorithm terminates after *num_samples* iterations. MC-SAT is to date one of the most widely used algorithms for approximate probabilistic inference in the MLN domain.

Before proceeding to approximate MAP/MPE inference we would like to point out an exact approach which is performing MAP/MPE inference by converting the ground MN to a Weighted Constraint Satisfaction Problem (WCSP) and using the toulbar2 WSCP solver [101]. Now, with respect to approximate MAP/MPE inference, the most widely used algorithm is MAXWALKSAT [102]. MAXWALKSAT is a variant of lifted WALKSAT that solves weighted satisfiability problems, something very convenient for Markov logic where we have weighted clauses and formulas. Algorithm 5 illustrates MAXWALKSAT in pseudocode [98]. Given a set of weighted clauses L, a maximum number of tries *max-tries*, a maximum number of flips *max-flips*, a target solution cost *target-cost*, and a probability of taking a random step p, MAXWALKSAT finds a solution (a truth assignment of clause variables for which the cost is smaller or equal to the target solution cost) or returns the best "solution" found. The algorithm iterates for *max-tries* times and every time it performs a random truth assignment to the variables of the weighted clauses which becomes the *solution*. Then it calculates the *cost* of this *solution* by summing over the weights of the unsatisfied clauses in it. The next step is to execute variable flips for *max-flips* times. If the *cost* $\leq$ *target-cost* then it returns the *solution*. If not, it chooses an unsatisfied clause at random and calculates a uniform deviate from the interval $[0,1]$. If *deviate* $<p$ then for each variable v in the previously randomly chosen unsatisfied clause it computes the "change in the sum of weights of unsatisfied clauses that results from flipping v in the current *solution*" (DeltaCost(v)). Then, v with the lowest DeltaCost(v) is flipped in the current *solution*, which essentially becomes the new current *solution*. Finally, the *cost* of the new current *solution* is increased by DeltaCost(v_f) where, v_f is the flipped v from the previous step. If not *cost* $\leq$ *target-cost* during *max-tries* and *max-flips* iterations then, the algorithm returns failure as well as the best *solution* computed up to that point.

In the category of offering improved inference speed, we find the work presented in [103]. The approach is called BAM. BAM, which stands for Break and Match inference, is a meta-inference algorithm which aims at speeding up existing inference algorithms such as MC-SAT (in theory any algorithm can be used) by clustering query literals. In the same spirit (speeding-up inference), we find the approach of Fast Reduction of Grounded networks (FROG) [104]. FROG can be applied irrespective of whether we are interested in MAP/MPE or probabilistic inference since it is concerned with reducing the size of the ground network. To conclude, some scientific effort has been made in the direction of parallelizing inference [105, 106].

[14] the previous network state $x^{(i-1)}$ was calculated by the previous sampling iteration $i-1$.

```
Algorithm MAXWALKSAT(L,max-tries,max-flips,target-cost,p)
  Input: L, a set of weighted clauses
         max-tries, the maximum number of tries
         max-flips, the maximum number of flips
         target-cost, the target solution cost
         p, the probability of taking a random step

  vars ← variables in L
  for i ← 1 to max-tries do
      solution ← random truth assignment to vars
      cost ← ∑weights of unsatisfied clauses in solution

      for j ← 1 to max-flips do
          if cost ≤ target-cost then
              return "Success, the solution is:" solution
          end
          c ← random unsatisfied clause
          if Uniform(0,1) < p then
              v_f ← random variable from c
              else
                  for each variable v in c do
                      compute DeltaCost(v)
                  end
                  v_f ← v with lowest DeltaCost(v)
              end
          end
          solution ← solution with v_f flipped
          cost ← cost+ DeltaCost(v_f)
      end
  end
  return "Failure, the best truth assignment is:" best solution found
```

Algorithm 5: The MAXWALKSAT algorithm for approximate MAP/MPE inference in Markov logic networks.

2.4 Summary and Discussion

This chapter was concerned with presenting a broad field of the Artificial Intelligence (AI) discipline which is knowledge representation and reasoning. Conceptually we divided the chapter into three main sections in order to present logic-based, probabilistic and hybrid approaches respectively.

Regarding propositional logic, it is a basic logic representation language which lacks the expressive power required for real world applications since it cannot encode relations between objects and consequently it cannot capture the complexity of real world domains. Representation of relations between objects is possible with the use of first-order logic which has much richer semantics than propositional logic that utilises propositions. Inference procedures of propositional logic are applicable in first-order logic but we first need to propositionalize the first-order KB which makes inference slow. An alternative is to lift

propositional inference to the first-order paradigm. In this manner, forward and backward chaining can also be used for first-order logic. Backward chaining is also used in the logic programming paradigm of inference in first-order logic. Prolog is the most commonly used logic programming language which employs the database semantics of first-order logic. That is, the unique names, closed world and domain closure assumptions of first-order logic. Inference in first-order logic is sound and complete and logical consequence (entailment) is semidecidable. With respect to description logics, they are a family of logical languages that are subsets of first-order logic and compared to first-order logic some of them are also decidable. Description logics are the basis for the OWL DL language which is a language for authoring ontologies (see Chapter 3). There are also other types of logic for representing knowledge such as Fuzzy logic [107] and Modal logic [108] but they are outside the scope of this thesis.

Regarding probabilistic approaches, they provide the means for representing and reasoning about uncertain knowledge as opposed to logic-based representations which do not deal with uncertain knowledge explicitly. In this chapter we presented two main probabilistic graphical models namely, Bayesian and Markov networks, which are directed and undirected probabilistic models respectively. Bayesian networks are well suited for expressing causal links between random variables while Markov networks are better suited for expressing soft constraints between random variables. For both Bayesian and Markov networks we analysed their building blocks, looked into several inference approaches (exact and approximate) and provided information on their applications in diverse fields of the scientific spectrum.

Further to the logic-based and probabilistic approaches we have also presented the domain of statistical relational learning which is concerned with hybrid approaches, i.e. approaches that unify the world of logic and probability (uncertainty) into a single representation. In this manner, we can benefit from the advantages of both worlds. That is, on the one hand having the means for encoding complex relations among entities which stems from the logic aspect of hybrid models while at the same time being able to encode uncertain knowledge explicitly in a well defined and structured manner. In this category we presented multi-entity Bayesian networks and Bayesian logic networks which are representative Bayesian-based hybrid approaches and Markov logic networks which are a representative Markov-based hybrid approach. However, we focused on Markov logic networks as being the most mature line of research and due to the fact that both multi-entity Bayesian networks and Bayesian logic networks can be thought of as special cases of Markov logic networks. We have also presented various inference procedures (algorithms) both for exact and approximate inference in the Markov logic setting.

Chapter 3

Knowledge Engineering with Ontologies

Knowledge engineering is a scientific field residing within the artificial intelligence discipline and is concerned with developing knowledge-based systems that aim at solving real world problems. As such, knowledge engineering provides the means to represent, organise and manage large amounts of knowledge in a structured and well defined manner.

Ontologies are formal knowledge representation mechanisms that capture relational knowledge and semantics about domains of interest. The term ontology first appeared in philosophy denoting the study of what exists, what are the attributes and features of things that exist and how they relate to each other. Over the years the term crossed the borders of philosophy and was adopted by a wide range of fields, including but not limited to Information Technology (IT), Artificial Intelligence (AI), Computational Linguistics (CL) and Mechanical Engineering (ME) [109]. The diversity of the aforementioned fields has led to the absence of a standardized definition for an ontology. However, the most wide spread is the one given by Thomas Robert Gruber in [110]: An ontology is a"formal, explicit specification of a shared conceptualization of a domain". In an ontological approach for knowledge engineering, logic-based representations and reasoning are the natural choice. First of all, because logic-based representations are ideal for representing relational knowledge and capturing interactions between entities. Secondly, because logic-based systems can be very expressive depending on the kind of logic used. Finally, because logic-based representations are governed by well defined syntax and semantics and supported by a variety of well researched inference approaches (see Section 2.1). When developing ontologies we agree on a vocabulary and a taxonomy based on which we represent and structure knowledge.

3.1 Languages for Authoring Ontologies

To date, ontologies have mostly been used in the Semantic Web. An initial approach towards expressing ontologies for the Semantic Web was based on the RDF [111] and the RDF Schema (RDFS) [112] languages. These languages were though found to be of limited expressiveness so the World Wide Web Consortium (W3C) recommended OWL as a knowledge representation language for Semantic Web ontologies as of 2004 [113]. The lan-

guage was designed in a way that it provides maximal compatibility with RDF and RDFS and its normative syntax is RDF/XML[1] with formal semantics[2]. OWL comes in three variants: OWL Full, OWL DL and OWL Lite [114]. OWL Full is the most expressive of the three but is undecidable. OWL DL (Description Logics) is based on the $\mathcal{SHOIN}^{(\mathcal{D})}$ family of DL languages (see Section 2.1.3). OWL DL provides a very good balance between expressiveness of first-order logic and very desirable features such as soundness, completeness and decidability. OWL Lite is based on the $\mathcal{SHIF}^{(\mathcal{D})}$ family (see Section 2.1.3) and is the least expressive of the three and is intended for users that mainly require simple constraint features and a classification hierarchy. More information on the three variants of OWL can be found in [113]. Figure 3.1 illustrates how the three different OWL variants relate to each other in terms of expressive power.

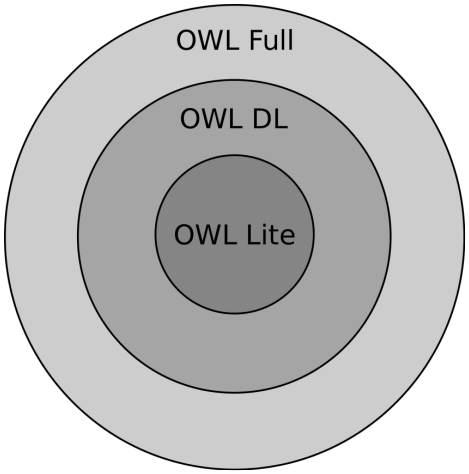


Figure 3.1. Relation of the OWL variants in terms of their expressive power. OWL Lite is a subset of OWL DL while both are subsets of the OWL Full variant.

As of October 2009, W3C recommended OWL 2 [115] as a knowledge representation language for ontologies that is based on the $\mathcal{SROIQ}^{(\mathcal{D})}$ family of languages. The primary syntax for OWL 2 is RDF/XML. Alternative syntaxes for OWL 2 are the OWL 2 XML[3], OWL 2 Manchester Syntax[4] and Turtle[5]. OWL 2 semantics are expressed in two ways. The first is the so called direct semantics[6] and the second the RDF-based semantics[7]. Ontologies that are 'interpreted' with the direct semantics of OWL 2 are sometimes referred to as OWL 2 DL ontologies while ontologies (RDF graphs) 'interpreted' with RDF-based semantics are sometimes referred to as OWL 2 Full ontologies. OWL 2 has also three additional variants (sub-languages) called profiles which are OWL 2 EL, OWL 2 QL and OWL 2 RL that are intended for different application scenarios [115]. An analysis of the complexity of OWL 2 and its sub-languages in comparison to OWL DL is given in [116].

[1] http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/
[2] http://www.w3.org/TR/2004/REC-owl-semantics-20040210/
[3] http://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/
[4] http://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/
[5] http://www.w3.org/TR/turtle/
[6] http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/
[7] http://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/

Another language used for ontological representations is the Knowledge Interchange Format (KIF) that was designed and developed in order to promote the interchange of knowledge among disparate computer systems. There are many KIF versions but probably the most well known is SUO-KIF [117] that was used to define the SUMO ontology [118] which is an upper ontology that contains various domain specific ontologies. The KIF language has declarative semantics which means that the semantics of expressions in KIF do not require an interpreter in order to be understood. KIF is also logically comprehensive which means that it supports arbitrary sentences expressed in first-order logic. As a full first-order language KIF presents the inherent advantages and limitations of full first-order logic. Additionally, also very expressive representations include, the Scone KB project [119], the CycL language [120] and Topic Maps [121] but they lack efficient reasoning approaches [2]. Yet another approach to ontologies is LOOM. The LOOM system provides the LOOM knowledge representation language and environment and it was developed in the University of Southern California. In LOOM knowledge is represented in terms of definitions, rules, default rules and facts. A deductive engine that utilizes forward chaining is used by LOOM to compile knowledge into a network that supports deductive query processing [122]. F-logic [123] is a language that integrates object-oriented programming with logic. F-logic extends classical predicate calculus to incorporate types, classes and objects based on the object-oriented programming paradigm.

We believe that the usage of the OWL family of languages is the best choice for the project due to the plethora of OWL ontologies developed so far. In this way we can leverage from all the relevant successful work and apply the same principles in our domain. Especially well designed and implemented robotic knowledge representation and processing systems such as KnowRob [2] and the conceptual closeness of our work to them have shifted our interest towards OWL DL (see Section 3.3.1 for more information on KnowRob). Another reason for this choice is purely based on reasoning. In OWL DL along with retaining much of the expressiveness we also have soundness, completeness and decidability upon applying reasoning.

3.2 Components of OWL DL Ontologies

OWL DL ontologies comprise three components through which modelling of domains of interest is achieved, namely, *classes*, *individuals* and *properties*. It is important at this point to emphasize for clarity that these components are not unique to OWL DL but are shared with the other two OWL variants, that is, OWL Lite and OWL Full. In the remainder of this section we will be describing the aforementioned components in detail. In addition, in the context of OWL DL, we will be explaining their connection to Description Logics (DLs) notions as they were presented in section 2.1.3.

Classes: An ontology *class* is a representation of a set of entities that exist in a domain and share common characteristics that allow them to be grouped together. Ontology classes are

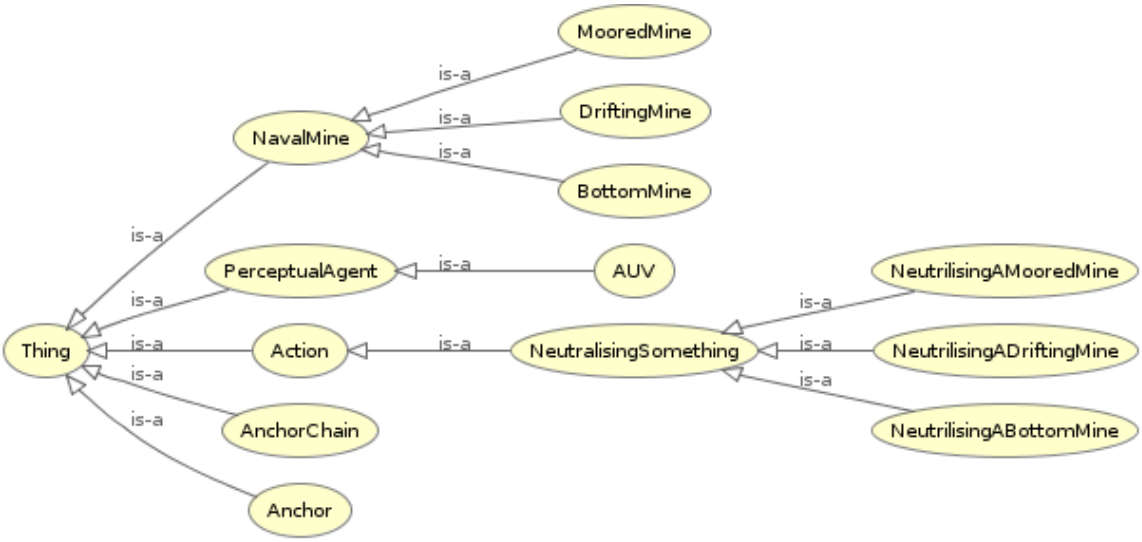


Figure 3.2. Ontology classes of entities that can be found in the hypothetical underwater MCM domain organised in a taxonomy. The *is-a* notation denotes a superclass-subclass connection between classes. *Thing* is a special class that plays the role of the root class (root superclass) of all classes in a domain. This is analogous to root nodes found in tree data structures.

very much like classes in the context of object oriented programming. The notion of a class is equivalent to the notion of a concept as this was described in Section 2.1.3. Let us replicate the example given in the aforementioned section in which an underwater vehicle needs to locate and neutralize sea mines as part of an MCM mission. In this context, ontology *classes* such as *PerceptualAgent*, *NavalMine*, *MooredMine*, *AUV*, *NeutralisingSomething* and *NeutralisingAMooredMine* are classes that could be used for modelling the hypothetical domain. Ontology classes are organized into a *hierarchy* in the form of superclass-subclass occurrences (subclass *is-a* superclass) which is also known as a *taxonomy*. Consequently, an *AUV is-a PerceptualAgent*, a *MooredMine is-a NavalMine* and *NeutralisingAMooredMine is-a NeutralisingSomething*. Classes should not be thought of only as sets comprising physical entities but also as sets that can represent non-physical entities as evidenced by classes such as *NeutralisingAMooredMine* which refers to an action. Figure 3.2 provides a graphical representation of the classes discussed above as well as additional hypothetical domain classes that could be used, organised in a taxonomy, as they are rendered in Protégé[8], an open source knowledge management system and ontology editor.

Individuals: *Individuals* are the individual entities (instantiations) of ontology classes. Individuals are also referred to as *instances*. Again, ontology individuals are very much like class instances in the context of object oriented programming. Now consider the *MooredMine*, *Anchor* and *AnchorChain* concepts. Individuals of these concepts would be the actual moored mines, anchors and anchor chains present in the domain (e.g. *MooredMine1*) for an individual in the first case, *Anchor1* for an individual in the second case, etc. If for in-

[8]http://protege.stanford.edu

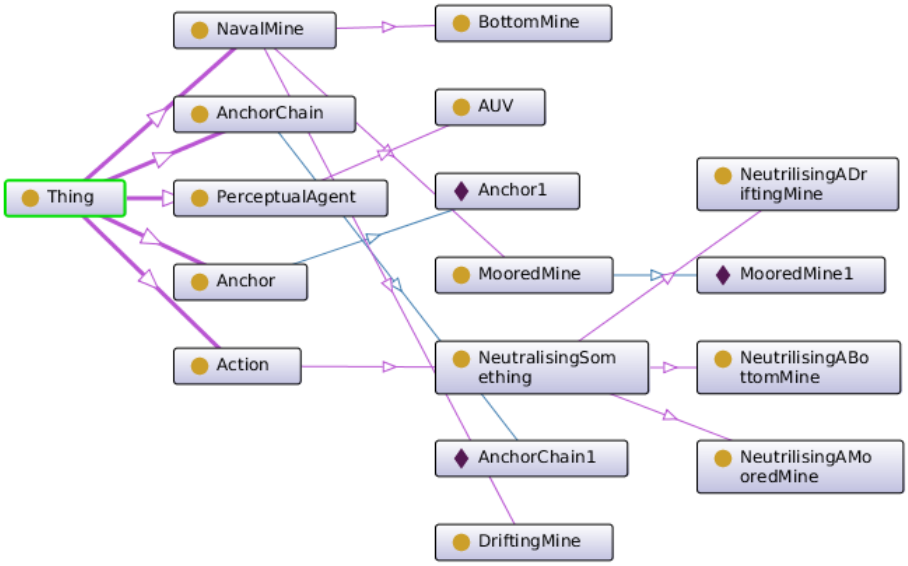


Figure 3.3. Ontology individuals as well as their respective classes. Individuals can be identified by the purple diamond preceding their name while classes can be identified by the yellow circle. Blue arcs link individuals to the classes they belong while purple arcs link classes to each other forming superclass-subclass connections. *MooredMine1*, *Anchor1* and *AnchorChain1* are individuals (instantiations) of their respective classes.

stance more moored mines were to be present in the domain then we would have additional instances of the concept *MooredMine* and so on. Figure 3.3 illustrates such individuals as they are rendered in Protégé. The taxonomy presented in Figure 3.3 is the same as the one presented in figure 3.2 with the addition of individuals. Moreover we have used a different visualization tool within Protégé that enables the user visualise the connections between the various building blocks of the ontology.

Properties: *Properties* are the last component found in OWL DL ontologies. Properties in OWL DL are equivalent to *roles* found in DLs. There are two types of properties in OWL DL, namely, *object* and *data* properties. Both object and data properties can be used in class definitions to restrict their domain and range to individuals with certain attributes (see Section 2.1.3). Moreover, object properties can be used to link two individuals forming *binary relations*. Finally, data properties can be used to link an individual to data values[9] such as strings, literals, numeric values etc., acting in this manner as attributes. Figure 3.4 illustrates an object property between two individuals forming a binary relation.

The analogies between DLs and OWL DL do not come as a surprise since OWL DL is part of the DLs family of languages. Recall from Section 2.1.3 that when using DLs the available knowledge resides in the DL KB which comprises the TBox and ABox. In the same manner, the available knowledge in ontologies authored in OWL DL resides in the so called OWL DL KB which also comprises the TBox and ABox. The TBox contains all classes as well as class definitions through restrictions while the ABox contains all

[9]Strictly speaking a link between an individual and a data value is also a *binary relation* between them but we reserve this term for when describing relations between individuals.

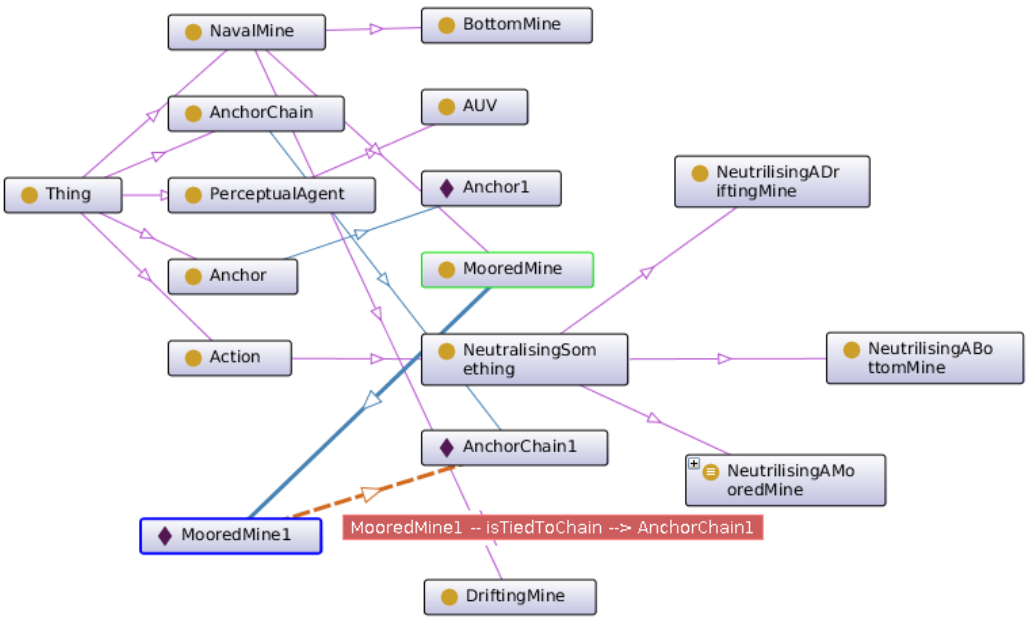


Figure 3.4. Object property *isTiedToChain* forms a binary relation (orange arc) between individuals *MooredMine1* and *AnchorChain1*. Blue arcs link individuals to the classes they belong while purple arcs link classes to each other forming superclass-subclass connections. The information illustrated in this figure was produced by manually creating the *isTiedToChain* object property inside the ontology illustrated in Figure 3.3 in order to form a binary relation.

individuals with their attributes as well as all binary relations between them.

3.3 Ontologies in Robotics

Over the last few years the idea of utilizing ontologies for robotic applications as a way of promoting autonomy has naturally become appealing to the robotics domain. The utilization of autonomous robots for air, land and sea operations has shown that the quality of knowledge representation and reasoning is equally important to the quality of sensor data for a successful task execution. Poor knowledge representation and structuring leads to poor inference outcomes and ultimately failure. Ontologies can help solve this problem through their inherent ability to capture relational knowledge and to provide metadata upon which reasoning can be extended and consequently cognition can be enhanced. In that spirit, the remainder of this section is dedicated to the applications of ontologies in robotics.

Matheus et al. in [124] present a generic ontology-based information fusion system for enhancing situation awareness. One of the scenarios examined is the maintenance of reliable situation awareness in battlefield. Their findings prove the viability of the approach as well as the flexibility that an ontology-based system can offer to an end user performing queries regarding the state of the world.

The use of ontologies in robotics is increasing, as evidenced by the mature and sophisticated KnowRob system [2, 125], an ontology-based knowledge representation and knowledge processing framework for indoor robots. KnowRob will be described in detail

in Section 3.3.1. A similar framework is the ORO system [126], which leverages a core robotics ontology to integrate data from diverse sources such as sensors, domain knowledge, and human input, and to provide useful, structured knowledge that can help the robot in interactions with humans. An ontology for general situation awareness is given by [124], with applications to military robotics, and [127] create a general-purpose ontology for specifying robotic domains.

A particularly relevant work is [128], which describes a planning and control system for AUVs built on an ontology. Planning and mission data, vehicle capabilities, and vehicle status information are all stored in the ontology, which allows for much more reactive and flexible planning. The authors describe an in-water test where the planning system was able to re-plan the mission to deal with component failures thanks to semantic knowledge encoded in the ontology. Other benefits of using an ontology are the ease of using a reasoning system (for example for problem diagnosis), and the ease of sharing data between modules on the vehicle.

3.3.1 The KnowRob System

KnowRob [125, 2] is both a knowledge representation and a knowledge processing system. The core of the system is a series of OWL DL ontologies. In addition, KnowRob incorporates inference algorithms, mechanisms for dynamic computation of knowledge, a framework for executing actions and interfacing to robot sensors, a ROS interface, and interfaces to other external modules for tasks such as probabilistic reasoning and knowledge extraction. KnowRob was developed with household assistant robots in mind, but the system is generic and flexible enough to be utilized in any robotic context.

For representing knowledge KnowRob uses ontologies expressed in OWL DL (Description Logics), as described in Sections 2.1.3 and 3.2. KnowRob includes a core ontology which is based on the OpenCyc ontology [129], with extensions for the robotics domain. While some of these extensions are more useful for household robots (such as *FoodOrDrinkOrIngredient*), many are more general (such as *Vector*).

KnowRob is implemented mostly in SWI Prolog [130], together with some C++ and Java code (although custom modules that use the KnowRob API must be written in Prolog). Using Prolog, KnowRob loads OWL files (ontologies) into the system and stores them as triples of the form `rdf(Subject, Predicate, Object)`. OWL files are parsed using the Thea OWL parser [131, 132], which itself uses SWI-Prolog's Semantic Web library [133] for parsing the RDF/XML syntax into RDF triples. Thea then builds a representation of the OWL ontology meaning the OWL ontology syntax is represented as Prolog terms that form Prolog predicates. Predicates can be used to access knowledge residing in the Prolog KB by executing them as queries (e.g. `owl_has(A, rdf:type, subsea:'MooredMine').`). The example predicate tells KnowRob to look for all instances of the concept *MooredMine* which is defined in the *subsea* ontology and assign it to variable *A* successively. Prolog predicates are not only used for accessing knowledge in the KB but are also used for in-

terfacing the KB with external data and for implementing KnowRob reasoning modules which can be extended based on application needs. This extensibility is a major advantage of KnowRob, meaning it can be adapted to the requirements of many robotic projects.

Another benefit of KnowRob is that not all knowledge needs to be immediately available in the knowledge base: a special class of Prolog predicate called a *computable* can be defined that dynamically queries or calculates a result. This can be used, for example, to query the perception system for the current value of a variable, or calculate the relationship between two objects using their positions. In this way, the world plays the role of a virtual knowledge base, meaning that correct information is always used, and the on-demand computation of knowledge reduces the computational load. Further, the knowledge base can grow and change at run-time. In contrast to many logic-based systems where all available knowledge must be inserted into the knowledge base at the start of the inference process to enable all possible queries of interest to be answered, in KnowRob this is not required.

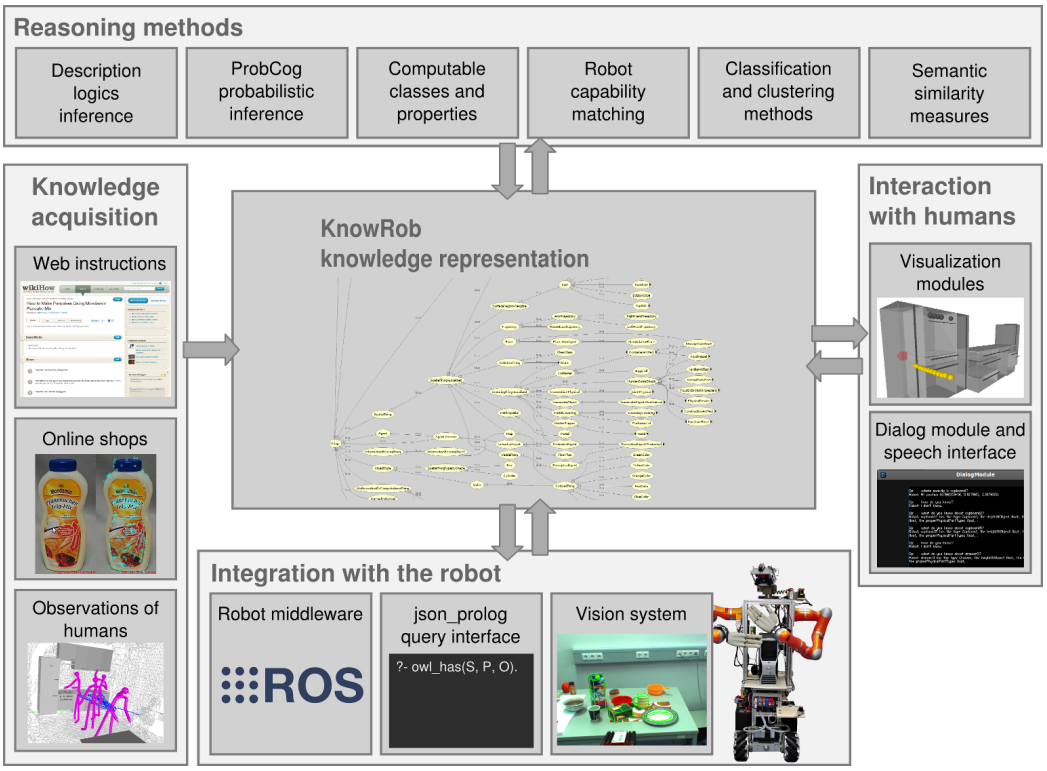


Figure 3.5. KnowRob Architecture. Taken from [2].

An overview of KnowRob's architecture is shown in Figure 3.5. This shows the key KnowRob interfaces and illustrates how modular the system is. Integration with the robot includes interfacing to the perception and executive systems, as well as the ability to call ROS services and topics. The KnowRob system itself is actually distributed as a ROS stack, meaning it can easily be used in existing ROS-based architectures, including the one developed at Heriot-Watt's Ocean Systems Laboratory. The interfaces to knowledge acquisition systems and human interaction systems are most useful for household robots while the visualization module enables humans to visualise the knowledge a robot holds about its environment given appropriate CAD models.

The reasoning interfaces at the top of Figure 3.5 are important and include a module for computing spatial and temporal relations between objects, a module for matching the robot's knowledge of its own capabilities to actions needed to perform a task, and a module interfacing to classification systems. Finally the ProbCog module provides probabilistic inference capabilities and its built on BLNs (see Section 2.3.2).

3.4 Summary and Discussion

This chapter was concerned with presenting the field of knowledge engineering with ontologies. In doing so, we presented various languages for authoring ontologies and outlined our strong preference for the OWL family and in particular for OWL DL. Being a language that is part of the DL family of languages, OWL DL offers logic-based knowledge representation and reasoning. This is a very desirable feature for the development of applications like ours which is concerned with persistently autonomous underwater operations. In such a complex field, where entities interact with each other, it is of utmost importance that their relations are captured clearly and correctly so that correct conclusions are drawn. In this, logic is a very useful tool, especially when there is a balance between the expressive power of logic and desirable features such as soundness, completeness and decidability upon applying reasoning. OWL DL offers such a balance. Further, we have presented and analysed the components (building blocks) of OWL DL ontologies in order to provide better understanding of how knowledge is engineered using OWL DL ontologies.

By presenting an overview of how ontologies are used in the robotics field we have provided a solid intuition on the benefits of an ontological approach in structuring knowledge. Among the ontology-based systems one that stands out is KnowRob. As was mentioned in this chapter, even though KnowRob was developed with household robots in mind it is generic enough to be extended in any robotic context. This flexibility and extensibility of KnowRob led us in its adoption for our work in building a framework for persistently autonomous underwater operations. Another reason is that autonomous operations are in most cases concerned with dynamic environments for which there is a constant flow of information which will make the KB grow and change at runtime. This makes the usage of traditional DL reasoners impractical since they require that all knowledge be present in the KB before the inference process commences and relevant queries can be answered. In the event that something changes the whole KB needs to be reclassified form scratch [2]. For small KBs this reclassification is insignificant in terms of computational cost. However this is not the case for large, constantly increasing KBs where the computational cost of reclassification is very high [2]. With KnowRob, observations can be continuously fed into the inference process as they are produced without having to reclassify the KB.

Chapter 4

Artificial Intelligence Planning

The purpose of this chapter is to present an overview of the field of Artificial Intelligence (AI) planning as it is an important aspect of autonomous underwater operations in general and by extension of operations concerning maritime defence (see Sections 1.1, 1.1.1 for a reminder of the problem statement as well as Section 1.2 for a reminder of the objectives of the thesis). Regarding this chapter, in Section 4.1 we present an overview of AI planning while in Section 4.2 we present classical planning. Moreover, in Section 4.3 we present temporal planning while Section 4.4 is concerned with planning under uncertainty. The Planning Domain Definition Language (PDDL) is the center of focus of Section 4.5. In Section 4.6 we briefly present a temporal planner called OPTIC. Finally we present research outputs with respect to planning for autonomy in Section 4.7.

4.1 Overview

Artificial Intelligence (AI) planning, also know as automated planning or simply planning, is a field of AI that is concerned with the formulation and organization of actions in a way that when executed by some intelligent agent, system, vehicle etc. will enable it to achieve some goals. In other words, planning is the process of solving a planning problem by formulating actions and organizing them in a way that when executed will enable an intelligent agent achieve some goal(s) as dictated by the planning problem at hand [134], [3]. Figure 4.1 illustrates a conceptual model for planning. Before going into detail about Figure 4.1, let us first define Σ which represents a state-transition system [3]. A state transition system is a 4-tuple $\Sigma = (S, A, E, \gamma)$ where:

- $S = \{s_1, s_2, ..., s_n\}$ is a finite or recursively enumerable set of states.
- $A = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ is a finite or recursively enumerable set of actions.
- $E = \{e_1, e_2, ..., e_n\}$ is a finite or recursively enumerable set of events. Events are contingent transitions. That is, they are governed by the internal dynamics of the system instead of being controlled by a plan executor.
- $\gamma : S \times A \times E \rightarrow 2^S$ is a state-transition function.

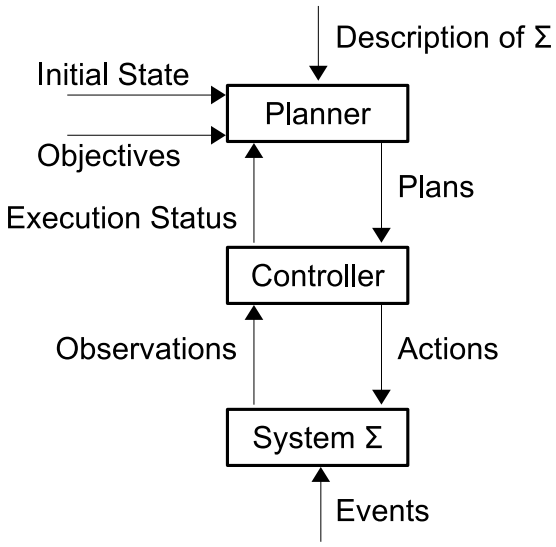


Figure 4.1. A conceptual model for planning [3].

Now, judging from figure 4.1 planning can be described in terms of three main interacting components. First, a planner, which, given a description of Σ, an initial state and some objective[1], generates a plan for the controller in order to achieve that objective. Second, a controller, outputs an action α according to some plan. That is, of course, given as an input the state s of the system. At this point it worth noting that in many cases the information that the controller has about the state of the system in incomplete. That is, the controller has partial knowledge. Partial knowledge can be modelled as an observation function $\eta : S \rightarrow O$ where O is a discrete set of observations: $O = \{o_1, o_2, ...o_n\}$. As such, the controller's input is the observation $o = \eta(s)$ that corresponds to the current state [3]. Finally, the third component is a state-transition system Σ that evolves according to its state-transition function γ as well as the events and actions that it receives. One last observation to make by looking at Figure 4.1 is that the execution status is fed into the planner by the controller. This allows interleaving planning and executing as well as the utilization of replanning and plan revising mechanisms in response to discrepancies (differences) between Σ and the real world. A more simplistic alternative would be to omit the execution status arrow from the controller to the planner in Figure 4.1. That would imply that the controller is robust enough to deal with discrepancies on its own which, for the vast majority of cases, is hardly the case.

With respect to planning in general, there are eight restrictive assumptions that can be made. The existence or absence of combinations of those assumptions defines the type of planning that is going to take place [3] (see Sections 4.2, 4.3, 4.4). The following list presents those assumptions:

- Assumption 0 (A0): Finite Σ. That is, Σ comprises a finite set of states.
- Assumption 1 (A1): Fully observable Σ. That is, we have full knowledge about the state of Σ.

[1] An objective can be a goal state or a set of goal states. In a more general manner, the objective is to satisfy some condition over the sequence of states followed by the system [3].

- Assumption 2 (A2): Deterministic Σ. More intuitively, given that an action is applicable to some state, the application of that action in that state will bring the system deterministically to a single other state. The mathematical expression for this is $|\gamma(s,u)| \leq 1$ where γ denotes a state-transition function, s is a state and u is is an action or event. Note that we also include events because this is also applicable to events.

- Assumption 3 (A3): Static Σ. That is, there are no events ($E = \emptyset$). This effectively means that unless some controller applies an action, Σ remains in the same state.

- Assumption 4 (A4): Restricted goals. This means that the planner only handles goals that are part of an explicit goal state (s_g) or set of goal states (S_g).

- Assumption 5 (A5): Sequential plans. This means that a (solution) plan is a finite sequence of actions that are linearly ordered.

- Assumption 6 (A6): Implicit time. The notion of implicit time dictates that actions and events have no duration. As a consequence, the application of an action or the existence of some event incurs an immediate state transition.

- Assumption 7 (A7): Offline planning. That is, while planning, a planner is not concerned with changes that may occur in Σ. As a consequence, it plans only for the given initial and goal state(s).

A state-transition system that embodies all the aforementioned assumptions is called a restricted state-transition system.

Except for different types of planning based on the aforementioned assumptions, planning can be also categorized based on the problem at hand [3]. For instance, path and motion planning is concerned with the formulation of plans that synthesize a path from some starting point in space to some goal point in space and a control trajectory along that path. Perception planning on the other hand, is concerned with formulating plans that involve sensing actions that are necessary for collecting data. Moreover, manipulation planning is concerned with formulating plans that will enable an acting agent manipulate objects in its environment. In fact any problem can be thought of as a planning problem because any problem would require a series of actions to be solved. As such, we could have transportation planning referring to planning for running transportation means (buses, airplanes, etc.). Consequently, the categorization of planning based on the problem at hand is rather wide and in many cases different terms can be used for planning for the same type of problem. In the case of manipulating objects for example where we referred to that form of planning as manipulation planning, we could just as easily use the term task planning. That is, manipulation could be thought of as a task that some agent, human or artificial, could undertake.

Having presented a fraction of the diverse domain of planning problems we would like to make another categorization based on whether the process of planning in general is de-

pendant on the domain of application or not. In this case we can have domain-independent planning and domain-specific planning [3]. In the case of domain-specific planning any commonalities even between diverse domains are mostly not taken into consideration. As such there can be an unnecessary fragmentation of planning approaches. On the other hand, in domain-independent planning such commonalities are exploited in a manner that justifies the usage of generic planners. That of course does not mean that domain-specific planning should not be employed. According to Ghallab et. al. [3] it would be as if we argued in favour of replacing specialized computation techniques in a computer, something that would be highly inefficient.

4.2 Classical Planning

Classical planning refers in general to planning under the restricted state-transition system. In this case, the state-transition system is no longer a 4-tuple as we described in section 4.1 but a triple denoted as $\Sigma = (S, A, \gamma)$. Note that events are missing because of the assumption of a static Σ (see section 4.1). Given the aforementioned restricted Σ, a planning problem, denoted as P, is also a triple: $P = (\Sigma, s_0, g)$ where s_0 is an initial state and g a goal state or a set of goal states. In this case, finding a solution to P is finding an action sequence $(\alpha_1, \alpha_2, ..., \alpha_k)$ that corresponds to a state transition sequence $(s_0, s_1, ..., s_k)$ such that $s_1 = \gamma(s_0, \alpha_1), ..., s_k = \gamma(s_{k-1}, \alpha_k)$, and s_k is a goal state [3]. As was mentioned in the beginning of this chapter, it is outside the scope of this chapter and indeed of this thesis to give a full account of AI planning. In the same manner, it is outside the scope of this section to give a full account of classical planning. As such, we will present three main approaches to represent planning domains and problems in the classical planning field (see Sections 4.2.1, 4.2.2, 4.2.3). The interested reader is also referred to detailed textbooks about planning written by Ghallab et. al. [3], [134]. In addition, we will, describe planning algorithms for classical planning. Again, the reader is referred to the planning textbooks cited in this chapter for a full account of AI planning.

4.2.1 Set-Theoretic Representation

Under the set-theoretic representation, world states are propositions. Each action represents an expression identifying which propositions belong to a state for the action to be applicable and which (propositions) need to be added or removed so that a new state of the world can be made. Let us now present some formal definitions [3].

Definition 4.1 *Let $L = \{p_1, p_2, ..., p_n\}$ be a finite set of proposition symbols. A planning domain on L under the set-theoretic representation is a restricted state-transition system $\Sigma = (S, A, \gamma)$ such that:*

- *$S \subseteq 2^L$ which means that each state s is a subset of L. If $p \in s$, then p holds in the state represented by s. If not, i.e. $p \notin s$, then p does not hold i the state represented*

by s.

- *Each action $\alpha \in A$ is a triple of subsets of L, which are denoted as $\alpha = (precond(\alpha), effects^{-}(\alpha), effects^{+}(\alpha))$. $precond(\alpha)$ is called the precondition of α. That is, the condition(s) that must hold for α to be applicable. In addition, $effects^{-}(\alpha)$, $effects^{+}(\alpha)$ are sets of effects of α. That is, the effects (postconditions) of applying α, negative and positive respectively. Negative and positive effects must be disjoint, i.e. $effects^{-}(\alpha) \cap effects^{+}(\alpha) = \emptyset$. For action α to be applicable in a state s, we need to have $precond(\alpha) \subseteq s$.*

- *S has the property that if $s \in S$, then for every action α that is applicable to s, the set $(s - effects^{-}(a)) \cup effects^{+}(\alpha) \in S$. More intuitively, whenever an action is applicable to a state, it generates a new state.*

- *The state-transition function is $\gamma(s, \alpha) = (s - effects^{-}(a)) \cup effects^{+}(\alpha)$ if $\alpha \in A$ is applicable to $s \in S$, and $\gamma(s, \alpha)$ is undefined otherwise.*

Definition 4.2 *A planning problem under the set-theoretic representation is a triple $\mathcal{P} = (\Sigma, s_0, g)$ where:*

- *s_0, that is, the initial state, is a member of S.*

- *$g \subseteq L$ is a set of propositions known as goal propositions that provide the requirements that a state must satisfy for it to be a goal state. The set of goal states, denoted as S_g, is $S_g = \{s \in S | g \subseteq s\}$.*

Definition 4.3 *A plan is any sequence of actions $\pi = \langle \alpha_1, \alpha_2, ...\alpha_k \rangle$, where $k \geq 0$. the length of π is $|\pi| = k$. That is, it equals the number of actions.*

Definition 4.4 *Let $\mathcal{P} = (\Sigma, s_0, g)$ be a planning problem. A plan π is a solution for $\mathcal{P}$ if $g \subseteq \gamma(s_0, \pi)$. A solution π is redundant if there is a subsequence of π that is also a solution for $\mathcal{P}$. In addition, a solution plan π is minimal if no other solution plan for $\mathcal{P}$ comprises fewer actions compared to π.*

With the above, we have now provided all the necessary definitions for planning domains (Definition 4.1), planning problems (Definition 4.2), plans (Definition 4.3) and solution plans (Definition 4.4) under the set-theoretic representation.

4.2.2 Classical Representation

Under the classical representation, instead of utilising propositions, we utilize first-order literals as well as logical connectives. For a reminder about propositions and first-order constructs the reader is advised to have a look at Sections 2.1.1 and 2.1.2 respectively. Like in the case of the the set-theoretic representation in Section 4.2.1 we will be providing definitions for planning domains, planning problems, plans, and solution plans under the

classical representation setting. Before doing so let us first provide some definitions for operators and actions. The requirement to do so stems from the fact that actions in the classical representation are represented by (planning) operators.

Definition 4.5 *A planning operator is a triple* $o = (name(o), precond(o), effects(o)$ *where:*

- $name(o)$ *is the name of the operator and is an expression of the form* $n(x_1, x_2, ..., x_k)$ *with n being a symbol known as an operator symbol,* $x_1, x_2, ..., x_k$ *are all of the variable symbols that appear in o, and n is unique, (that is, no two operators in* $\mathcal{L}$ *have the same operator symbol).*
- $precond(o)$ *and* $effects(o)$ *are generalizations of the preconditions and effects of the preconditions and effects of a set-theoretic action: instead of being sets of propositions, they are sets of literals, (that is, atoms and negations of atoms).*

In the above definition (Definition 4.5), $\mathcal{L}$ is a first-order language that comprises a finite number of predicate symbols as well as a finite number of constant symbols. As a reminder have a look at Section 2.1.2 for first-order constructs. Furthermore, $\mathcal{L}$ has no function symbols and as such every term of $\mathcal{L}$ is either a constant or a variable while a state is a set of ground atoms of $\mathcal{L}$ [3].

Definition 4.6 *For any set of literals L,* L^+ *is the set of all atoms in L, and* L^- *is the set of all atoms whose negations are in L. If o is an operator or operator instance, then* $precond^+(o)$ *and* $precond^-(o)$ *are o's positive and negative preconditions respectively, while* $effects^+(o)$ *and* $effects^-(o)$ *are o's positive and negative effects respectively.*

Definition 4.7 *An action is any ground instance of a planning operator. If* α *is an action and s is a state such that* $precond^+(\alpha) \subseteq s$ *and* $precond^-(\alpha) \cap s = \emptyset$*, then* α *is applicable to s and the result of applying* α *to s is the state:* $\gamma(s, \alpha) = (s - effects^-(\alpha)) \cup effects^+(\alpha)$.

In case there is a confusion about operators, actions, preconditions and effects let us provide an exemplar operator and its instantiation in the following snippet.

Snippet 1 An operator and its instantiation (action). Taken from [3].

An operator:

```
move(r,l,m)
    precond: adjacent(l,m), at(r,l), ¬occupied(m)
    effects: at(r,m), occupied(m), ¬occupied(l), ¬at(r,l)
```

An instantiated operator (action):

```
move(robot1,loc1,loc2)
    precond: adjacent(loc1,loc2), at(robot1,loc1), ¬occupied(loc2)
    effects: at(robot1,loc2), occupied(loc2), ¬occupied(loc1), ¬at(robot1,loc1)
```

Snippet 1 illustrates a move operator and a move action. Inputs r represents a robot while l, m locations. What the operator tells us is that in order for a robot r to move from

some location l to some location m, l, m must be adjacent, robot r must be at location l while location m must be not occupied. Notice here that we have two positive and one negative precondition. The effects of some robot r moving from some location l to some location m would be that some robot r is now at some location m, some location m is now occupied, some location l is no longer occupied and of course that some robot r is no longer at some location l. Notice that the set of effects comprise two positive and two negative ones. Now, with respect to the corresponding action, it is nothing more than an instantiation of the operator. That is, variable r representing a robot and variables l, m representing locations have been instantiated with a specific robot and locations. That is, $robot1$ and $loc1, loc2$ respectively.

Let us now provide the definitions for planning domains, problems, plans and solutions under the classical representation [3].

Definition 4.8 *Let $\mathcal{L}$ be a first-order language that has finite number of predicate and constant symbols. A classical planning domain in $\mathcal{L}$ is a restricted state-transition system $\Sigma = (S, A, \gamma)$ such that:*

- $S \subseteq 2^{\{all\ ground\ atoms\ of\ \mathcal{L}\}}$
- $A = \{$*all ground instances of operators in O*$\}$*, where O is a set of operators as defined earlier.*
- $\gamma(s, \alpha) = (s - effects^{-}(\alpha)) \cup effects^{+}(\alpha)$ *if* $\alpha \in A$ *is applicable to* $s \in S$*, and* $\gamma(s, \alpha)$ *is otherwise undefined.*
- *S is closed under* γ*, that is, if* $s \in S$*, then for every action* α *that is applicable to s,* $\gamma(s, \alpha) \in S$*.*

Definition 4.9 *A classical planning problem is a triple $\mathcal{P} = (\Sigma, s_0, g)$, where:*

- s_0 *is the initial state and can be any state in set S;*
- *g is the goal state and can be any set of ground literals*
- $S_g = \{s \in S \mid s\ satisfies\ g\}$

The statement of a planning problem $\mathcal{P} = (\Sigma, s_0, g)$ is $P = (O, s_0, g)$. The statement of a planning problem represents a syntactic specification used in order for $\mathcal{P}$ to be "understandable" by a computer program, that is, for a computer program to to be able to process it [3]. Both the definitions of a plan π and the outcome of applying π to a state are the same as the ones described in Section 4.2.1. As such, the interested reader is referred to the aforementioned section. Let us just present the definition for the solution (plan).

Definition 4.10 *A plan π is a solution for $\mathcal{P}$ if:*

- $\gamma(s_0, \pi)$ *satisfies g, where g is the goal and* s_0 *the initial state.*

Again, redundancy is defined in the same way as in the case of the set-theoretic representation (see Section 4.2.1).

4.2.3 State-Variable Representation

Under the state-variable representation each state is no longer a proposition or first-order literal but a tuple of values of n state variables $(x_1, x_2, ..., x_n)$. As far as actions are concerned, they are represented by partial functions that map tuples of the form that we just presented into other tuples of values of the n state variables. According to Ghallab et. al. [3], this representation approach is "particularly useful for domains in which states are sets of attributes that range over finite domains and whose values change over time". In order to define operators and actions we first need to define the necessary concepts related to their definitions. Let us start with constant symbols. Constant symbols, also known as object symbols, represent the objects in a domain, very much like constants in first-order logic. In the example that we provided in the previous section (Section 4.2.2) about operators and actions, $robot1$, $loc1$, $loc2$ are constants. Moving on, object variable symbols are typed variables that range over classes of constants. Again, from the example given in Section 4.2.2, r is an object variable symbol for which $r \in robots$. State variable symbols are functions from the set of states and none or more sets of constants into a set of constants. Relation symbols on the other hand represent rigid relations on the constants. A rigid relation is any relation that is state-invariant. That is, it does not change irrespective of what is the state of the world or the state of an acting agent in it. To better understand rigid relations let us use the operator example that we used in Section 4.2.2 (see Snippet 1). Recall that, one of the preconditions of operator $move(r,l,m)$ was $adjacent(l,m)$. This represents a rigid relation because it does not change over time or over states. In our case, if two locations are adjacent they will always be adjacent. If they are not adjacent, they will never be adjacent. Anything else would be wrong[2]. The opposite of a rigid relation is a flexible relation which is also known as fluent. The explanation of flexible relations is not needed since one can understand that the opposite of state-invariance is state-variance. Let us now provide definitions for operators and actions [3].

Definition 4.11 *A planning operator is a triple* $o = (name(o),\ precond(o),\ effects(o))$ *where:*

- $name(o)$ *is a syntactic expression of the form* $n(u_1, u_2, ..., u_k)$*, where n is a symbol called an operator symbol,* $u_1, u_2, ..., u_k$ *are all of the object variable symbols that appear in o, n is unique. That is, two or more operators are not allowed to have the same operator symbol.*
- $precond(o)$ *is a set of expressions on state variables and relations.*
- $effects(o)$ *is a set of value assignments to state variables and have the form of* $x(t_1, t_2, ..., t_k) \leftarrow t_{k+1}$*, where each* t_i *is a term in the appropriate range.*

Definition 4.12 *An action* α *is a ground operator o that meets the rigid relations in* $precond(o)$*, meaning that every object variable in o is substituted by a constant of the corresponding*

[2]Unless we are talking about a domain in which we are able to bend or stretch space.

class such that such constants meet the rigid relations in precond(o). An action α is applicable in a state s when the values of the state variables in s meet the conditions in precond(α). As such, updating the values of the state variables according to the assignments in $effects(a)$ will generate a new state $\gamma(s,\alpha)$.

Let us now provide the definitions for planning domains and problems under the state-variable representation [3].

Definition 4.13 *Let $\mathcal{L}$ be a planning language in the state-variable representation defined by X, R, where X is the set of all ground state variables of $\mathcal{L}$ and R a finite set of rigid relations. A planing domain in $\mathcal{L}$ is a restricted state-transition system $\Sigma = (S,A,\gamma)$ such that:*

- *$S \subseteq \prod_{x\in X} D_x$, where D_x is the range of the ground state variable x; a state s is denoted as $s = \{(x = c) \mid x \in X\}$, where $c \in D_x$.*
- *$A = \{$all ground instances of operators in O that meet the relations in $R\}$, where O is a set of operators; an actions α is applicable to a state s if and only if every expression $(x = c)$ in precond(α) is also in s; R is the set of rigid relations.*
- *$\gamma(s,\alpha) = \{(x = c) \mid x \in X\}$, where c is specified by an assignment $x \leftarrow c$ in $effects(\alpha)$ if there is such an assignment, otherwise $(x = c) \in s$.*
- *S is closed under γ, that is, if $s \in S$, then for every action α that is applicable to s, $\gamma(s,\alpha) \in S$.*

Definition 4.14 *A planning problem is a triple $\mathcal{P} = (\Sigma, s_0, g)$, where s_0 is an initial state in S and the goal g is a set of expressions on the state variables in X.*

A statement of a planning problem $\mathcal{P}$ is a 4-tuple $P = (O,R,s_0,g)$, where O is the set of operators, R is the set of rigid relations, s_0 is the initial state, and g the goal. The definitions of plans and solutions in this representation are the same as in Sections 4.2.1, 4.2.2 (see Definitions 4.3, 4.4).

4.2.4 Algorithmic Families

Regarding planning algorithms in the classical domain they can be mainly categorized into two families: state space algorithms and plan space algorithms. Let us first start with state space algorithms.

State space algorithms are search algorithms traversing graphs and use a subset of the state space as a search space. Each node in the graph represents a state of the world while edges (arcs) represent state transitions or actions. In this case a plan is path in the search space with the two ends being the initial and the goal state respectively. A famous state space algorithm is the forward search algorithm [3]. The algorithm takes as input a set of

operators, an initial state and a goal state (O, s_0, g). If the planning problem $\mathcal{P}$ is solvable, i.e. there is an action sequence that makes the transition from the initial to the goal state feasible then the algorithm will return this action sequence (plan). If not, it will return failure. Another state space algorithm is the backward search algorithm which, as its name suggests, works backwards. That is, starting from the goal state and applying inverses of the operators so that subgoals can be generated. The algorithm terminates if a the set of subgoals is satisfied by the initial state and returns the sequence of actions or returns failure otherwise [3]. The returned solution of the backward search algorithm, if any, needs to be reversed to provide the solution to the planning problem. Note that both forward and backward search algorithms are sound and complete. Yet another planning algorithm that lies within the state space planning algorithmic family is STRIPS [3]. The STRIPS algorithm is in its essence similar to a backward search algorithm since it works from the goal to reach the initial state but as opposed to the backward search algorithm it is not complete. This incompleteness of the STRIPS algorithm is due to the state branching factor reduction that it implements which reduces the size of the search space considerably but at the cost of rendering the algorithm incomplete. We will not go in to detail about how STRIPS works but the interested reader is referred to two very detailed textbooks by Ghallab et. al. [3], [134].

With respect to plan space search algorithms, the search space is not one comprising of states like in the case of state space algorithms but of partially specified plans [3]. As such, plan space algorithms search graphs whose nodes are partially specified plans while their edges (arcs) are plan refinement operations and not state transitions or actions. The goal of such refinement operations is to incrementally, complete a partial plan further. That is, fulfilling open[3] goals or eradicating potential inconsistencies. A fundamental concept of plan space planning algorithms is the least commitment principle. This principle dictates that refinement operations are to avoid adding constraints to the partial plan that are not strictly required for addressing the refinement purpose. Except for the search space being different in plan space planning when compared to state space planning, another difference lies in the definition of solution plans. That is, instead of a sequence of actions, two things are considered: action choices and action ordering. Let us provide some definitions in order to clarify things before we talk about the algorithms themselves [3].

Definition 4.15 *A partial plan is a tuple $\pi = (A, \prec, B, L)$ where:*

- *$A = \{\alpha_1, \alpha_2, ..., \alpha_3\}$ is a set of partially instantiated planning operators.*
- *$\prec$ is a set of ordering constraints on A which have the form of $(\alpha_i \prec \alpha_j)$.*
- *B is a set of binding constraints on the variables of actions in A that have the form of $x = y$, $x \neq y$ or $x \in D_x$ with D_x being a subset of the domain of x.*

[3]Not yet fulfilled

- *L is a set of causal links of the form* $\langle \alpha_i \xrightarrow{p} \alpha_j \rangle$*, such that* a_i *and* a_j *are actions in A, the constraint* $(\alpha_i \prec \alpha_j)$ *is in* $\prec$*, proposition p is an effect of* α_i *and a precondition of* α_j *while the binding constraints for variables of* α_i, α_j *appearing in p are in B.*

Definition 4.16 *A partial plan* $\pi = (A, \prec, B, L)$ *is a solution plan for problem* $P = (\Sigma, s_0, g)$ *if:*

- *its ordering constraints* $\prec$ *and binding constraints B are consistent and*
- *every sequence of totally ordered and totally instantiated actions of A satisfying* $\prec$ *and B is a sequence that defines a path in the state-transition system* Σ *from the initial state* s_0 *corresponding to effects of action* a_0 *to a state containing all goal propositions in g.*

Definition 4.17 *An action* α_k *in a plan* π *is a threat on a causal link of the form* $\langle \alpha_i \xrightarrow{p} \alpha_j \rangle$ *if and only if:*

- α_k *has an effect* $\neg q$ *that is possibly inconsistent with p, that is, q and p are unifiable;*
- *the ordering constraints* $(\alpha_i \prec \alpha_k)$ *and* $(\alpha_k \prec \alpha_j)$ *are consistent with* $\prec$*;*
- *the binding constraints for the unification of q and p are consistent with B.*

Definition 4.18 *A flaw in a plan* $\pi = (A, \prec, B, L)$ *is either:*

- *a subgoal, that is, a precondition of an action in A without causal links or*
- *a threat, that is, an action that may interfere with a causal link.*

Now, regarding the search algorithms themselves in plan space planning, let us briefly describe the rationale behind the PSP (Plan Space Planning) algorithm, a generic algorithm that can be ”instantiated” into several variants. PSP is initially provided with a partial plan that contains two dummy actions *init* and *goal* in order to encode the initial state and the goal condition into the search problem. In addition, the initial partial plan π contains one ordering constraint of the form *init* $\prec$ *goal* and no variable bindings or causal links. The algorithm proceeds with identifying the flaws of π and if π contains none then it returns π as the solution plan. If the plan is flawed then the algorithm selects a flaw and it attempts to find all possible ways in which it can be resolved. Each such way is called a resolver. If there is no resolver then it returns failure, i.e. the planning problem cannot be solved. If there are resolvers it chooses one of them non-deterministically and refines the plan with that resolver. The new plan (after applying the resolver) is utilized in order to make a recursive call to the algorithm. PSP is sound and complete. A famous variant of the PSP is the POP (Partially Ordered Planning) algorithm. The main difference to PSP is that it employs a different approach when resolving flaws. Recall from Definition 4.18 that a flaw is either a subgoal or a threat. While PSP does not distinguish between the two when choosing a flaw non-deterministically at each recursion, POP first refines with regards to a subgoal and then proceeds with solving all threats due to the resolver of that subgoal. This is done for each recursion. POP is also sound and complete.

4.3 Temporal Planning

Recall from section 4.1 that Assumption 6 refers to implicit time, i.e. actions and events have no duration. As such, the application of an action or the existence of some event incurs an immediate state transition. The main difference between temporal and classical planning is that in temporal planning we make no such assumption. That is, actions and effects have duration. In this section we will approach temporal planning from the perspective of utilizing temporal databases. Other approaches such as utilizing chronicles [3] are outside the scope of this section. Moreover, we will be using concepts from point and interval algebras, abbreviated as PA and IA respectively, in order to provide definitions for temporal planning related concepts [3]. Both PA and IA are symbolic calculi that allow us to relate a set of (time) instants and (time) intervals respectively with qualitative and quantitative relations in time forming in this way qualitative and quantitative temporal constraints [3]. Qualitative (temporal) constraints embody the need to express time qualitatively, i.e. to deal with synchronization of actions and events. For instance, to plan for a robot to pick up an object from some location when the object is yet not there. As such, the object needs to be there at a time instant before or at the same time that the robot will pick it up without specifying when the time instants will be. On the other hand, quantitative (temporal) constraints embody the need to express time quantitatively as a resource, i.e. to model durations of actions with regards to possible deadlines that may exist of some cost functions that need to be optimized. For instance, to plan for a robot to pick up an object from some location in exactly three minutes. However, in this section temporal planning will be presented from the scope of qualitative temporal constraints. Consequently, when using the term temporal constraints we will be referring to qualitative temporal constraints unless explicitly stated otherwise. For temporal planning with quantitative temporal constraints the interested reader is referred to Ghallab et. al. [3].

4.3.1 Planning with Temporal Databases

Let us present some concepts related to the representation scheme (temporal databases) that are necessary in defining temporal databases, operators, actions, domains, problems and plans. The concepts are the following: constant symbols, variable symbols, relation symbols, constraints as well as temporally qualifying expressions. Constant symbols are no different from constant symbols as they were presented in classical planning (see Section 4.2). Variable symbols on the other hand, are either object variables as they where presented in Section 4.2 or temporal variables which range over the set of real numbers $\mathcal{R}$. Relation symbols can either represent rigid or flexible relations. That is, relations that do not change and change over time respectively. In section 4.3 we introduced the notion of temporal constraints. However, constraints can be of two types: temporal and binding.

Temporal constraints involve temporal variables and are used to create restrictions between them either in the form of restrictions between (time) instants or (time) intervals.

Regarding temporal constraints and time instants, they are formed based on the following set of primitive relation symbols $Pr_{inst} = \{<, =, >\}$ denoting the notions of *before*, *equal to* (or *at the same time*) and *after* respectively. Now, the set of all temporal constraints on time instants is $R_{inst} = 2^{Pr_{inst}} = \{\emptyset, \{<\}, \{=\}, \{>\}, \{<,=\}, \{<,>\}, \{>,=\}, Pr_{inst}\}$. For instance, if we have two instants t_1, t_2 and wanted to to express that t_1 comes before t_2 we would use $t_1 < t_2$. Alternatively, if we wanted to express that the instants are not equal we would use $[t_1\{<,>\}t_2]$ ($[t_1 \neq t_2]$ for better readability). The set of temporal constraints on instants is really self explanatory except for the case of the constraints $\emptyset, Pr_{inst}$ which are special constraints. Constraint $\emptyset$ is the empty constraint denoting a constraint that cannot be satisfied while Pr_{inst} is the universal constraint denoting a constraint that can be satisfied by every tuple of values of the temporal variables it involves. Each element of R_{inst} is itself a set of primitive relation symbols. With respect to temporal constraints on (time) intervals things are a bit more complicated. Instead of three primitive relation symbols we have the following thirteen primitive relation symbols: $Pr_{inter} = \{b, b', m, m', o, o', s, s', d, d', f, f', e\}$, which stand for *before*, *after*, *meet*, *is met by*, *overlap*, *is overlapped by*, *start*, *is started by*, *during*, *includes*, *finish*, *is finished by*, *equal* respectively. An observant reader may have noticed that, in pairs[4], the primitive relation symbols can form symmetrical constraints between intervals. Let i, j be intervals. As such, $i\ b'\ j$ when $j\ b\ i$. That is, interval i is after interval j when interval j is before interval i and so on. Figure 4.2 illustrates seven of the primitive relations between intervals. Note that the length of the intervals in Figure 4.2 is

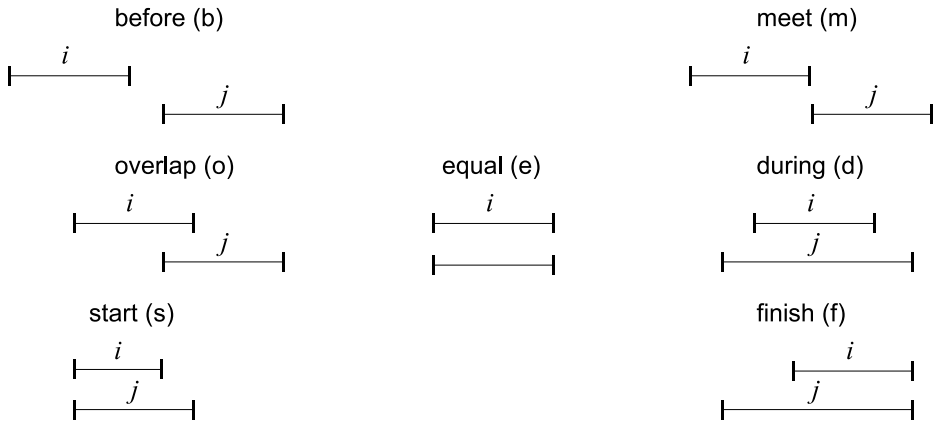


Figure 4.2. Primitive temporal relations between intervals (i, j). The symmetrical relations are omitted but are very straightforward to form.

variable. We have obviously done so in order to demonstrate the various primitive relations without having to resort the usage of multiple intervals. Again, an observant reader may have noticed that e (equal) does not have a symmetrical relation. The fact is that it does but it is identical. As such it is not necessary to include it the the set of primitive relations. Now the set of temporal constraints on intervals is $R_{inter} = 2^{Pr_{inter}} = \{\emptyset, \{b\}, \{m\}, \{o\}, ..., \{b,m\}, \{b,o\}, ..., \{b,m,o\}, \{b,m,s\}, ..., Pr_{inter}\}$. That is 8192 constraints. Each element of

[4]Pairs here refer to pairs of the form (b, b'), (m, m') and so on. They do not refer to random pairs such as (b, o') or (o, s).

the R_{inst} set is itself a set of primitive relation symbols and is interpreted as the disjunction of those primitives [3]. For instance, let i, j be intervals. A constraint of the form $[i\ \{b,s\}\ j]$ is interpreted as $[(i\ b\ j) \vee (i\ s\ j]$.

Switching over to binding constraints, they are applied to object variables and are of the following form: $x = y$, $x \neq y$ and $x \in D$, with D being a set of constant symbols. Rigid relations and binding constraints represent time-invariant expressions while flexible relations and temporal constraints represent time-variant, also known as time-dependent, expressions. A term that is often used when referring to rigid relations and binding constraints is the term object constraints [3]. Let us now proceed with the definition of temporally qualified expressions, abbreviated as TQE, with are essential in defining temporal databases.

Definition 4.19 *A temporally qualified expression (TQE) is an expression of the form:* $p(\zeta_i, ..., \zeta_k)@[t_s, t_e)$*, where:*

- p *is a flexible relation*
- $\zeta_i, ..., \zeta_k$ *are constants or object variables*
- t_s, t_e *temporal variables such that* $t_s < t_e$
- $@[t_s, t_e)$ *represents the time interval* $[ts, t_e)$ *at which (@) the flexible relation holds.*

In fact, what a temporally qualified expressions "tells" us is that $\forall t$ such that $t_s \leq t < t_e$, the relation $p(\zeta_i, ..., \zeta_k)$ holds at time t [3]. Now let us provide the definition of a temporal database (Definition 4.20) [3].

Definition 4.20 *A temporal database denoted as* Φ *is a pair* $\Phi = (\mathcal{F}, \mathcal{C})$ *where:*

- $\mathcal{F}$ *is a finite set of TQEs*
- $\mathcal{C}$ *is a finite set of temporal and object constraints*
- $\mathcal{C}$ *is required to be consistent, that is, there exist values for variables that meet all the constraints.*

In short, as a construct, a temporal database holds assertions about how the world evolves over time. Now in order to be able to define temporal planning operators, actions, domains, problems, plans and solutions (see Section 4.3.1.1) we first need to provide definitions about: i) when a set of TQEs supports another TQE and when another set of TQEs (see Definition 4.21), ii) when a temporal database supports a set of TQEs and when another temporal database (see Definition 4.22), iii) when a temporal database entails another database (see Definition 4.23) [3].

Definition 4.21 *A set* $\mathcal{F}$ *of TQEs supports a TQE* $e = p(\zeta_i, ..., \zeta_k)@[t_1, t_2)$ *if and only if there is, in* $\mathcal{F}$*, a TQE* $p(\zeta'_i, ..., \zeta'_k)@[\tau_1, \tau_2)$ *and a substitution* σ *such that* $\sigma(p(\zeta_i, ...\zeta_k)) = \sigma(p(\zeta'_i, ...\zeta'_k))$*. An enabling condition for* e *in* $\mathcal{F}$ *is the conjunction of the two temporal*

constraints $\tau_1 \leq t_1$ and $t_2 \leq \tau_2$, together with the binding constraints of σ. $\mathcal{F}$ supports a set of TQEs $\mathcal{E}$ if and only if there is a substitution σ that unifies every element of $\mathcal{E}$ with an element of $\mathcal{F}$. An enabling condition for $\mathcal{E}$ in $\mathcal{F}$ is the conjunction of enabling conditions for the elements of $\mathcal{E}$. The set of all possible enabling conditions for $\mathcal{E}$ in $\mathcal{F}$ is denoted as $\theta(\mathcal{E}/\mathcal{F})$.

Definition 4.22 *A temporal database $\Phi = (\mathcal{F}, \mathcal{C})$ supports a set of TQEs $\mathcal{E}$ when $\mathcal{F}$ supports $\mathcal{E}$ and there is an enabling condition $c \in \theta(\mathcal{E}/\mathcal{F})$ that is consistent with $\mathcal{C}$. $\Phi = (\mathcal{F}, \mathcal{C})$ supports another temporal database $(\mathcal{F}', \mathcal{C}')$ when $\mathcal{F}$ supports $\mathcal{F}'$ and there is an enabling condition $c \in \theta(\mathcal{F}', \mathcal{F})$ such that $\mathcal{C}' \cup c$ is consistent with $\mathcal{C}$.*

Definition 4.23 *A temporal database $\Phi = (\mathcal{F}, \mathcal{C})$ entails another temporal database $(\mathcal{F}', \mathcal{C}')$ if and only if $\mathcal{F}$ supports $\mathcal{F}'$ and there is an enabling condition $c \in \theta(\mathcal{F}', \mathcal{F})$ such that $\mathcal{C} \models \mathcal{C}' \cup c$.*

The concept of entailment was introduced and analysed earlier in this thesis in section 2.1 when presenting logic-based knowledge representation approaches and is connected to inference. The concept of entailment was expanded to KBs which consisted of formulas. Unsurprisingly, entailment is applicable to temporal databases which instead of formulas consist of TQEs and constraints.

4.3.1.1 Temporal Operators, Actions, Axioms, Domains, Problems, Plans and Solutions

As in the case of classical planning where we presented different representation approaches and provided definitions for planning operators, actions, domains etc., in this section we are going to do the same but for the perspective of temporal planning with temporal databases. Let us first provide the definition for a temporal planning operator [3] (see Definition 4.24).

Definition 4.24 *A temporal planning operator is a tuple $o = (name(o), precond(o), effects(o), const(o))$, where:*

- *$name(o)$ is an expression of the form $o(x_1, x_2, ..., x_k, t_s, t_e)$ such that o is an operator symbol and $x_1, x_2, ..., x_k$ are all the object variables that appear in o together with the temporal variable appearing in $const(o)$. The other unconstrained temporal variables in o are free variables.*
- *$precond(o)$ and $effects(o)$ are TQEs.*
- *$const(o)$ is a conjunction of temporal constraints and object constraints with the latter being either rigid relations or binding constraints on object variables of the form $x = y$, $x \neq y$, or $x \in D$, with D being a set of constants.*

Continuing with definitions let us now present the definition for actions and the applicability of actions [3] (see Definition 4.25).

Definition 4.25 *An action α is a partially instantiated operator o such that $\alpha = \sigma(o)$ for some substitution σ. An action α is applicable to a temporal database $\Phi = (\mathcal{F}, \mathcal{C})$ if and only if $precond(\alpha)$ is supported by $\mathcal{F}$ and there is an enabling condition c in $\theta(\alpha/\mathcal{F})$ such that $\mathcal{C} \cup const(\alpha) \cup c$ is a consistent set of constraints.*

The result of applying an action to a database is a set of possible databases [3]. An observant reader may have noticed that negative effects are not made explicit in the definition of planning operators. Consequently, the effects of an action states only what holds as an effect of the action and not what does not hold. To achieve the latter we use domain axioms. The definition of a domain axiom is the following [3]:

Definition 4.26 *A domain axiom is a conditional expression of the form $p = cond(p) \rightarrow disj(p)$, where:*

- *$cond(p)$ is a set of TQEs*
- *$disj(p)$ is a disjunction of temporal and object constraints.*

It is not the scope of this chapter to be used as a substitute to textbooks on AI planning but rather to provide an intuition on various aspects of it. As such, the interested reader is referred to Ghallab et. al. [3] for a detailed account on domain axioms and their usage. Let us now provide the definition of a planning domain under the representation scheme of temporal databases (Definition 4.27) [3].

Definition 4.27 *A temporal planning domain is a triple $\mathcal{D} = (\Lambda_\Phi, \mathcal{O}, X)$, where:*

- *Λ_Φ is the set of all temporal databases that can be defined with the constraints as well as constant, variable and relation symbols within our representation.*
- *$\mathcal{O}$ is a set of temporal operators.*
- *X is a set of domain axioms*

Given the above definition we can define a temporal planning problem as follows:

Definition 4.28 *A temporal planning problem in planning domain $\mathcal{D}$ is a tuple of the form $\mathcal{P} = (\mathcal{D}, \Phi_0, \Phi_g)$, where:*

- *$\Phi_0 = (\mathcal{F}, \mathcal{C})$ in Λ_Φ that satisfies the axioms of X, Φ_0 represents an initial scenario that describes both the domain's initial state and the evolution predicted to take place independently of the actions to be planned.*
- *$\Phi_g = (\mathcal{G}, \mathcal{C}_g)$ is a database that represents the goals of the problem as a set $\mathcal{G}$ of TQEs together with a set $\mathcal{C}_g$ of objects and temporal constraints on variables of $\mathcal{G}$.*

The statement of a planning problem $\mathcal{P}$ is $P = (\mathcal{O}, X, \Phi_0, \Phi_g)$ while a plan, denoted as π is a set of actions $\pi = \{\alpha_1, \alpha_2, ..., \alpha_k\}$. A plan π is a solution if and only if there is a database that entails Φ_g after the application of π to Φ_0. Again, an observant reader may have noticed that the a plan is a set of actions and not a sequence. This normal since actions in temporal planning are situated in time. This enables us to have concurrent actions with interfering effects. In classical planning one cannot really have actions that have interfering effects to be executed concurrently. A workaround for this would be the definition of a third operator through which we could express both their joint preconditions and effects [3]. However this is not necessary in the context of temporal planning as one can reason on the joint applicability and effects of actions without the need to resort to additional operators [3].

4.4 Planning under Uncertainty

In this section we will briefly look into approaches for planning under uncertainty. Recall from section 4.1 that we presented eight assumptions that can be made with respect to planning. Moreover, when we presented classical planning in section 4.2 we stated that in classical planning we make all eight assumptions, one of which is assumption A2 referring to a deterministic state-transition system. When planning under uncertainty we no longer make such an assumption[5]. Consequently, each action can have multiple outcomes and the execution of an action and by extension of a plan can bring a system into a desired state but following potentially many different execution paths determined by the outcomes action execution. Another assumption that we made as part of classical planning was assumption A4 referring to restricted goals (see Sections 4.1, 4.2). However, in domains that exhibit non-determinism we make no such assumption[6]. The reason being that we will potentially need to specify goals of different strengths [3]. For instance, consider a scenario in which we have a robot and a series of bridges, say three, that the robot needs to cross in order to get to its final destination. The first two bridges are very easy to cross but the third is quite narrow. As such, we might require that the robot attempts to reach its final destination, i.e. crossing the three bridges but at the same time if it does not manage to do so, it guarantees that it will not fall over. Such goals are known as extended goals. Extended goals can be represented with utility functions modelling in this manner costs or rewards associated with action/plan execution. Let us now present one approach for planning under uncertainty that is based on Markov Decision Processes (see Section 4.4.1).

[5]That is not to say that we cannot have any actions which have deterministic effects when planning under uncertainty. We can have both deterministic and non-deterministic actions

[6]That is not to say that we cannot have restricted goals when planning under uncertainty. We can have both restricted and extended goals

4.4.1 Planning with Markov Decision Processes

At the very core of planning with Markov Decision Processes (MDPs) lies the concept of stochastic systems [3]. Recall from Section 4.2 that we introduced deterministic state-transition systems $\Sigma = (S, A, \gamma)$. Stochastic systems are non-deterministic state transition systems. Now, let us provide a formal definition.

Definition 4.29 *A stochastic system is a non-deterministic state transition system with a probability distribution over each state transition denoted as $\Sigma = (S, A, P)$, where:*

- *S is a finite set of states*
- *A is a finite set of actions*
- *$P_\alpha(s'|s)$, where $\alpha \in A$, $s, s' \in S$ and P is a probability distribution. That is, for each $s \in S$, if there exists $\alpha \in A$ and $s' \in S$ such that $P_\alpha(s'|s) \neq 0$, we have $\sum_{s' \in S} P(s, \alpha, s') = 1$.*

When planning using MDPs domains are expressed as stochastic systems. In the above definition (Definition 4.29) $P_\alpha(s'|s)$ is the probability that executing some action α in some state s will lead the system to transition to some state s'. Now, not all actions are executable in a state. The set of executable actions A in some state s is denoted as $A(s)$ and is the set of actions that have a non zero probability of transitioning the system to some state s'. Of course state s' is not necessarily the same for each action. For instance, consider the case of a robot being stationary at some position and two non-deterministic actions, move-forward and move-backward with move-forward related to a non zero probability of moving the robot forward and move-backward related to a non zero probability of moving the robot backward. In this case both actions are executable but the execution of each action will lead the system into different states that are not the same with each other.

Recall from section 4.4 that extended goals can be represented with utility functions. In the MDP setting, goals are represented just like this, i.e. by the means of utility functions. Utility functions are numerical functions that express preferences about which states should be traversed and/or actions to be performed by the means of rewards and costs [3]. Let us now provide the definition of a MDP.

Definition 4.30 *A Markov Decision Process (MDP) is a 5-tuple (S, A, P, U, γ), where:*

- *S is a finite set of states*
- *A is a finite set of actions*
- *$P_\alpha(s'|s)$, where $\alpha \in A$, $s, s' \in S$ and P is a probability distribution. That is, for each $s \in S$, if there exists $\alpha \in A$ and $s' \in S$ such that $P_\alpha(s'|s) \neq 0$, we have $\sum_{s' \in S} P(s, \alpha, s') = 1$.*

- *$U_\alpha(s) = R(s) - C_\alpha(s)$, where $\alpha \in A$, $s \in S$ and U is a utility function represented by the means of rewards and costs. As such, R is a reward function for which $R : S \to \mathcal{R}$ and C is a cost function for which $C : S \times A \to \mathcal{R}$. $\mathcal{R}$ is the set of real numbers.*

- *γ is a discount factor and $\gamma \in [0,1]$.*

A plan in the MDP setting is the specification of actions that some controller should execute given a state [3]. As such plans can be represented as policies, denoted as π. Policies are functions of the form $\pi : S \to A$ which means that a policy maps states into actions. Executing policies corresponds to infinite sequences of states. Such infinite sequences are called histories [3]. Histories are Markov chains. Markov chains are sequences of random events/values in which the probability of each event/value depends only on the previous state. In our case a history is a Markov chain that comprises sequences of states whose probabilities at a time interval depend only on the probabilities at the previous time interval. The utility function that we presented as part of Definition 4.30 can be generalized to policies and histories. For example, the utility of a policy in a state s is $U(s|\pi) = R(s) - C(s,(\pi(s))$ while the utility of a history is:

$$U(h|\pi) = \sum_{i \geq 0} \gamma^i (R(s_i) - C(s_i, \pi(s_i))) \tag{4.1}$$

where, h is a history, π is the policy that induced that history and s_i are states of history h, i.e. $h = \langle s_0, s_1, s_2, ... \rangle$ and γ is the discount factor. The usage of γ is necessary because without it the utility of h would never be able to converge to a finite value. The problem with this would be that we would not be able to use utilities for any meaningful calculation. Now it is time to put everything together and finalise our analysis of planning with MDPs.

Given the set of all possible histories of a stochastic system Σ denoted as H and π, a policy for Σ, the expected utility of π, denoted as $E(\pi)$, is:

$$E(\pi) = \sum_{h \in H} P(h|\pi) U(h|\pi) \tag{4.2}$$

where, $U(h|\pi)$ is the utility of a history h induced by policy π while $P(h|\pi)$ is the probability of history h induced by policy π and is given by:

$$P(h|\pi) = \prod_{i \geq 0} P_{\pi(s_i)}(s_{i+1}|s_i) \tag{4.3}$$

Now, a planning problem is actually an optimization problem in which we attempt to find an optimal policy. A policy is optimal if its expected utility is greater or equal than any other policy's expected utility. A solution to the planning problem is, as expected, an optimal policy.

Before going in to the next section where we talk about algorithms for solving MDPs we would like to discuss a few things. First, the utility function used in the definition of an MDP is the general case where we have both costs and rewards with costs $C : S \times A \to \mathcal{R}$

being what is known as a transition function and rewards $R : S \rightarrow \mathcal{R}$ being what is known as a state function. For a moment forget about the terms costs and rewards and focus on the terms transition functions and state functions. Costs and rewards are concepts that can be used for both transition functions and state functions. This is important because using the terms costs and rewards can lead to confusion in several ways one of which is the following: For instance, one can restrict the values of functions to the positive real numbers ($\mathcal{R}_{>0}$) instead of all real numbers ($\mathcal{R}$) and call both transition and state functions as reward functions that combined give us the utility function. In this case the utility function is the summation of the two reward functions instead of their subtraction. In addition, when using MDPs we can make simplifications such as that the underlying stochastic system deals only with transition functions and not state functions. As such, we would get rid of the terms referring to "reward" (state functions) where applicable. For instance, in equation 4.1 we would get rid of $R(s_i)$ and invert the sign. Having said that, it is better to view Definition 4.30 as a definition that provides the formulation of MDPs with both transition and state functions and by extension, to view what we have presented in this section as formulations that apply to this case.

4.4.1.1 Planning Algorithms

Continuing from the discussion in the last paragraph of the previous section, in this section we will briefly describe two main algorithms for solving MDPs whose utility functions are transition functions. Let us use the term reward here to denote that we want to maximize the expected utility.

The first algorithm is known as the *Policy Iteration* algorithm [3]. The rationale behind this algorithm is that given a stochastic system Σ, a utility function in the form of a transition function representing rewards and a discount factor γ, the algorithm calculates the optimal policy by alternating between two phases. The first phase which is known as the value determination phase in which the expected reward is calculated for the current policy and the second phase which is known as the policy improvement phase. During this phase, the current policy is refined to a new policy that has a higher expected reward (maximization). The algorithm terminates returning the current policy when there are no alternative actions that can improve the policy further.

The second algorithm is the *Value Iteration* algorithm [3]. Again, the input to the algorithm is a stochastic system Σ, a utility function in the form of a transition function representing rewards and a discount factor γ. The algorithm starts by randomly assigning an estimated reward for each state $s \in S$. The next step is to iteratively refine the value for each state by selecting an action that maximizes its expected reward. At each iteration step the value of the expected reward is computed for each state based on the previous value, i.e. the expected reward value for that state at the previous step. The algorithm finds an optimal, i.e. that maximizes reward, action for each state $s \in S$ and stores it in the policy. The algorithm terminates after the expected reward for two consecutive policies differs less

that some arbitrarily very small number ε which is chosen by the user.

4.4.2 Other Approaches

In Section 4.4.1 we presented an approach for planning under uncertainty that is based on MDPs. Under the aforementioned setting, we dealt with non-determinism and extended goals.

Another approach for planning under uncertainty is the utilization of Partially Observable Markov Decision Processes (POMDPs) [3], [135]. This approach is intended for domains in which we do not have complete knowledge about the state of Σ which contradicts assumption A1 (see Section 4.1) about a fully observable Σ. The main difference between POMDPs and MDPs is that in the case of POMDPs we can observe a probability distribution over the states of the system called belief states whereas in MDPS we can observe the state exactly. As such, in the POMDP setting plans are again policies but over belief states. In addition, planning problems are optimization problems like in the case of MDPs with the usage of utility functions. Regarding planning, one approach is to utilize the same algorithms that where described in Section 4.4.1.1 with modifications that will allow them to operate over belief states. However, it is more efficient to utilize algorithms that provide approximations of the optimal policy because solving POMDPs exactly entails high computational complexity and can, in practice, be applied only to problems of small size [3], [136].

Yet another approach for planning under uncertainty in which we can have non-determinism, partial observability and extended goals is model checking [3]. At the very heart of the model checking approach lies a non-deterministic state transition system which as opposed to the case of MDPs and POMDPs is not associated with probabilities. The execution of an action to a state can lead the system to many different states. Goals are expressed in temporal logic. As such, given a non-deterministic state transition system and a formula in temporal logic, planning by model checking involves the generation of plans that lead the evolution of the system in a way that all the behaviours of the system make the formula true [3]. Algorithms for planning under the model checking paradigm can use symbolic model checking techniques [137], [138], [139].

4.5 The Planning Domain Definition Language

The Planning Domain Definition Language (PDDL) is an action-centred planning language that is inspired by the STRIPS planning language [140] and was initially developed by McDermott et al. [141] for the 1998 International Planning Competition (IPC). Each PDDL action comprises preconditions, i.e. conditions for it to be applicable, and postconditions, i.e. the effects of executing the action. PDDL separates the representation of domains from the representation of problems that need solving within a domain. As such, given a domain, various problems can be provided to a PDDL planner for generating plans.

Although the initial version of PDDL was successful in making PDDL a standard among the planning community of IPC competitions the language received criticism for its lack of applicability in real world problems [142]. In the light of this criticism, Fox and Long [142] presented a new version of PDDL which is known as PDDL2.1. Following the PDDL2.1 standard, modelling numeric fluents and not just binary ones was now possible. In this manner, resources such as time, energy and distance could now be represented and monitored. Moreover, the aforementioned standard introduced the concept of plan metrics the usage of which allowed minimization/maximization of fluents. This took the language one step further since planning was no longer just goal-driven but also utility-driven, i.e. the minimization/maximization of fluents could now be used complementary to planning goals. Since PDDL2.1 which revolutionised PDDL additional versions appeared, providing various improvements. However, it is outside the scope of this thesis to present a full review of PDDL. The interested reader is referred to Ghallab et al. [134]. However, we will provide an intuition and definitions with respect to temporal planning domains, durative actions, planning problems as well as plan and plan metrics of PDDL2.1 (see Section 4.5.1).

4.5.1 The Planning Domain Definition Language 2.1

Let us first present the various building blocks of temporal domains in PDDL 2.1. A PDDL *temporal domain* contains *type*, *predicate*, *function* and *durative action* definitions.

Domain *types* are the types of the objects in the environment that are of interest[7] and are used in the definition of domain *predicates*, *functions* and *durative actions*. Domain *predicates* represent either properties or relations between objects of interest and are strictly binary. In contrast, domain *functions* are fluents that return numerical values. Both domain *predicates* and *functions* are also used in the definition of domain *durative actions*. Durative actions can be of two forms: discretised and continuous. Every durative action comprises *parameters*, *preconditions* and an *effects*. In order for a domain action to be applicable and be considered as part of a plan its precondition must hold before applying the action. Moreover its effect should be contributing towards solving the problem. Snippet 2 illustrates an exemplar temporal domain. The temporal domain illustrated in Snippet 2 comprises two *types* of objects, namely robots and locations. In addition, the domain comprises two *predicates*, namely `(at ?r - Robot ?l - Location)` and `(reachable ?l1 ?l2 - Location)`, which express whether some robot is at some location and whether some location is reachable form some other location respectively. We use the term some because `?r` is a variable that can be instantiated with a robot while `?l`, `?l1`, `?l2` are variables that can be instantiated with locations. Variables in PDDL are represented with the `?` symbol preceding them. Moreover, the domain comprises two *functions*, namely `(energy ?r - Robot)` and `(distance ?l1 ?l2 - Location)`, which are used for expressing the

[7] Objects of interest are not only limited to physical objects but can also be immaterial things such as waypoints

Snippet 2 PDDL exemplar domain with types, predicates, functions and a simple durative action for moving a robot from one location to another.

```
(define (domain example_1)
(:requirements :typing :fluents :durative-actions)
(:types Robot Location)
(:predicates
        (at ?r - Robot ?l - Location)
        (reachable ?l1 ?l2 - Location)
)
(:functions
        (energy ?r - Robot)
        (distance ?l1 ?l2 - Location)
)
(:durative-action move
 :parameters (?r - Robot ?from ?to - Location)
 :duration ( = ?duration (*(distance ?from ?to) 1))
 :condition (and (at start (at ?r ?from))
                 (at start (reachable ?from ?to)))
 :effect (and (at start (not (at ?r ?from)))
         (at end (at ?r ?to))
         (at end (decrease (energy ?r) (distance ?from ?to))))
)
)
```

remaining energy of some robot and the distance between two locations respectively. Let us know analyse the *durative action*, namely `move` illustrated in Snippet 2. The parameters of the action are some robot and two locations. Notice that the parameters are variables of the aforementioned types. As such `?from` is a variable representing the location from which a robot is going to start moving while `?to` is a variable representing the location to which the same robot is going to move. `Duration` represents the duration of the action and is equal to the distance between the two locations multiplied by one. Number one here is a random selection for demonstration purposes. It effectively represents a multiplication factor for estimating duration based on the distance travelled. Now, `condition` represents the preconditions of the action. In the case of our example one of the the preconditions for the move action is that the robot needs to be at the location represented by the variable `?from`. The term `at start` represents a temporal annotation that enforces a temporal constraint. That is, in the beginning (at start) of the interval of the action the robot represented by the `?r` variable needs to be at the location represented by the `?from` variable. In PDDL2.1 we can have three temporal annotations in total that form constraints, namely `at start`, `at end` and `over all`. Temporal annotations `at end` and `over all` refer to the end and the full interval (from start to end) of the action respectively. Note that `over all` refers to an open interval instead of a closed one, i.e. (t_1, t_2) instead of $[t_1, t_2]$. As such, precondition `at start (reachable ?from ?to)` states that the location towards the robot is moving must be accessible at the start of the interval of the action (the point at which the action is applied). Continuing with our analysis, the `effect` (postcondition(s)) of the `move` action is that immediately after starting the execution of the action the robot is not at its initial location while at the end of the action execution the robot is at the location it has moved

to and, at the end again, the remaining energy of the robot decreases relatively to the distance travelled, i.e. the distance between the two locations. At this point we would like to emphasize that the durative action illustrated in Snippet 2 is a discrete action because its effects occur only at the start and end points of the action's time interval. The action could be transformed into a continuous one by encoding for example:

$$\text{(decrease (energy ?r) } (* \text{ \#t 1))} \tag{4.4}$$

instead of:

$$\text{(at end (decrease (energy ?r) (distance ?from ?to)))} \tag{4.5}$$

where, #t refers to the continuously changing time from the start of a durative action during its execution [142] and number 1 represents the rate at which remaining energy is decreasing. In our example 1 is an arbitrarily chosen number for demonstration purposes. Note that the terms discrete and continuous durative actions refer to effects and not their preconditions.

Let us now present an exemplar problem for the domain illustrated in Snippet 2. The purpose of the exemplar problem is to demonstrate PDDL problems and their components. Snippet 3 illustrates such a problem. By observing Snippet 3 we can see that a PDDL

Snippet 3 PDDL exemplar problem for the domain defined in Snippet 2.

```
(define (problem example_1)
(:domain example_1)
(:objects location1 location2 location3 location4 - Location
          robot1 - Robot
)
(:goal
          (and
          (at robot1 location4)
          (>= (energy robot1) 0))
)
(:init
          (= (energy robot1) 100)
          (at robot1 location1)
          (reachable location1 location2)
          (reachable location1 location3)
          (reachable location2 location4)
          (reachable location3 location4)
          (= (distance location1 location2) 10)
          (= (distance location1 location3) 10)
          (= (distance location2 location4) 5)
          (= (distance location3 location4) 2)
)
          (:metric minimize (total-time))
)
```

`problem` consists of three main building blocks, that is, *objects*, an *initial state* and a goal state. Recall from the beginning of this section that we talked about the types of the objects that are of interest. Those objects and problem objects are the same thing. The difference

Table 4.1. Plan solving the problem illustrated in Snippet 3 given the temporal domain illustrated in Snippet 2.

Plan
(0.000: move robot1 location1 location3) [10.000]
(10.000: move robot1 location3 location4) [2.000]

is that in a PDDL problem reside the objects themselves along with their types. In order to encode both the initial and the goal states in a PDDL problem we utilize the predicates and functions that are defined in the PDDL domain instantiated with the objects. This is, logical since the objects are the ones whose states need to change given the goal. In order to better understand the building blocks of PDDL problems let us analyse the problem presented in Snippet 3. We have in total five objects of which four are locations and one is a robot. The initial state of the problem represents that the robot (robot1) is at location1 and the robot has energy equal to 100 units. Moreover, the initial state represents which locations are reachable from other locations. In our case location2 is reachable only form location1 as is location3. Furthermore, the initial state also encodes the distances between the locations that are reachable from each other. Regarding the goal state, it encodes that the robot should be at location4 and its energy at that location must be $>= 0$. There is one last thing to observe in Snippet 3 and that is the metric at the end of the problem. PDDL2.1 supports metrics for optimizing criteria in the form of minimization or maximization. Such criteria are optional as opposed to goals. That is, they can be omitted. In our case the metric instructs the planner to devise a plan which will be minimizing the total time[8]. That is, the planner will attempt to devise a plan which will satisfy the goal state, given the initial state of course and the objects but at the same time the devised plan needs to be such that will minimize the temporal span of the entire plan. Now, given the temporal domain and the problem illustrated in Snippets 2 and 3 respectively we can use a PDDL planner that supports PDDL2.1 to formulate a temporal plan (see Table 4.1). The inputs to the planner are the temporal domain and the problem. In fact, every PDDL planner requires a domain and a problem as inputs in order to generate a plan. For generating the plan illustrated in Table 4.1 we used the OPTIC temporal planner. OPTIC is a planner that is briefly described in Section 4.6. As far as the illustrated plan is concerned, the numbers on the left represent the time points at which each action will start executing while the numbers on the right the duration of each action. As such, one can see that robot1 will start moving from location1 towards location3 at time point $t_1 = 0$ while the time interval for that action (duration) is 10 seconds. The next action will commence at time point say $t_2 = 10$ which is the starting time point for the action through which robot1 will transition from location3 to location4. duration of that second action will be 2 sections. As such the duration of the whole plan will be 12 seconds in total. At this point we would like to emphasize the fact that since we have durative actions that are situated in time we can have concurrency, i.e. more than one actions executing at the same time. Let us now provide some core definitions as well as

[8]The term *total-time* shown in Snippet 3 is build-in in PDDL2.1 and refers to the temporal span of the entire plan.

definitions of what we have presented so far. Note that we will not be providing definitions for continuous durative actions as it is outside the score of this section, and of this thesis. We have however, touched upon the concept of continuous durative actions earlier in this section. The definitions are taken directly from the foundational paper of PDDL2.1 [142].

For better readability let us first provide a "roadmap" for the definitions so that the interested reader can navigate through them easier. Definition 4.31 defines simple planning instances, Definition 4.32 defines logical states and states. Definition 4.33 defines the assignment proposition. Definition 4.34 defines the normalization of ground propositions. Definition 4.35 defines actions and ground actions. Definition 4.36 defines valid ground actions. Definition 4.37 defines updating functions for valid ground actions. Definition 4.38 defines when ground propositions are satisfied. Definition 4.39 provides a definition for the applicability of ground actions. Definition 4.40 defines simple plans. Definition 4.41 provides a definition for mutex (mutually exclusive) actions. Definition 4.42 provides a definition for the execution of happenings (see also Definition 4.40). Definition 4.43 provides a definition for the executability of simple plans while Definition 4.44 provides a definition for the validity of simple plans. Definition 4.45 provides a definition for discrete durative and ground durative actions. Definition 4.46 defines plans with discrete durative actions while Definition 4.47 defines induced simple plans given Definition 4.46 for plans. Finally, Definition 4.48 provides a definition for the executability of plans containing discrete durative actions while Definition 4.49 defines what it means for a plan containing discrete durative actions to be valid. As a final remark before proceeding with the definitions we would like to urge the reader to come back to this section and all the definitions that will be presented here in the event that he/she has any doubts about the PDDL code presented in Chapters 7, 8.

Definition 4.31 *A simple planning instance is defined to be a pair* $I = (Dom, Prob)$ *where* $Dom = (Fs, Prs, As, arity)$ *is a 4-tuple consisting of finite sets of function symbols, predicate symbols, non-durative actions, and a function arity mapping all of these symbols to their respective arities.* $Prob = (Os, Init, G)$ *is a triple consisting of the objects in the domain, the initial state specification and the goal state specification respectively. The primitive numeric expressions of a planning instance (PNEs) are the terms constructed from the function symbols of the domain applied to (an appropriate number of) objects drawn from Os. The dimension of the planning instance (dim), is the number of distinct primitive numeric expressions that can be constructed in the instance. The atoms of the planning instance (Atms), are the (finitely many) expressions formed by applying the predicate symbols in Prs to the objects in Os (respecting arities). Init consists of two parts:* $Init_{logical}$ *and* $Init_{numeric}$. $Init_{logical}$ *is a set of literals formed from the atoms in Atms while* $Init_n$*umeric is a set of propositions asserting the initial values of a subset of the primitive numeric expressions of the domain. These assertions each assign to a single primitive numeric expression a constant real value. The goal condition is a proposition that can include both atoms formed from the relation symbols and objects of the planning instance and numeric*

propositions between primitive numeric expressions and numbers. As is a collection of action schemas (non-durative actions) each expressed in the syntax of PDDL. The primitive numeric expression schemas and atom schemas used in these action schemas are formed from the function symbols and predicate symbols (used with appropriate arities) defined in the domain applied to objects in Os and the schema variables.

Definition 4.32 *Given the finite collection of atoms for a planning instance I, $Atms_I$, a logical state is a subset of $Atms_I$. For a planning instance with dimension dim, a state is a tuple in $(\mathbb{R}, \mathbb{P}(AtmsI), \mathbb{R}_{\perp}^{dim})$ where $\mathbb{R}_{\perp} = \mathbb{R} \cup \{\perp\}$, and $\perp$ denotes the undefined value. The first value is the time of the state, the second is the logical state and the third value is the vector of the dim values of the dim primitive numeric expressions in the planning instance. The initial state for a planning instance is $(0, Init_{logical}, \mathbf{x})$ where $\mathbf{x}$ is the vector of values in $\mathbb{R}_{\perp}$ corresponding to the initial assignments given by $Init_{numeric}$ (treating unspecified values as $\perp$).*

Definition 4.33 *The syntactic form of a numeric effect consists of three things. First, one of the assignment operators (assign, increase, decrease, scale-up, scale-down. The assign operator assigns a value, the increase operator increases a value and it represents the C programming language style +=, the decrease operator decreases a value and it represents the C programming language style -=, the scale-up operator scales up a value and it represents the C programming language style *= and the scale-down operator scales down a value and it represents the C programming language style /=. Second, one primitive numeric expression, referred to as the lvalue. Third, a numeric expression (which is an arithmetic expression whose terms are numbers and primitive numeric expressions), referred to as the rvalue. The assignment proposition corresponding to a numeric effect is formed by replacing the assignment operator with its equivalent arithmetic operation (that is $(increasepq)$ becomes $(= p(+pq))$ and so on) and then annotating the lvalue with a "prime". A numeric effect in which the assignment operator is either increase or decrease is called an additive assignment effect, one in which the operator is either scale-up or scale-down is called a scaling assignment effect and all others are called simple assignment effects.*

Definition 4.34 *Let I be a planning instance of dimension dim_I and let $index_I : PNEs_I \rightarrow \{1, ..., dim\}$ be an (instance-dependent) correspondence between the primitive numeric expressions and integer indices into the elements of a vector of dim_I real values, $\mathbb{R}_{\perp}^{dim_I}$. The normalised form of a ground proposition, p, in I is defined to be the result of substituting for each primitive numeric expression f in p, the literal $X_{index_I(f)}$. The normalised form of p will be referred to as $\mathcal{N}(p)$. Numeric effects are normalised by first converting them into assignment propositions. Primed primitive numeric expressions are replaced with their corresponding primed literals. $\mathbf{X}$ is used to represent the vector $\langle X_1 ... X_n \rangle$.*

For the next definition of actions and ground actions (Definition 4.35) we are going to assume that we have no quantifiers and no conditional effects. For a definition considering quantifiers and conditional effects please refer to Fox and Long [142].

Definition 4.35 *Given a planning instance, I, containing an action schema $A \in As_I$, the set of ground actions for A, GA_A, is defined to be the set of all the structures, α, formed by substituting objects for each of the schema variables in each schema, X, where the components of α are:*

- *Name is the name from the action schema, X, together with the values substituted for the parameters of X in forming α.*
- *Pre_α, the precondition of α, is the propositional precondition of α. The set of ground atoms that appear in Pre_α is referred to as $GPre_\alpha$.*
- *Add_α, the positive postcondition of α, is the set of ground atoms that are asserted as positive literals in the effect of α.*
- *Del_α, the negative postcondition of α, is the set of ground atoms that are asserted as negative literals in the effect of α.*
- *NP_α, the numeric postcondition of α, is the set of all assignment propositions corresponding to the numeric effects of α.*

The following sets of primitive numeric expressions are defined for each ground action, $\alpha \in GA_A$:

- $L_\alpha = \{f | f$ *appears as an lvalue in* $\alpha\}$
- $R_\alpha = \{f | f$ *is a primitive numeric expression in an rvalue in* α *or appears in* $Pre_\alpha\}$
- $L^*_\alpha = \{f | f$ *appears as an lvalue in an additive assignment effect in* $\alpha\}$

Definition 4.36 *Let α be a ground action. α is valid if no primitive numeric expression appears as an lvalue in more than one simple assignment effect, or in more than one different type of assignment effect.*

Definition 4.37 *Let α be a valid ground action. The updating function for α is the composition of the set of functions: $\{NPF_p : \mathbb{R}^{dim}_\perp \to \mathbb{R}^{dim}_\perp | p \in NP_\alpha\}$ such that $NPF_p(\mathbf{x}) = \mathbf{x}'$ where for each primitive numeric expression x'_i that does not appear as an lvalue in $\mathcal{N}(p)$, $x'_i = x_i$ and $\mathcal{N}(p)[\mathbf{X}' := \mathbf{x}', \mathbf{X} := \mathbf{x}]$ is satisfied. The notation $\mathcal{N}(p)[\mathbf{X}' := \mathbf{x}', \mathbf{X} := \mathbf{x}]$ should be read as the result of normalising p and then substituting the vector of actual values $\mathbf{x}'$ or the parameters $\mathbf{X}'$ and actual values $\mathbf{x}$ for formal parameters $\mathbf{X}$.*

Definition 4.38 *Given a logical state, s, a ground propositional formula of PDDL2.1, p, defines a predicate on $\mathbb{R}^{dim}_\perp$, $Num(s,p)$, as follows: $Num(s,p)(\mathbf{x})$ if and only if $s \models \mathcal{N}(p)[\mathbf{X} := \mathbf{x}]$ where, $s \models q$ means that q is true under the interpretation in which each atom, α, that is not a numeric comparison, is assigned true if and only if $\alpha \in s$, each numeric comparison is interpreted using standard equality and ordering for reals and logical connectives are given their usual interpretations. p is satisfied in a state $(t,s,\mathbf{X})$ if*

$Num(s,p)(\mathbf{X})$. Comparisons involving $\perp$, including direct equality between two $\perp$ values are all undefined, so that enclosing propositions are also undefined and not satisfied in any state.

Definition 4.39 *Let α be a ground action. α is applicable in a state s if the Pre_α is satisfied in s.*

Definition 4.40 *A simple plan, SP, for a planning instance, I, consists of a finite collection of timed simple actions which are pairs (t,a), where t is a rational-valued time and α is an action name. The happening sequence, $\{t_i\}_{i=0\ldots k}$ for SP is the ordered sequence of times in the set of times appearing in the timed simple actions in SP. All t_i must be greater than 0. It is possible for the sequence to be empty (an empty plan). The happening at time t, E_t, where t is in the happening sequence of SP, is the set of (simple) action names that appear in timed simple actions associated with the time t in SP.*

Definition 4.41 *Two ground actions, α and β, are non-interfering if:*

- $GPre_\alpha \cap (Add_\beta \cup Del_\beta) = GPre_\beta \cap (Add_\alpha \cup Del_\alpha) = \emptyset$ *and*
- $Add_\alpha \cap Del_\beta = Add_\beta \cap Del_\alpha = \emptyset$ *and*
- $L_\alpha \cap Pr_\beta = Pr_\alpha \cap L_\beta = \emptyset$ *and*
- $L_\alpha \cap L_\beta \subseteq L^*_\alpha \cup L^*_\beta$

If two actions are not non-interfering they are mutex.

Definition 4.42 *Given a state, $(t,s,\mathbf{x})$ and a happening, H, the activity for H is the set of ground actions $A_H = \{\alpha | \text{the name for } \alpha \text{ is in } H,\ \alpha \text{ is valid and } Pre_\alpha \text{ is satisfied in } (t,s,\mathbf{x})\}$. The result of executing a happening, H, associated with time t_H, in a state $(t,s,\mathbf{x})$ is undefined if $|A_H| \neq |H|$ or if any pair of actions in A_H is mutex. Otherwise, it is the state $(t_H, s', \mathbf{x}')$ where:*

- $s' = (s \setminus \bigcup_{\alpha \in A_H} Del_\alpha) \cup \bigcup_{\alpha \in A_H} Add_\alpha$
- $\mathbf{x}'$ *is the result of applying the composition of the functions $\{NPF_\alpha | \alpha \in A_H\}$ to $\mathbf{x}$.*

Definition 4.43 *A simple plan, SP, for a planning instance, I, is executable if it defines a happening sequence, $\{t_i\}_{i=0\ldots k}$, and there is a sequence of states, $\{S_i\}_{i=0\ldots k+1}$ such that S_0 is the initial state for the planning instance and for each $i = 0\ldots k$, S_{i+1} is the result of executing the happening at time t_i in SP. The state S_{k+1} is called the final state produced by SP and the state sequence $\{S_i\}_{i=0\ldots k+1}$ is called the trace of SP. Note that an executable plan produces a unique trace.*

Definition 4.44 *A simple plan SP, for a planning instance, I, is valid if it is executable and produces a final state S, such that the goal specification for I is satisfied in S.*

As is the case of the definition of simple ground actions (Definition 4.35) we are going to present a definition for discrete durative ground actions without considering quantifiers or conditional effects (see Definition 4.45). For a definition considering quantifiers and conditional effects please refer to Fox and Long [142].

Definition 4.45 *Durative actions are grounded in the same way as simple actions (see Definition 4.35), by replacing their formal parameters with constants from the planning instance. The definition of durative actions requires that the condition be a conjunction of temporally annotated propositions. Each temporally annotated proposition is of the form:* (at start p) *or* (at end p) *or* (over all p)*, where p is a non-annotated proposition. Similarly, the effects of a durative action (without continuous or conditional effects) are a conjunction of temporally annotated simple effects. The duration field of DA defines a conjunction of propositions that can be separated into* DC^{DA}_{start} *and* DC^{DA}_{end}*, the duration conditions (DC) for the start and end of DA, with terms being arithmetic expressions and ?duration. The separation is conducted in the obvious way, placing at start conditions into* DC^{DA}_{start} *and at end conditions into* DC^{DA}_{end}*. Each ground durative action, DA, with no continuous effects and no conditional effects defines two parametrised simple actions* DA_{start} *and* DA_{end}*, where the parameter is the ?duration value, and a single additional simple action* DA_{inv} *as follows.* DA_{start} *(*DA_{end}*) has precondition equal to the conjunction of the set of all propositions, p, such that* (at start p) *(*(at end p)*) is a condition of DA, ogether with* DC^{DA}_{start} *(*DC^{DA}_{end}*), and effect equal to the conjunction of all the simple effects, e, such that (at start e) (*(at end e)*) is an effect of DA (respectively).* DA_{inv}*, is defined to be the simple action with precondition equal to the conjunction of all propositions, p, such that* (overallp) *is a condition of DA. It has an empty effect. Every conjunct in the condition of DA contributes to the precondition of precisely one of* DA_{start}*,* DA_{end} *or* DA_{inv}*. Every conjunct in the effect of DA contributes to the effect of precisely one of* DA_{start} *or* DA_{end}*. For convenience,* DA_{start}*, (*DA_{end}*,* DA_{inv}*) will be used to refer to both the entire (respective) simple action and also to just its name.*

Definition 4.46 *A plan, P, with durative actions, for a planning instance, I, consists of a finite collection of timed actions which are pairs, each either of the form* (t, α)*, where t is a rational-valued time and* α *is a simple action name an action schema name together with the constants instantiating the arguments of the schema, or of the form* $(t, \alpha[t'])$*, where t is a rational-valued time,* α *is a durative action name and* t' *is a non-negative rational-valued duration.*

Definition 4.47 *If P is a plan then the happening sequence for P is* $\{t_i\}_{i=0...k}$*, the ordered sequence of time points formed from the set of times* $\{t|(t, \alpha) \in P$ *or* $(t, \alpha[t']) \in P$ *or* $(t - t', \alpha[t']) \in P\}$*. The induced simple plan for a plan P, simplify(P), is the set of pairs defined as follows:*

- (t, α) *for each* $(t, \alpha) \in P$*, where* α *is a simple (non-durative) action name.*

- $(t, \alpha_{start}[?\textbf{duration} := t'])$ *and* $(t+t', \alpha_{end}[?\textbf{duration} := t'])$ *(these expressions are simple timed actions the square brackets denote substitution of* t' *for* ?**duration** *in this case) for all pairs* $(t, \alpha[t']) \in P$*, where* α *is a durative action name.*

- $((t_i + t_{i+1})/2, \alpha_{inv})$ *for each pair* $(t, \alpha[t']) \in P$ *and for each i such that* $t \leq t_i < t+t'$*, where* t_i *and* t_{i+1} *are in the happening sequence for P.*

Definition 4.48 *A plan, P (for a planning instance), is executable if the induced simple plan for P, simplify(P) is executable, producing the trace* $\{S_i = (t_i, s_i, \mathbf{v}_i)\}_{i=0\ldots k}$.

Definition 4.49 *A plan, P (for a planning instance), is valid if it is executable and if the goal specification is satisfied in the final state produced by its induced simple plan.*

4.6 OPTIC PDDL Planner

OPTIC is a temporal planner which combines forward search with ideas from partial-order planning. It supports the encoding of preferences as they are defined in the PDDL3 standard [143] and time-dependant goal achievement costs. Broadly speaking, preferences can be of two categories. The first category refers to preferences that are either soft preconditions on actions or soft goals. Such preferences are known as "simple" [144]. The second category comprises "complex" preferences that are formulated using special operators [144], [143]. In short, preferences in PDDL3 represent soft constraints in the form of logical expressions. As opposed to "traditional" hard constraints their satisfaction is not necessary but non-satisfaction can affect the quality of the plan. As far as time-dependent goal achievement costs two types are supported in OPTIC: i) those with deadlines that must be met on time (refer to Gerevini and Long [143]) and ii) those with costs that increase over time.

It is outside the scope of this section and of this thesis to deal with preferences and time-dependent costs. As such, OPTIC will be used in this thesis for handing temporal planning as it is specified by the PDDL2.1 standard. The interested reader is advised to follow the references within (this section) for a detailed analysis.

4.7 Work on Planning for Autonomous Robots

In this section we present research work on planning for autonomous robots. The problem of planning for autonomous robots is one of dealing with unexpected events, uncertainty, faults (hardware, software), and failures during task/mission execution. For this problem various approaches have been developed with each approach tackling aspects[9] of the problem.

[9] Aspects refer to the challenges in which the problem can materialize. For instance, planning for autonomy under hardware faults is different than planning for autonomy under observation uncertainty. However, that is not to say that both aspects do not fall under the same general problem; that of planning for autonomy.

The work presented by Saigol in [145] is concerned with planning for autonomous hydrothermal vent prospecting using AUVs. In doing so the author formulated the problem as a POMDP in order to deal with limited information on the location of vents caused by the uncertainty arising from vehicle sensor readings. The experimental results demonstrate the efficiency of the system in locating hydrothermal vents.

Another research output is the work presented by Fox et. al. in [146] where plan-based policy learning is utilised for tracking biological ocean features autonomously using AUVs. More specifically, the authors have applied the aforementioned approach for tracking the surfaces of harmful algal blooms. Tracking is modelled as a planning problem where an AUV has to decide how detect the edges of blooms and navigate their boundaries while at the same time conserving energy and avoiding danger [146]. The rationale behind plan-based policy learning is to exploit sampling and classification in order to learn a policy for tracking. As such the problem is modelled as a deterministic one using PDDL. Several instances of the deterministic problem are chosen (by sampling) in simulation in a way that they cover a variety of situations likely to be seen in the real world and they are then solved by a PDDL planer. Each solution is used as a training example for learning a policy. That is, given the set of solutions to the deterministic problems, decision-tree learning is applied to classifying decision variables with action choices forming in this way state-action pairs that are effectively the policy. In this manner, although uncertainty arising from dispersion and change of the blooms is not directly modelled, the choice of "interesting" problem instances that reflect situations likely to be seen in the real world lead to robust solutions. However, as the authors state, producing robust policies can take several days of training. Of course, the amount of days is dependant on the number of samples used for training which in turn affects the robustness of the policy learned.

T-REX is another research output with respect to planning for autonomy [147]. It was developed at the Monterey Bay Aquarium Research Institute (MBARI) for adaptive mission control of AUVs. T-REX is an adaptive constraint-based temporal planning system that is based on NASA'S EUROPA (Extensible Universal Remote Operations Planning Architecture) version 2, a library and tool set for building planners that follow the constraint-based temporal planning approach [148]. T-Rex comprises control agents (Teleo-Reactors) each of which follows a Sense-Plan-Act (SPA) approach to control. Each control agent has a different functional and temporal scope [149]. The functional scope refers to the goals the agent: i) understands, ii) can plan for, iii) monitor the progress of. The temporal scope on the other hand refers to how much time in the future the agent has to generate a plan for. Experimental results from using the framework over a period of a year in various sea trials have shown that by utilising a partial plan representation that can be both refined and extended during mission execution can minimize replanning occurrences due to effectively invalidating a mission plan [150].

Of particular interest is the PANDORA project [151] which was focused on persistent autonomy for AUVs. One of the aspects of the project was to focus on AUVs undertaking underwater inspection tasks of submerged structures and intervention tasks in the form of

valve-turning and chain-cleaning in the presence of unexpected events and faults. Valve-turning refers to an intervention task in which a valve in an underwater structure needs to be turned, for instance in the case of an underwater oil rig, while chain-cleaning refers to an intervention task in which a chain, e.g. an anchor chain, needs to be cleaned. The approach followed was deterministic PDDL-based temporal planning [152] with replanning in the presence of vehicle component faults and action failures demonstrating in this way an adaptive behaviour. Knowledge about the environment, the vehicle state its components, capabilities and available actions is expressed through ontologies which the planning and execution system also benefits from in order to devise plans and execute them in order to undertake the aforementioned tasks. Experimental results have shown the ability of the framework to successfully adapt under task failures and faults. Earlier we have stated that PANDORA is of particular interest. This interest stems from the conceptual closeness to our project and as such we will see in Chapter 7 how PANDORA and our project are related and how they are different in the context of persistently autonomous operations. One last remark to make about PANDORA is that its planning and execution system was later materialised as a standalone framework called ROSPlan [153]. ROSPlan, as its name suggests, is a framework for planning tasks within the Robot Operating System (ROS) (for ROS see Chapter 6). We will talk more about ROSPlan in Chapter 7.

4.8 Summary and Discussion

This chapter was concerned with AI planning. Initially we presented an overview of the field and provided a conceptual model for planning where we described the various restrictive assumptions that can be made for planning. Conceptually, the chapter can be divided into three main parts: i) classical planning, ii) temporal planning and iii) planning under uncertainty.

Regarding classical planning, we have presented three main representation approaches, namely the set-theoretic, classical and state-variable representations. For each representation we provided formal definitions for planning domains, actions, problems, plans and solutions. Under the set-theoretic representation world states are propositions while each action specifies which propositions are part of the state so that the action is applicable to that state and which propositions will be added or removed in order to create a new world state. In contrast, the classical representation utilises first-order literals and logical connectives instead of propositions. As far as the state-variable representation is concerned, states are represented as tuples of values of state variables while actions are represented by partial functions that map state tuples into other state tuples of different values i.e, other states. Furthermore, we described algorithmic families for solving classical planning problems. However, classical planning is quite limiting for real world applications due to the restrictive assumptions it makes. Among others, the implicit time assumption as well as the static and deterministic state transition system assumptions.

Regarding temporal planning, we have presented it through an approach based on tem-

porally qualifying expressions and temporal databases without formalising a specific language but rather focusing on the semantics of the underlying representation. In formally presenting definitions for temporally qualifying expressions, temporal databases, domains, actions problems, plans and solutions we have utilised notions and concepts from point and temporal algebras. Temporal planning does not have the assumption about implicit time as opposed to classical planning. As such actions have durations and can overlap since they are situated in time. Consequently temporal planning is more suited for real world applications.

With respect to planning under uncertainty, we do not assume a deterministic state-transition system. In addition, we do not assume that goals are restricted as opposed to classical planning in which we make both assumptions. The lack of the latter assumption helps us to specify goals of different strengths while the lack of the former allows as to encode actions that have non-deterministic effects. One of the approaches that we presented is based on planning with Markov decision processes where we provided formal definitions about the process and described planning algorithms. In addition we have presented a variation of Markov decision processes (MDPs) called partially observable Markov decision processes (POMDPs) which can be used when we do not make the assumption of a fully observable state-transition system. Both in the case of MDPs and POMDPs planning is an optimization problem of utility functions. Both MDPs and POMDPs utilise probabilities for handing uncertainty. Finally, we have briefly described the model checking approach for planning under uncertainty which as opposed to MDPs and POMDPs does not utilise probabilities for encoding uncertainty but rather a general non-deterministic state transition system in which the execution of an action can lead to many different states. Again, as in the case of temporal planning, planning under uncertainty approaches are better suited for real world applications.

Further to the presentation of the classical, temporal as well as the planning under uncertainty fields within the AI planning domain, we have focused on PDDL and specifically PDDL2.1 which supports temporal planning. We have done so because we will be utilising temporal planning which is based on PDDL temporal planners. PDDL2.1 was presented with formal definitions about domains, durative actions, problems etc. However, we focused on discrete durative actions rather than continuous ones with formal definitions since they will be used in the thesis. In the context of temporal planning and PDDL we have also briefly described OPTIC, a PDDL temporal planner.

Finally, in this chapter we have presented research outputs with respect to planning for autonomy. The approaches presented include POMDPs, plan-based policy learning and temporal planning. Of particular interest is the PANDORA project due to the conceptual closeness of our project and ROSPlan, a standalone framework for task planning within ROS.

Chapter 5

Maritime Situation Awareness

As was stated early on in the thesis, where we presented our problem statement (see Sections 1.1, 1.1.2), maritime situation awareness plays a key role in promoting maritime security. Maritime security is one of the two aspects of the maritime defence and security domain[1] which this thesis is concerned with and as such the purpose of this chapter is to present an overview of the field of maritime situation awareness. In doing so, we present common sources of information in building maritime situation awareness in Section 5.1. In Section 5.2 we analyse the role of context, prior knowledge, and uncertainty in the domain while in Section 5.3 we present data fusion models that are commonly used in the field. Finally, in Section 5.4 we present pieces of work on maritime situation awareness systems.

5.1 Sources of Information

When referring to situation awareness in general, the first thing that comes to mind is information. Their nature, quality, and quantity have a high impact on the correctness of the situation awareness that will be built and by extension on the decisions that will need to be taken in the event that an incident is identified. The purpose of this section is to present common sources of information used in building maritime situation awareness.

One of the sources of information commonly used in building maritime situation awareness is an Automatic Identification System (AIS) installed on ships. AIS transponders transmit data periodically which among others contain information about the vessel's position, speed and destination [7]. Another source of information commonly used is coastal radars which detect and track vessel movements within their range. Satellite imagery is yet another source of information that is used for tracking vessels and estimating their speeds. Satellite images can also function complementary in the sense that they provide a natural view of the area they capture. The information gathered from the aforementioned sources can provide valuable insight on the situation at hand. In many cases weather conditions can have an impact on a vessel's route. For instance, in case of stormy weather and rough sea in an area the captain may choose to avoid that area and follow an alternative route to ensure

[1] The other, obviously, being maritime defence.

safety [7]. As such, weather conditions are considered another source of information used in building maritime situation awareness. Outwith information originating from sensors, valuable information gained from human intelligence reports, if any, are also used. Such information is gathered from various sources: intelligence agencies that have infiltrated organizations involved in illegal activities, accessaries that have been caught and turned into informants as well as "competitors" that seek to destroy rival organizations involved in the same or similar illegal activity to name but a few.

5.2 The Role of Context, Uncertainty and Prior Knowledge

It is an undeniable fact that sensory information as well as other information originating from human intelligence reports play an important role in building maritime situation awareness. Another equally important factor however, is the exploitation of such information in context by using contextual knowledge to ensure that such information is interpreted appropriately. The existence of different contexts affects the relations formed among different entities in the environment. This by extension leads to different contexts giving different meaning to the same observations. Consider for instance maritime situation awareness in two different contexts: i) harbour surveillance and ii) piracy at open sea. Now consider the same observation of a vessel stopping: i) at a harbour and ii) at open sea. In the first context, such an observation is not abnormal since vessels generally stop in harbours while in the context of piracy at open sea, a vessel stopping at open sea is alarming in the sense that pirates usually board vessels and immobilise them. Except for different relations and interpretations arising form different contexts different relations and interpretations can arise from different "configurations" of the same context. A representative example of such a case is the existence of different operational regulations in harbours or even different harbour environments in the context of harbour surveillance [8]. In short, context in a situation awareness environment provides the relations among entities as well as their structure.

In a deterministic perception of maritime situation awareness, observations as well as relations among entities and their structure suffice to assess a situation. However, as in most real world applications, there is uncertainty arising both from observations themselves and potentially multiple probable outcomes given the same observations. As such, the existence of uncertainty can lead to incorrect interpretations and reduced situation awareness, converting the identification of incidents into a real challenge. This is highly undesirable, especially when extremely dangerous incidents may be missed, e.g. arms trafficking. That being said, prior knowledge can prove useful in "weighting" the importance of what we observe and enhance the interpretation and reasoning capabilities of human operators and situation awareness systems. To better understand the role of prior knowledge under uncertainty recall the example given in the beginning of this section and in particular the vessel stopping at open sea being interpreted as abnormal. With the help of logic this can be

represented as follows:

$$Stopped(Vessel) \wedge isIn(Vessel, OpenSea) \Rightarrow Abnormal(Vessel) \quad (5.1)$$

This is nothing more than the representation of a deterministic view of maritime situation awareness in context. However, prior knowledge under uncertainty would also provide some sort of weight or probability to the aforementioned representation as follows:

$$85\% \quad Stopped(Vessel) \wedge isIn(Vessel, OpenSea) \Rightarrow Abnormal(Vessel) \quad (5.2)$$

denoting that 85% of the cases the antecedent implies abnormal behaviour while 15% of the cases does not.

Prior knowledge is not easy to obtain. In the case of human operators it is synonymous to many hours of theoretical training and experience gained after dealing with a significant amount of situations and incidents while in the case of maritime situation awareness systems, prior knowledge is either encoded with the contribution of domain experts or gained by learning from data.

5.3 Data Fusion Models

As was explained in Section 5.2, information from various sources should always be examined in context using prior knowledge so that correct interpretations of situations are being formed. This calls for data fusion approaches which are particularly helpful especially in cases where there is contradictory information with respect to a situation. This section briefly presents two common data fusion models that are implemented within situation awareness applications. For additional data fusion models the interest reader is referred to [154] for a recent review.

5.3.1 The Observe-Orient-Decide-Act (OODA) Loop

The Observe-Orient-Decide-Act (OODA) loop [155], is a decision making loop that was coined by John Richard Boyd, a United States military strategist. Although initially intended for use by air force pilots, the OODA loop crossed those borders and was adopted in disciplines such as business and law as well as organizations whose main task is to provide surveillance and control services [7]. Figure 5.1 illustrates the OODA loop schematically. Let us explain the four stages of the loop within maritime situation awareness. The decision making loop is initiated with the observation stage (*observe*) the purpose of which is to gather up to date information from all available sources in order to obtain knowledge of what is happening. However, knowing what is happening is not the same as having an understating of a situation. As such, the information from the observation stage is fed forward into the orientation stage (*orient*) whose purpose is to perform fusion of the provided information and analyse them in context and by using prior knowledge so that an understanding

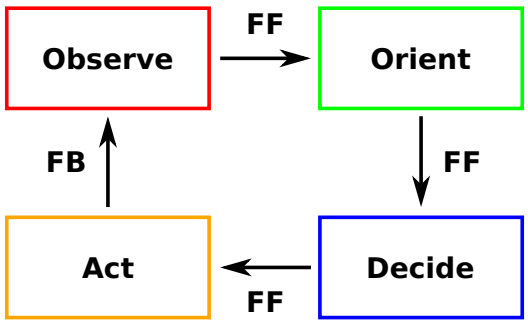


Figure 5.1. The OODA loop. FF corresponds to feed forward flow of information while FB to feedback.

of the unfolding situation at hand is obtained. Next, the outcome of the orientation stage is fed forward into the decision stage (*decide*) whose purpose is to determine a course of action by weighting all available options and picking the most suitable. Actions are then executed during the action stage (*act*) and the outcome is used as feedback to the observation stage along with additional, up to date, information from all available sources. This effectively initiates another OODA cycle and the process continues as described above.

5.3.2 The Joint Directors of Laboratories (JDL) Model

As in the case of the OODA loop, the JDL data fusion model was initially developed for defence applications [156]. Over the years however, modifications have been suggested to the model in the form of revisions aiming at its generalization for applications outside the defence domain. The first revision was the 1999 one [157] while the second, the 2004 one [158]. In this section we present the JDL model from the scope of the latest (2004) revision in which the model comprises four fusion Levels (0-3) as listed below [158]:

Level 0 - Signal/Feature Assessment: Estimation and prediction of signal or feature states that are of interest. "A Level 0 fusion process may include inferences that do not require assumptions about the presence or characteristics of entities ("targets") possessing such observable features. They concern the structure of measurement sets (their syntax) not their cause (their semantics)".

Level 1 - Entity Assessment: At the center of this Level lie the entities that are of interest. More specifically, Level 1 fusion processes are tasked with the "estimation of the identity, classification (in some given taxonomic scheme), attributes, activities, location, and the dynamics and potential states of entities".

Level 2 - Situation Assessment: Given the assessment of entities in Level 1, Level 2 fusion processes involve inferences that are concerned with relations among entities as well the impact of context on both entities and relations. In this manner an assessment of an unfolding situation is built.

Level 3 - Impact Assessment: Given some predefined objectives (e.g. mission goals),

Level 3 fusion processes are concerned with assessing the impact of a previously assessed situation on those objectives.

Level 4 - Performance Assessment: As its name suggests, the fusion Level 4 is concerned with assessing the performance Level 4 fusion processes are concerned with assessing the performance of the data fusion system. This is achieved by combining information to estimate measures of performance as well as measures of effectiveness, abbreviated as MOP and MOE respectively, given a set of desired data fusion system states and/or responses.

5.4 Work on Maritime Situation Awareness Systems

One of the research paths followed in the field is the usage of expert systems for maritime situation awareness. Expert systems initially appeared back in the 1960's and have since then been applied in various fields as decision making support mechanisms as well as situation assessment and awareness support systems [9]. Among others, these fields include the finance, manufacturing and the maritime fields [9, 159]. However, maritime situation awareness with expert systems, is mostly structured around if-then-else rules for providing context, fusing it with sensory information and guiding reasoning. As such, uncertainty arising from sensors or interactions of entities within their environment is not taken into consideration. One representative piece of work following the rule-based, expert systems path is the one found in [160], where the author presents a prototype of such a system offering automated rule-based reasoning for detecting anomalies in support of human operators. In the work presented in [9], the authors focus on the challenges stemming from acquiring knowledge from domain experts in formulating rule-based systems stating that only a small portion of domain knowledge is actually transferred. To remedy this, they observed operators in action and used this as input in the formulation of if-then-else rules. Moreover, the authors created a tool with which operators can modify existing rules and add new ones based on their experience. The results showed that actively engaging operators improves the quality of rules and the level of support that a rule-based maritime situation awareness system can offer to human operators.

Another research path followed in the maritime situation awareness domain is the utilization of well established probabilistic techniques. In [161] the authors utilise both static and dynamic Bayesian Networks (BNs) in identifying anomalous behaviours in vessel movements. In their approach, they learn both the network structure and parameters from AIS data. The results demonstrate that by combining the assessment of the static networks with the assessment of their dynamic counterparts, anomaly detection is enhanced when compared to their individual performance. Bayesian networks are also used in the work presented in [162]. The goal is to determine the probability of high-level threats being present given the likelihood of five vessel anomalous behaviours: i) deviation from standard routes, ii) unexpected AIS activity, iii) unexpected arrival at a port, iv) close approach between vessels (rendezvous) and v) zone entry, e.g. entering a prohibited zone of military

installations or environmental interest. As such, high level threats are treated as a combination of lower level anomalous behaviours. The approach is demonstrated using simulated data. Gaussian processes for maritime anomaly detection are used in [163]. Gaussian processes are actively learned from AIS data, i.e. by subsampling the data to reduce the complexity of training the model. Given the trained model the authors calculate a measure of normality for each newly acquired AIS transmission in order to identify potentially anomalous behaviours of vessels based on their speed as well as longitude and latitude.

Another research path followed in the field is the usage of ontologies in building maritime situation awareness systems. In the work presented in [164] the authors present an approach for detecting abnormal ship behaviour based on a spatial ontology which is associated with a rule-based geographical inference engine. The hierarchy of the ontology is developed with the help of experts while its population is based on a dataset comprising the routes of more than 2000 vessels in the Mediterranean sea. Similarly, rules are defined based on direct expert knowledge (i.e expert interviews). In this manner contextual knowledge is taken into consideration. Similarly, a prototype of a maritime situation awareness system based on ontologies is presented in [165]. More specifically, the developed Maritime Domain Ontology (MDO) as the authors call it, is used in conjunction with description logics inference for detecting behavioural anomalies as well as classifying vessels of interest. Yet another research output in the direction of placing ontologies in the center of a maritime situation awareness system is presented in [166]. More specifically, a maritime situation awareness ontology is utilized for fusing information from various sources including other maritime situation awareness systems while a predefined set of rules is used for identifying abnormal behaviours. One thing that the aforementioned ontology-based approaches share in common is that the usage of ontologies is ideal for encoding context and relations among entities. However, another thing that they share is the lack of mechanisms that will enable them deal with uncertainty. Identifying this limitation, researchers in the field started combining ontologies with mechanisms for dealing with uncertainty.

Gòmez-Romero et al. in [8] present a context-based multilevel information fusion framework for harbour surveillance. In their work ontologies are used both for encoding factual and contextual knowledge. Upon this knowledge the framework initially applies deductive and rule-based reasoning in order to, as the authors state, “extend tracking data and to classify objects according to their features”. Next the framework combines the Belief Argumentation System (BAS) which performs logic-based abductive reasoning [167] with the Transferable Belief Model (TBM) [168] in order for reasoning under uncertainty to be supported, correct assessment of situations to be achieved and the level of the threat to be determined. Integration of Bayesian Networks (BNs) and ontologies for identifying pirate activities in the Gulf of Aden is the approach followed in [169]. More specifically, the Intelligent Maritime Awareness Situation System (IMASS), as the authors call it, is based on: i) an ontology for storing observations and for encoding relations among entities in the environment and ii) a set of logical rules in the form of logical conjunctions for defining normal or abnormal behaviours. The logical rules however, strangely, are not directly

used in assessing the situation but rather have an activation role, i.e. a deterministic rule will trigger its BN counterpart for providing probabilistic inference instead of deterministic. According to the authors, deterministic rules are easier to be understood by non experts as opposed to BNs, justifying in this way their existence. Moving on, Carvalho et al. in [170] present an approach for modelling probabilistic ontologies for supporting maritime situation awareness. The modelling approach is based on the Uncertainty Modelling Process for the Semantic Web (UMP-SW) [171] while probabilistic ontologies are encoded in PR-OWL [172], an upper OWL ontology used for representing uncertainty. Laskey et al. in [173] present a case study of deploying such an ontology in support of a human operator in identifying vessels demonstrating suspicious behaviour. Reasoning on the probabilistic ontology is achieved through Multi-Entity Bayesian Networks (MEBNs), hybrid networks that combine logic with probability theory (see Section 2.3.1). The results demonstrate the effectiveness of the approach. MEBNs fall under the SRL scientific domain and are one of the two SRL approaches known to us being used in the maritime situation awareness field, the other being the work presented in [5] which is based on Markov logic networks. MLNs are used for fusing sensor information with context and providing reasoning under uncertainty for assessing the situation in two different scenarios. The first is concerned with the identification of rendezvous of suspicious vessels while the second (is concerned) with cargo vessels approaching a port, carrying materials who may be hazardous when combined. The proposed MLN-based system covers the first three fusion Levels (levels 0-2) of the JDL model presented in Section 5.3.2. Both the structure and the weights of the MLNs are determined by domain experts. The results demonstrate the impact of context in assessing a situation correctly and the overall advantages of combining first-order logic with probabilistic graphical models in a unified model, i.e. an MLN, in the context of enhancing maritime situation awareness.

5.5 Summary and Discussion

In this chapter we presented common sources of information used in building maritime situation awareness. In addition, we analysed the role of context, uncertainty, and prior knowledge and the importance of data fusion in building maritime situation awareness correctly. Regarding data fusion, we presented the OODA loop and the JDL model which are commonly used data fusion models in maritime situation awareness. Moreover, we presented research outputs concerning maritime situation awareness systems whose purpose is to support human operators.

Regarding approaches based on expert systems, even though they can simplify the highly demanding task of building maritime situation awareness to some extent, they are incapable of capturing and reasoning about the uncertainty arising from the domain. As such, they can be used when deterministic and simplification assumptions are made. On the other hand, traditional probabilistic approaches such as BNs offer a structured representation for uncertainty upon which we can apply probabilistic reasoning for assessing a

situation. Regarding maritime situation awareness systems based purely on ontologies, they are very efficient in capturing relational knowledge in the domain of application and consequently for encoding context. Unfortunately, as in the case of expert systems, they lack mechanisms for addressing uncertainty. However, when combined with such mechanisms they constitute a powerful tool in providing enhanced maritime situation awareness. The combination, however can be either in the form of the two retaining their independence and combining their outcomes as in the case of [8, 169] or in the form of truly unifying them (logic and probability) in a single representation as in the case of MEBNs and MLNs (see [170, 173] and [5] respectively). Only in this manner can we achieve the best of both worlds, i.e. logic and probability theory (see Section 2.4).

Chapter 6

General Thesis Background

This rather brief chapter is concerned with general thesis background by presenting common hardware components found on AUVs (see Section 6.1) as well as the Robot Operating System (see Section 6.2). The content of the chapter is rather useful in understanding concepts presented in Chapters 7, 8 where we present our work with respect to persistently autonomous underwater operations (Chapter 7) and application scenarios within the MCM context (Chapter 8).

6.1 Common Autonomous Underwater Vehicle (Hardware) Components

Depending on the task assigned to an AUV its hardware components may vary. This section provides a list of common hardware components found on AUVs which are the following:

- **Sonar:** Sonar is short for SOund Navigation And Ranging. There are two types of sonar devices (sensors): i) passive sonar and ii) active sonar. A passive sonar, as its name suggests is passively listening to the sounds created by objects in the water whilst and active sonar creates sound pulses which are emitted in the water in order to actively detect reflections and consequently the objects that created them. AUVs commonly use active sonar devices mounted on various angles (e.g forward sonar, downward sonar etc.)

- **Camera & Range Camera:** A camera produces a continuous stream of images while the range camera does the same but with range images. That is, it produces images that show the distance to specific points in a scene from another specific point (depth images). Cameras and range cameras can be placed in any desirable angle (e.g. forward (range) camera, downward (range) camera.

- **GPS:** A GPS (Global Positioning System) device is a device that provides position information expressed in the geographic latitude and longitude. In order to achieve this GPS, devices communicate with satellites in orbit. GPS devices are commonly

used by AUVs in order to get a position fix and/or track movement whilst on the surface or very close to it since satellite signals cannot penetrate water but for usually less than a meter.

- **Compass:** A compass shows the orientation of the vehicle with respect to the direction of north, east south and west.
- **Gyroscope:** A gyroscope, also known as gyro, is a device that measures angular velocity. Gyroscopes are commonly used for building Inertial Navigation Systems (INS).
- **Doppler Velocity Log:** A Doppler Velocity Log (DVL) is a device that provides an estimate of the linear speed at which an AUV is moving.
- **Pressure Sensor:** A pressure sensor yields a pressure measurement that represents the pressure a vehicle receives when submerged.
- **Temperature Sensor:** Temperature sensors are commonly used by AUVs mainly for two different purposes. When placed inside an AUV body, temperature sensors can provide early warnings for components overheating while when placed outside they provide temperature readings of the external environment, i.e. the water.
- **Thruster(s) & Battery(ies):** Typically AUVs are equipped with a set of thrusters that enable them move underwater or on the water surface. Batteries on the other hand are the power source of thrusters as well as any other component on the AUV.

6.2 Robot Operating System

The Robot Operating System (ROS) [174] is an open-source operating system for robots. ROS offers a variety of services commonly found in operating systems in general. These include but are not limited to communication (message passing) between processes, low-level device control and package management [175]. Since its introduction, a high volume of libraries and tools have been written for it something that established ROS as one of the main solutions for developing robot applications and reusing code.

At runtime, ROS is realised as a network of processes in the form of nodes. Nodes can communicate with each other: i) over topics, ii) via services and iii) with a combination thereof (see Figure 6.1). More specifically, topics are communication buses over which nodes can exchange messages. A message is a data structure that consists of typed fields whose data type can be either primitive, e.g. float, or an array containing primitive types, e.g. an array of strings. When nodes are communicating with each other over topics they do so asynchronously with anonymous publish-subscribe semantics. That is, information generation is decoupled from information consumption. Data generating nodes publish to the relevant topic while data consuming nodes subscribe to the relevant topic. In contrast,

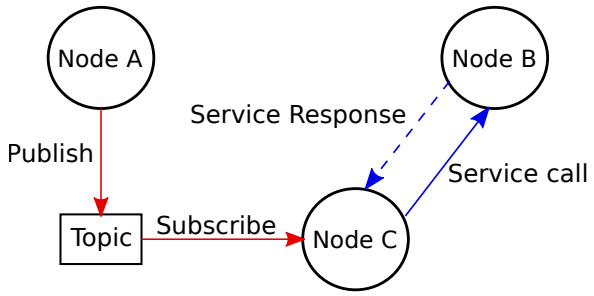


Figure 6.1. ROS nodes communicating at runtime over topics and via services.

a service offers synchronous communication between nodes using a pair of messages: one for requesting data and one for receiving a response. The node offering a service is called a server while the node that sends the request message to the server awaiting for a response is called a client.

Finally, software in ROS is organised in packages. A ROS package can contain ROS nodes, libraries, datasets or anything else useful for an application. ROS packages can also grouped together into ROS stacks.

6.2.1 The Actionlib Package

The actionlib package provides the tools with which accommodate the creation of action servers for executing actions which can be monitored and whose execution can be cancelled by the user through action clients. Every action is specified in .action file which contains an action goal, feedback and result definitions while the main body of the action resides in the node hosting the action server. An action server communicates with an action client over topics by exchanging messages. Figure 6.2 illustrates the communication schematically. Goal messages are published by the action client over a goal topic and are received by the action server who subscribes to the topic. As the action executes the action server publishes feedback messages over a feedback topic to which the client subscribes. In this manner the action client is informed about the progress of achieving the goal periodically. A result message is published by the action server over a result topic upon completion of the action. This differs from feedback since the result is sent only once. When dealing with multiple goals, the action server publishes a status message over a status topic informing action clients on the progress of each goal on the server. Finally, a cancel message can be published by an action client over a cancel topic to which an action server subscribes in order to instruct the action server to stop pursuing achieving a specific goal.

6.3 Summary and Discussion

This rather brief chapter was concerned with presenting some useful general thesis background. In doing so, we have presented hardware components found on AUV. The components presented do not represent an exhaustive list of hardware AUV components since

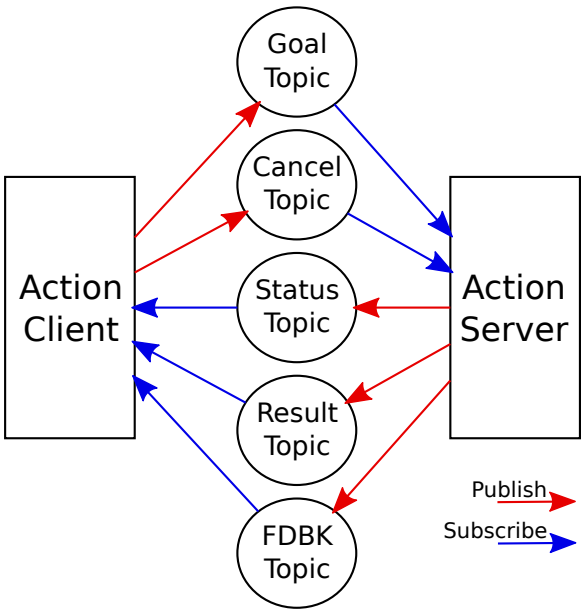


Figure 6.2. Communication between an action client and server.

based on application needs such components may vary. On the contrary, the focus was on presenting common hardware AUV components. That is, components that are most likely to be found on AUVs irrespective of the specialized application needs. In addition we have briefly presented ROS which is one of the most popular platform for developing robotic applications. In doing so, we have analysed the main building blocks of ROS and their functionality. Furthermore, we have presented the actionlib package of ROS which provides the tools for creating action servers for executing actions.

Part III

Design, Implementation, Results

Chapter 7

A Framework for Persistently Autonomous Operations

7.1 Framework Architecture

Carefully identifying the challenges in autonomous underwater operations and the requirements for addressing them is undeniably the key to providing effective solutions. However, this is not enough. The design and implementation of an integrated framework that can deliver those solutions efficiently, is flexible and easy to understand are equally important since these constitute the three major contributing factors towards wider adoption. In that spirit, the framework builds upon the KnowRob system which was developed with household assistant robots in mind but is generic and flexible enough to be adapted for any robotic or application context. Figure 7.1 illustrates the high level architecture of our framework. Existing KnowRob components are illustrated in green while our additions are illustrated in purple. The framework is divided into three main interacting parts: i) knowledge representation, ii) reasoning and iii) planning and execution.

With respect to knowledge representation we adopted a distributed approach following the architectural paradigm of KnowRob. Our KnowRob-based framework utilizes a set of four ontologies for modelling AUVs as well as their state. That is, a components ontology, a capabilities ontology, an actions ontology and a vehicle ontology (see Section 7.2.2). Furthermore, a world ontology models the environment in which the vehicle operates as well as generic concepts (see Section 7.2.1). Finally, the framework utilizes a mission planning and a mission execution ontology that hold all the necessary information concerning mission planning and execution respectively (see Sections 7.3.2, 7.3.3). This distribution of knowledge in a series of newly developed ontologies retains and further promotes the flexibility and modularity offered by KnowRob making it easy for users to replace or modify the aforementioned ontologies should they wish to do so. Moreover, the framework fully exploits KnowRob's capabilities for loading, parsing, storing, accessing and updating knowledge residing within the OWL ontologies using Prolog (see Section 3.3.1). In essence this means that OWL ontologies are internally translated into Prolog terms forming in this

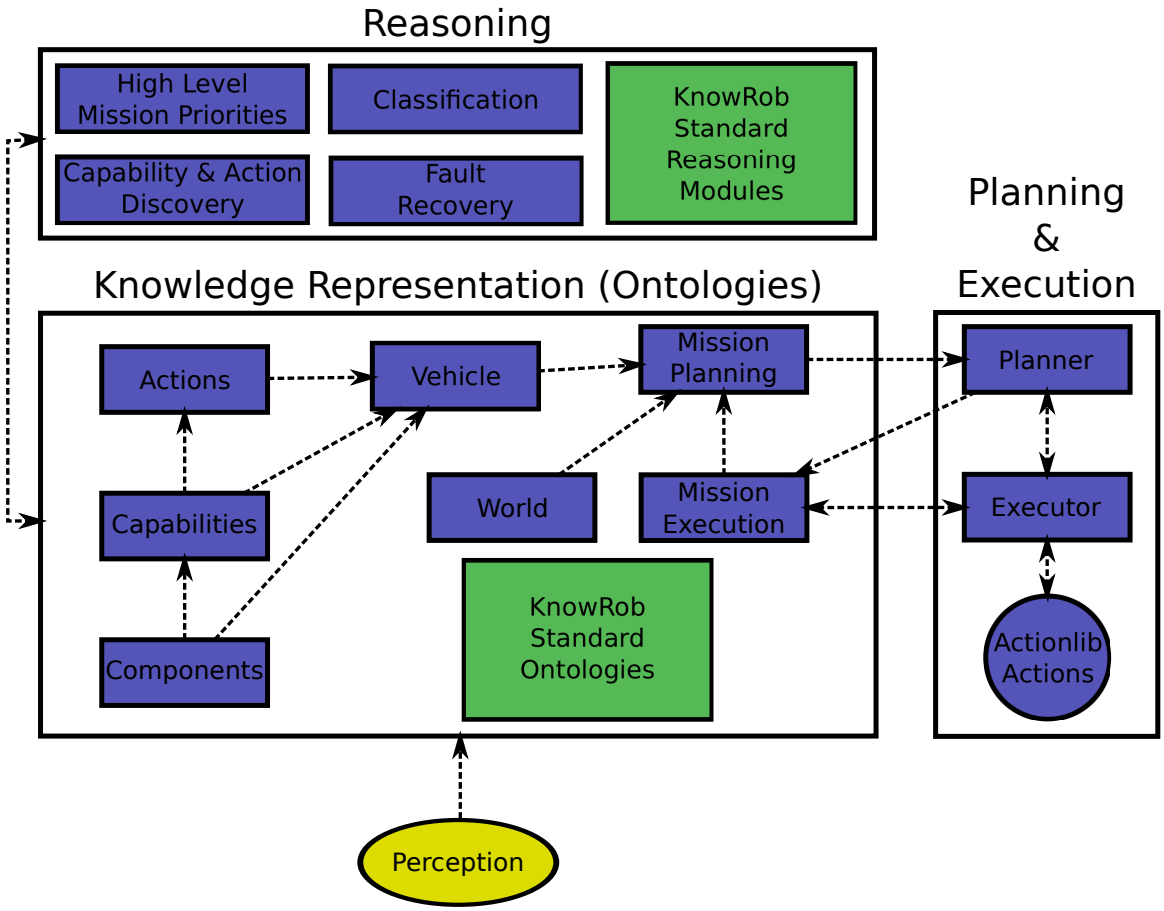


Figure 7.1. High level framework architecture for persistently autonomous operations.

way a Prolog KB and all the aforementioned operations on knowledge are performed by sending Prolog queries via ROS services. Even though the integration with ROS makes the system easily accessible in principal, it was still required that the user was familiar with Prolog syntax to be able to formulate queries. To address this limitation we have developed an additional knowledge interface that masks Prolog syntax by providing Python methods that perform Prolog queries via ROS services behind the scenes.

Regarding reasoning, our framework offers a set of reasoning modules that are implemented in Prolog. These modules interact with the KB (via the knowledge interface) during missions driving real time autonomous decision making and adaptation. These cover the spectrum of vehicle capability, action discovery and fault recovery based on the vehicle's components health status, classification of detections from the perception system as well as different behaviours based on high level mission priority changes communicated to the vehicle. Our reasoning modules constitute adjustments and additions to the existing KnowRob reasoning capabilities so that autonomous underwater operations can be accommodated (see Section 7.2.3).

Finally, the framework is complemented by a mission planning and execution system which interacts with the framework's KB (mission planning and execution ontologies) via the knowledge interface. The architecture of the planning and execution system is based on the work presented in [176] as part of the PANDORA project [151]. The system com-

prises one planner, one executor and ROS actionlib actions. Given information residing in the mission planning ontology the planner generates mission plans which are communicated to the executor and are logged into the mission execution ontology. The generated plans at this stage are only symbolic action representations which are transformed by the executor into actionlib actions and are executed. Execution outcomes are monitored by the executor which updates the execution ontology accordingly. Given the execution status (which is communicated to the planning ontology to update its own status), potential vehicle faults and potentially new high level mission priorities the executor can initiate a new planning-execution cycle if necessary. This occurs until mission objectives are satisfied to the maximum extent possible. This adaptive mission planning and execution approach will be explained in more detail in section 7.3.

7.2 Ontology-Based Knowledge Representation and Reasoning

7.2.1 World Ontology

The world ontology, which we developed from scratch, is the core ontology of our framework and constitutes the basis for modelling the environment in which an AUV will operate performing autonomous operations. It comprises three main classes, i.e *Feature*, *Object* and *Vector* as well as four subclasses. The class hierarchy of the ontology is shown in Figure 7.2.

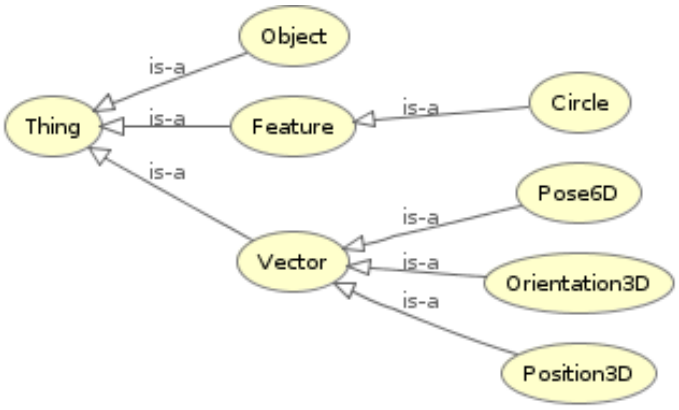


Figure 7.2. World ontology class hierarchy.

The *Feature* class is intended for features that are detected during the search for objects in the environment. As can be observed by looking at Figure 7.2 we have modelled a *Circle* subclass for circular features which are represented as instantiations (instances) of the subclass and are created by a circle feature detection system. Each circle instantiation is related (linked) to its radius through a *radius* float data property. Moreover, each such instance is related to its position in 3D space through a *position* object property that connects it to an

instance of the *Position3D* class. The instance of the *Position3D* class is linked to a 3D position in space via *north*, *east*, *depth* float data properties. This interlinking propagates the information to the Circle class instance. Depending on the nature of autonomous operations and the availability of feature detectors on the vehicle the ontology can be easily extended with additional feature classes as well as object and data properties necessary for representing them. Moving on, we have created an `Object` class which is intended for asserting objects in our KB by creating appropriate instantiations given feature instantiations and an appropriate classification system. Of course, each object instance is associated with a *Position3D* instance via a *position* object property and again, based on application needs, it may be associated with additional properties such as a probability of being that object given of course a probabilistic classifier. Finally, *Vector* subclasses are used for linking "things" to geometrical information. For instance we mentioned that objects are associated with positions in 3D space. Another example of using the vector classes is to express the positions and orientations (poses) of waypoints that the vehicle can use for navigation. The orientation does not refer to a waypoint per se, but rather to the orientation the vehicle should have when visiting this waypoint. How waypoints are generated and the different types of waypoints will be presented in Sections 7.3.1 and 7.3.2.

7.2.2 Components, Capabilities, Actions and Vehicle Ontologies

In Section 7.2.1 we presented the world (core) ontology of our framework. In this section we switch our focus to the modelling of AUVs through their components, capabilities and actions.

Every AUV consists of components which determine its capabilities. Based on those capabilities and the task at hand the AUV is able to execute appropriate actions. A simplification assumption that can be made is that the AUV components will be functional throughout the vehicle's deployment meaning that the mere knowledge of their existence as part of the AUV is sufficient to determine its capabilities and consequently its actions. However in reality, component faults are quite common during missions. This results in vehicles having some of their capabilities either reduced or absent affecting in this manner their set of available actions and putting missions at risk. Consequently, the vehicle must have the ability to reassess its state in response to a dynamic internal environment. With this in mind we have enriched the framework with a set of ontologies that are used for modelling the AUV and its internal state. We followed an incremental, bottom-up modelling approach in which AUV related knowledge is divided into four ontologies: i) the components ontology, ii) the capabilities ontology, iii) the actions ontology and iv) the vehicle ontology. At the very bottom lies the components ontology which models the AUV components as well as their health status. One modelling level above lies the capabilities ontology which models AUV capabilities by importing the components ontology and encoding dependency relations between capabilities and components. At the third modelling level resides the actions ontology that models AUV actions by importing the capabilities

ontology and encoding dependency relations between actions and capabilities. Finally, at the very top (fourth) level lies the vehicle ontology. It imports the actions ontology and by extension the components and capabilities ontology and is capable of modelling different classes of AUVs. This is achieved by encoding what set of components, capabilities and actions each AUV class should have. This bottom-up modelling approach follows the work presented in KnowRob [2] as well as in [177], where the authors utilize ontologies in the same manner in order to formulate a Semantic Robot Description Language (SRDL) that is capable of describing robot components, capabilities and actions. However, our approach differs in that it encodes the health status of vehicle components in their representation as opposed to [2, 177] where this feature is not present. This provides the ability of re-assessing the internal state of the vehicle dynamically and in real time in the presence of component faults something that is also not present in [2, 177]. This in succession enables the framework to adapt its mission planning and execution loop if necessary by feeding relevant information to the planning and execution system which is described in Section 7.3. Figure 7.3 illustrates the four-level bottom-up vehicle modelling approach schematically. Let us now present the components, capabilities, actions and vehicle ontologies in more

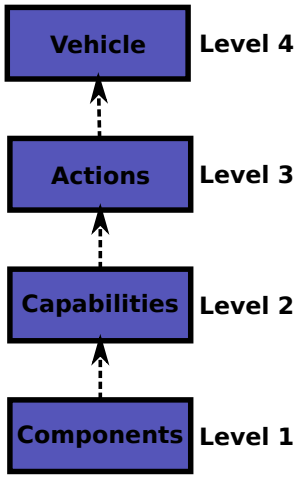


Figure 7.3. Four-level bottom-up vehicle modelling approach. The purple boxes represent ontologies while the arrows indicate the direction of incremental modelling starting from the components ontology up to the vehicle ontology.

detail starting with the components ontology.

7.2.2.1 The Components Ontology

The components ontology models both hardware and software components commonly found in AUVs. Regarding hardware components the ontology comprises: i) sensor classes such as a *DVL* (Doppler Velocity Log) class and a *Gyro* (gyrosscope) class for modelling sensors, ii) a *Thruster* class for modelling thrusters and iii) a *Battery* class for modelling batteries that power the vehicle. With respect to modelling software components, the ontology contains a *NavigationModule* class, a *DetectionModule* class, a *ClassificationModule* class, a *ReacquisitionModule* class, a *PlanningModule* class and an *ExecutionModule* class.

The class hierarchy of the components ontology is shown in Figure 7.4. In order to be able

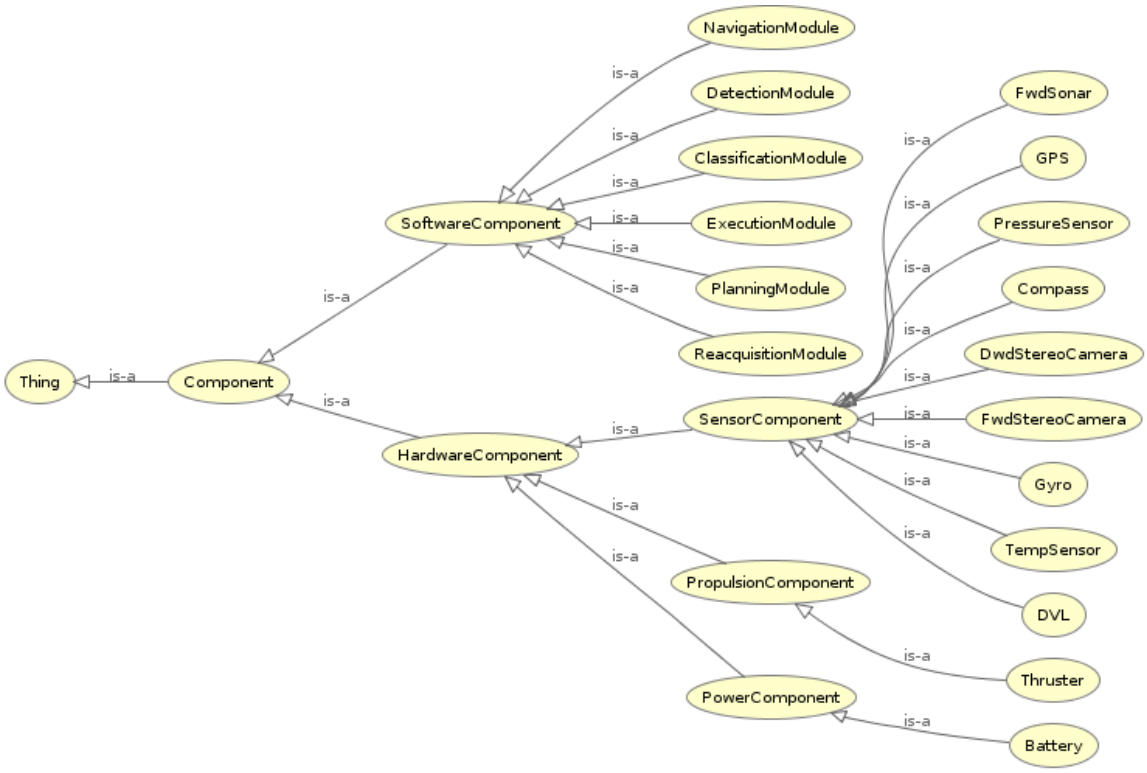


Figure 7.4. Components ontology class hierarchy.

to model the health status of each component we use a binary (true/false) data property named *functionalComponent*. Now, the existence of components in an actual AUV is represented by instantiations of the classes that model those components and each instance is linked to its health status via the *functionalComponent* data property as appropriate. However, our framework does not assess the health status of components. We rather assume that the vehicle is equipped with a fault detection system that provides such information and our framework asserts it in its KB.

7.2.2.2 The Capabilities Ontology

A capability implies both the capacity and the ability to perform a set of actions that are related to that capability. For now, let us focus on the intersection of the capacity and the ability and forget about the set of actions. A human being for example, has the capacity to walk if he has legs but might not have the ability to walk because he might not know how (an infant) or if for example his legs are numb. In these cases we can say that he is not capable of walking. Similarly, an AUV has the capacity to navigate if it is equipped with thrusters, a DVL, a gyroscope and a navigation module. It might not be capable of navigating though because it does not have the ability to do so, that is, one or more of the aforementioned components are faulty. Alternatively it might not be capable of navigating because it does not have the capacity to do so in the first place, i.e. it is missing the necessary components.

Using what we have just described as our logical basis we have created a capabilities ontology that imports the components ontology[1] and encodes dependency relations between capabilities and components. These dependency relations are modelled through *dependsOnComponent* object properties that relate capability classes to component classes. Dependency relations on the class level represent the set of required components for a capability to exist (the capacity). In addition, given that each component is linked to its health status, enables us to check whether the ability is also present for that capability to be instantiated. For example, consider the capability class *SomeCapability* that depends on component classes *SomeComponent* and *SomeOtherComponent*. The *SomeCapability* class will only be instantiated if there are instances of the aforementioned component classes whose health status is functional. Figure 7.5 illustrates the class hierarchy of the capabilities ontology. As can be seen by observing Figure 7.5 there are seven main capability classes, i.e.

Figure 7.5. Capabilities ontology class hierarchy. We have intentionally omitted the class hierarchy of the components ontology (see Figure 7.4) for better readability.

MissionPlanningCapability, *MissionExecutionCapability*, *ClassificationCapability*, *NavigationCapability*, *DetectionCapability*, *ReacquisitionCapability* and *InspectionCapability*. The first three classes depend only on software components. That is, the mission planning capability is instantiated if the AUV is equipped with a functional planning module (a planner). Furthermore, the mission execution capability is instantiated if the AUV is equipped with a functional execution module (an executor) while the classification capability is instantiated if the AUV is equipped with a functional classification module (a classifier).

There are cases though in which there is component redundancy for an AUV capability. This means that a component can be substituted for another component that will allow the

[1] By importing an ontology, say *ontologyA*, into another, say *ontologyB*, we make all the classes, instances and properties of *ontologyA* available to *ontologyB*.

vehicle to maintain that capability in case there is a problem with the initial component, i.e. if it becomes faulty. For this purpose we have introduced the notion of primary, secondary, tertiary ... n-ary capabilities depending on the number of existing redundancies modelled. We call such capabilities *semantically equivalent* because they represent alternative ways of achieving the same type of capability. Characteristic examples of semantically equivalent capabilities are the primary and secondary capabilities that extend their respective capability classes as shown in Figure 7.5. Table 7.1 illustrates the dependency relations between capabilities and components as well as redundancies where applicable.

Capability Class	**Object Property**	**Component Class**
MissionPlanningCapability	dependsOnComponent	PlanningModule
MissionExecutionCapability	dependsOnComponent	ExecutionModule
ClassificationCapability	dependsOnComponent	ClassificationModule
NavigationCapabilityPrimary	dependsOnComponent	Gyro DVL Thruster NavigationModule
NavigationCapabilitySecondary	dependsOnComponent	Compass DVL Thruster NavigationModule
DetectionCapabilityPrimary	dependsOnComponent	FwdSonar DetectionModule
DetectionCapabilitySecondary	dependsOnComponent	FwdStereoCamera DetectionModule
ReacquisitionCapabilityPrimary	dependsOnComponent	FwdSonar ReacquisitionModule
ReacquisitionCapabilitySecondary	dependsOnComponent	FwdStereoCamera ReacquisitionModule
InspectionCapabilityPrimary	dependsOnComponent	FwdStereoCamera
InspectionCapabilitySecondary	dependsOnComponent	FwdSonar

Table 7.1. Dependency relations between capabilities and components. Component classes in the same color (excluding the default black) represent redundancies for capability classes of the same type.

7.2.2.3 The Actions Ontology

In the previous section (Section 7.2.2.2) where we presented the capabilities ontology of our framework we focused on the intersection of the capacity and the ability that equips a vehicle with capabilities to perform a set of actions. In this section we focus on the actions themselves.

For the purpose of modelling actions we have created an actions ontology that imports the capabilities ontology and by extension the components ontology. Actions are modelled as classes and similarly to the case of the capabilities ontology that encodes dependency relations between capabilities and components the actions ontology encodes dependency

relations between actions and capabilities. This is achieved through a *dependsOnCapability* object property that relates them to each other. Dependency relations serve the purpose of defining which capabilities are required for each action to be available to the vehicle. In addition, similar to the case of the capabilities ontology, we consider redundancy and semantic equivalence but in the context of actions. More specifically, the existence of a primary, secondary, ... n-ary capabilities of some type denote redundancy for that capability type. Given the redundancy for capabilities upon which an action depends we have primary, secondary ... n-ary actions which are considered semantically equivalent. Figure 7.6 illustrates the class hierarchy of the actions modelled inside the ontology. We have modelled

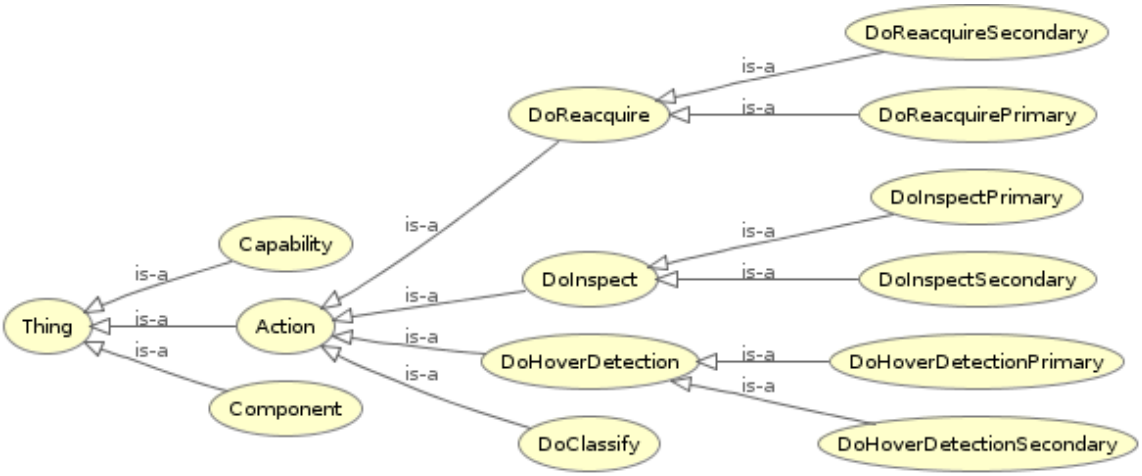


Figure 7.6. Actions ontology class hierarchy. We have intentionally omitted the class hierarchy of the components and capabilities ontologies (see Figures 7.4 and 7.5 respectively) for better readability.

four main action classes in total: *DoHoverDetection*, *DoClassify*, *DoReacquire* and *DoInspect*. The subclasses of the DoHoverDetection action class are used for modelling actions that enable the vehicle to survey areas performing detections of targets (e.g. objects) while the DoClassify action class is used for modelling actions that classify targets. Moreover, the subclasses of the DoReacquire action class are intended for vehicle actions that reacquire targets while the subclasses of the DoInspect action class are intended for modelling target inspection actions. Table 7.2 is indicative of action dependency relations as well as capability redundancies where applicable.

7.2.2.4 The Vehicle Ontology

So far we have presented the means for representing and expressing knowledge about AUV components, capabilities and actions in sections 7.2.2.1, 7.2.2.2 and 7.2.2.3 respectively. The last remaining piece in the vehicle modelling puzzle is the ability to model AUVs. This is exactly the purpose of the vehicle ontology.

More specifically, the vehicle ontology imports the actions and by extension the capabilities and components ontologies and is additionally complemented with a simple *Vehicle*

Action Class	Object Property	Capability Class
DoHoverDetectionPrimary	dependsOnCapability	DetectionCapabilityPrimary NavigationCapabilityPrimary
DoHoverDetectionSecondary	dependsOnCapability	DetectionCapabilitySecondary NavigationCapabilityPrimary
DoClassify	dependsOnCapability	Classificationcapability
DoReacquirePrimary	dependsOnCapability	ReacquisitionCapabilityPrimary NavigationCapabilityPrimary
DoReacquireSecondary	dependsOnCapability	ReacquisitionCapabilitySecondary NavigationCapabilityPrimary
DoInspectPrimary	dependsOnCapability	InspectionCapabilityPrimary NavigationCapabilityPrimary
DoInspectSecondary	dependsOnCapability	InspectionCapabilitySecondary NavigationCapabilityPrimary

Table 7.2. Dependency relations between actions and capabilities. Capability classes in the same color (excluding the default black) represent redundancies for action classes of the same type.

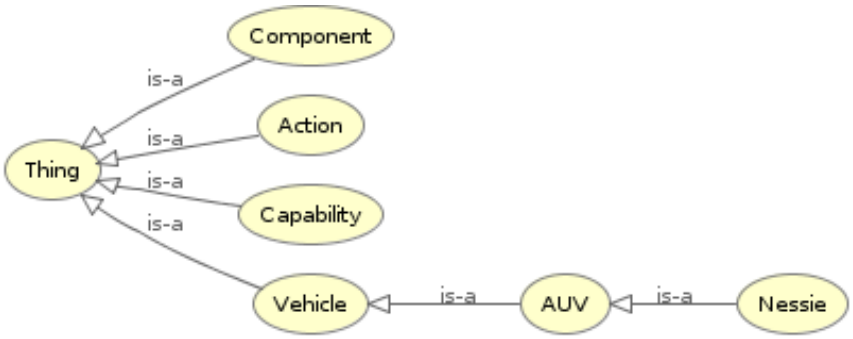


Figure 7.7. Vehicle ontology class hierarchy. We have intentionally omitted the class hierarchy of the components, capabilities and actions ontologies (see Figures 7.4, 7.5 and 7.6 respectively) for better readability.

class hierarchy as illustrated in Figure 7.7. As can be seen by observing Figure 7.7 we have extended the *Vehicle* class with a *Nessie* class since Nessie is the AUV that we have chosen to model inside the vehicle ontology. Given an AUV class, in our case Nessie, we manually encode relations of that class to a set of component, capability and action classes through *hasComponent*, *hasCapability* and *hasAction* object properties respectively. These relations on the class level denote what that AUV class is in principal equipped with. We say in principal because there may be a discrepancy between what is expected to be present or available (in principal) in an AUV in terms of components, capabilities and actions and what is actually the reality due to potentially missing or faulty components. In order to represent an actual AUV that we want to use for underwater operations we manually create an instance of its respective class (e.g. *Nessie1*) as well as instantiations of all of its existing components (e.g. *DVL1*, *FwdSonar1*, *Compass1* etc.) which we initially assert as functional[2]. It is then up to the reasoning process to construct the full vehicle picture by: i) link-

[2]Despite this manual initial assertion, the fault detection system can update the health status of each

ing component instances to the vehicle instance through *hasComponent* object properties, ii) instantiating capabilities and actions and linking them to the vehicle instance through *hasCapability* and *hasAction* object properties respectively. That is, given the health status of each component and the dependency relations inherited by the capabilities and actions ontologies. We will explain how this is done in Section 7.2.3. Figure 7.8 illustrates the full set of relations inside the vehicle ontology.

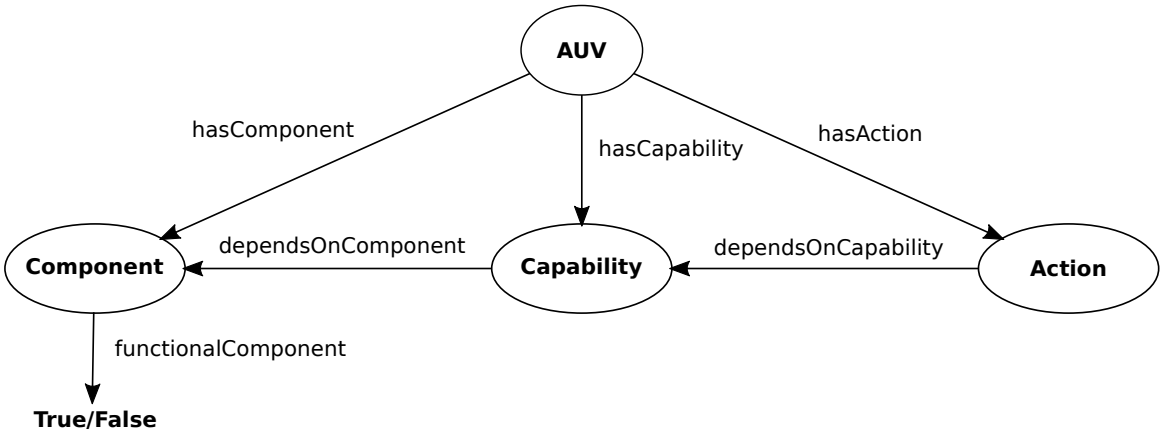


Figure 7.8. Relations between the vehicle, components, capabilities and actions inside the vehicle ontology.

7.2.3 Reasoning About the Vehicle with Prolog

Before going through this section the reader is strongly advised to go through the part of Section 2.1.2.3 that describes inference with Prolog. In addition, he/she is advised to go through Section 3.3.1 that describes the role of Prolog in KnowRob.

As was explained in Section 3.3.1 ontologies in KnowRob are internally represented as Prolog terms that form predicates. As a result, our KnowRob-based framework utilizes Prolog predicates for knowledge related operations (accessing, updating, asserting, deleting) and by extension reasoning. KnowRob offers a full set of query predicates for ontology classes, instances (individuals) as well as properties. We refrain from presenting all of them here but instead we present what we have used as part of our work. The interested reader should refer to [2].

- ***owl_subclass_of(?SubClass, ?SuperClass)*** If the *SubClass* variable is bound the query predicate returns the superclass(es) of *SubClass*, while if *SuperClass* is bound it returns the subclass(es) of *SuperClass*. In case both variables are bound it returns *true* or *false* based on whether the bound *SubClass* is a subclass of the bound *SuperClass*.
- ***owl_direct_subclass_of(?SubClass, ?SuperClass)*** Behaves exactly like the *owl_subclass_of(?SubClass, ?SuperClass)* predicate except for it returns (as its name suggests) only the direct subclass(es) or superclass(es).

component as appropriate.

- ***class_properties(?Class, ?Prop, ?Value)*** Depending on which variables are bound the query predicate returns a combination of classes, properties and property values. For instance, if the *Prop* variable is bound to a property, the predicate returns the class(es) who have (are linked to) that property as well as the property values for every class (if any).

- ***rdfs_individual_of(?Resource, ?Class)*** This query predicate returns *true* if both the *Resource* and *Class* variables are bound, and *Resource* is an individual (instance) of *Class*. That is, *Resource* has an *rdf:type* property that refers to *Class* or any subclass thereof. Moreover, by bounding only the *Resource* variable to an individual the predicate will return the class(es) that individual belongs to while by bounding only the *Class* variable to a class the predicate will return the individual(s) of that class (if any).

- ***rdf_assert(+S, +P, +O)*** This query predicate asserts new triples into the KB. For instance, *rdf_assert(robot:'Nessie1', rdf:type, robot:'Nessie'*) will assert a new individual (instance) *Nessie1* of type (class) *Nessie* into the KB. This is duplicate safe. That is, if the triple that is provided in the *rdf_assert* already exists in the KB the query predicate will have no effect.

- ***rdf_retractall(?S, ?P, ?O)*** This query predicate removes all matching triples from the KB. If the provided triple does not match any triple then *rdf_retractall* has no effect.

- ***owl_has(?S, ?P, ?O)*** This is the most generic and flexible query predicate since it can return any information residing in the KB. For instance, if none of the variables are bound the predicate returns all the information residing in the KB while if *S* is bound to an instance then it returns the property(ies), property value(s) and class(es) of that instance.

Around the query predicates listed above, we have built our own set of query predicates in the form of Prolog rules that we utilize in order to reason about the vehicle and its internal state. This is a continuous and real time time process that operates on part of the KB that is formulated by the components, capabilities, actions and vehicle ontologies described in Section 7.2.2. The choice of Prolog comes naturally since every ontology is internally translated into Prolog terms forming in this way a Prolog KB upon which we can execute queries. The process involves vehicle *system identification* and *system update* phases while the vehicle operates. System identification is tasked with formulating all those instantiations and relations that are required for representing the vehicle and its internal state with respect to the existence of functional components which give birth to capabilities and actions while system update is tasked with deleting instantiations and relations in the presence of non functional components which affect available capabilities and actions. As such, during the operation of the vehicle, system identification and system update are executed in succession and continuously forming effectively a never stopping loop while the vehicle

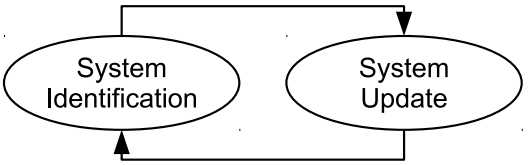


Figure 7.9. System identification and update phases.

operates. Figure 7.9 illustrates the phases schematically. In this manner, the full vehicle picture is represented, monitored and updated in real time.

Recall from Section 7.2.2 that our approach for knowledge representation and modelling of components, capabilities, actions and vehicles demonstrates some similarities to [2], [177]. This means that our approach for reasoning about the aforementioned concepts unavoidably demonstrates similarities too. For instance, we too, utilize Prolog and build on top of existing KnowRob reasoning predicates. At the same time, the fact that in our ontological modelling we have included the encoding the health status of components has transformed the manner in which we reason about the vehicle and its internal state significantly. As opposed to KnowRob, where the mere presence or absence of a robot component would mean that a robot does or does not have a capability or can or cannot execute a task, in our case we go one step further and additionally reason about a components health status not just its mere presence. Moreover, in KnowRob the dynamic assessment and reassessment of a robot's internal state is affected by querying for resources on the worldwide web. For instance, in case that the task for a robot is to make pancakes, the reasoning process will check whether the vehicle has some high level plan to perform the task and if not it will search on the web for a recipe to fill the gaps. If this proves successful, the vehicle will thereafter constantly be in a position to "make" pancakes given the existence of the necessary ingredients and utensils irrespective of whether its manipulator, for example, would break. This of course, leads to inconsistencies. In our case, we deal with such phenomena which are in many cases the norm rather than the exception. Consequently, on top of not having the luxury to query for information on the web we need to be able to capture the internal state of a vehicle more accurately in the presence of hardware or software faults. Losing a vehicle underwater is always a grave danger and most of the times it is accompanied with high recovery costs if recovery is at all possible. Having said that, let us now present and analyse the vehicle system identification and system update phases in Sections 7.2.3.1 and 7.2.3.2 respectively.

7.2.3.1 Vehicle System Identification Phase

The vehicle system identification phase is divided into three steps that handle components, capabilities and actions respectively. Figure 7.10 illustrates the realization of the system identification steps schematically. The first system identification step compares the set of components that should be present on the vehicle based on its class to what is actually present on the vehicle instance. It also formulates relations between the vehicle instance

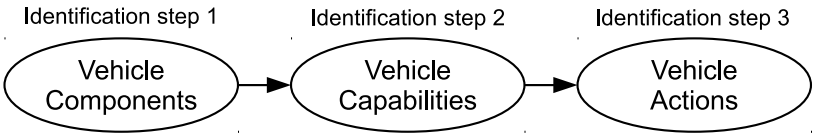


Figure 7.10. System identification steps. Step 1 handles the identification of vehicle components, step 2 handles the identification of vehicle capabilities while step 3 the identification of vehicle actions.

and the components that are found on board the vehicle through *hasComponent* object properties as appropriate. The Prolog rule shown in Snippet 4 demonstrates the aforementioned procedure, expanding the first step of Figure 7.10. The first input variable in the rule head

Snippet 4 Vehicle components system identification.

```
system_identification(RobotInst,RobotClass):-
   rdfs_individual_of(RobotInst,RobotClass),
   owl_direct_subclass_of(RobotClass,robot:'AUV'),
   findall(CompPresent,comp_present_in_KB(CompPresent),L1),
   findall(MissingComp,missing_comp_from_robot(RobotInst,MissingComp),L2),
   member(L1M,L1),
   member(L2M,L2),
   rdfs_individual_of(L1M,L2M),
   rdf_assert(RobotInst,comp:'hasComponent',L1M).
```

(*RobotInst*) is a vehicle instance and the second (*RobotClass*) is the class of the vehicle that the instance belongs to. When querying for this rule, input variables must be bound appropriately. That is, we must provide the instance and the class of the vehicle whose components we are interested in identifying. Notice that in the rule body each predicate is separated by a comma (,) which represents the logical AND ($\wedge$) in Prolog and means that body predicates must be true at the same time for the rule to be true. Inside the body, Prolog checks whether the provided vehicle instance is actually an instance (individual) of the provided vehicle class and whether the class is a subclass of *AUV*. Furthermore, it finds all the component instances in the KB and puts them in a list *L1*. At this point the component instances are not linked to the vehicle instance through *hasComponent* properties. In addition, it finds all the classes of components that should be present on the vehicle and whose instances are not associated with the vehicle instance through *hasComponent* relations and puts them in a list *L2*. Both *comp_present_in_KB* and *missing_comp_from_robot* are assistive rules developed by us and are necessary in the process of searching for all the component instances present in the KB and missing components from the robot respectively. Resuming our analysis, Prolog takes all the members (elements) of *L1* that are instances of *L2* members and asserts into the KB that the given robot (vehicle) instance has as a component each such *L1* member. In this manner the framework identifies the vehicle with respect to its components.

The second system identification step builds on top of the first one and is responsible for identifying the capabilities of the vehicle and formulating relations between the vehicle

instance and capability instances, that will be potentially created, via *hasCapability* object properties. That is given the existence of component instances related to the vehicle that the first identification step yielded, their health status, as well as dependencies among the capabilities that the vehicle should have and those vehicle components. The Prolog rule shown in Snippet 5 demonstrates the aforementioned procedure, expanding the second step of Figure 7.10. Please note that the head of the rule is the same as the one in the first iden-

Snippet 5 Vehicle capabilities system identification.

```
system_identification(RobotInst,RobotClass):-
   findall(Cap,cap_found_on_robot_class(RobotClass,Cap),L1),
   member(L1M,L1),
   findall(CompReq,required_comp_for_capability(L1M,CompReq),L2),
   member(L2M,L2),
   findall(CompMiss,non_functional_comp_for_capability(L1M,RobotInst,CompMiss),L3),
   (L3 == []),
   findall(A,rdfs_individual_of(A,L1M),L),
   (L == [] ->
      (unique_instance_name(L1M,Inst),
      rdfs_individual_of(CompInst,L2M),
      rdf_assert(Inst, 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',L1M),
      rdf_assert(Inst, comp:'dependsOnComponent',CompInst),
      rdf_assert(RobotInst,cap:'hasCapability',Inst));
      (rdfs_individual_of(B,L1M),
      rdfs_individual_of(CompInst,L2M),
      rdf_assert(B, 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',L1M),
      rdf_assert(B, comp:'dependsOnComponent',CompInst),
      rdf_assert(RobotInst,cap:'hasCapability',B))
   ).
```

tification step. What changed is the body of the rule. This is logical since vehicle system identification is one phase divided into three steps. As such, the head of the rule is the same so that we have the same input variables bound to the same values. Regarding the rule body, Prolog finds all the classes of capabilities the vehicle class should have and puts them in a list *L1*. In addition, it chooses an element of *L1* (*L1M*) and finds the classes of components the instances of which are required for instantiating *L1M* and puts them in a list *L2*. Prolog also chooses a element of *L2* (*L2M*). At the same time, Prolog takes the *L1M* member and finds all non functional component (component instances) that affect it, and puts them in a list *L3*. At this point if *L3* is not empty the rule will fail for that *L1M*,*L2M* pair and Prolog will investigate another one. In this manner, capabilities who depend on components that are not functional are not instantiated. In contrast, if *L3* is empty that means that there not any non functional component instances for capability class *L1M* and Prolog will search for instances of that capability inside the KB and put them in a list *L*, if any. Now, if *L* is empty, the framework creates a capability instance *Inst* and asserts it in the KB as well as formulates relations to the healthy component instances that the capability instance depends on via *dependsonComponent*. Moreover, the framework formulates a relation between the robot (vehicle) instance and the capability instance via a *hasCapability* object property. Finally, if *L* is not empty that means that there is a capability instance inside the KB and the framework proceeds with the formulation of the *dependsonComponent* and *hasCapability*

relations. This process is iterative and continues until all eligible *L1M*,*L2M* pairs are investigated. In this manner the framework identifies the vehicle with respect to its capabilities. That is, by instantiating all capabilities who depend on available functional components and linking them to the vehicle. To conclude this second identification step, we would like to mention that similar to the case of the first identification step, cap_found_on_robot_class, required_comp_for_capability, non_functional_comp_for_capability are assistive rules developed by us. They are are necessary in the process of searching for all the capability classes that should be present on the vehicle based on its class, all the required component classes for a capability to be instantiated and all component instances that are not functional affecting one or more of the capabilities respectively.

Finally we have the third system identification step which is responsible for identifying vehicle actions based on available capabilities. The process is very similar to the second identification step upon whose outcomes it builds. Therefore we will not go into great detail analysing it but instead we provide a higher level description of the process which we illustrate in Snippet 6. Snippet 6 represents an expansion of the third identification step depicted in Figure 7.10. During this third step, Prolog compares the actions that should

Snippet 6 Vehicle actions system identification.

```
system_identification(RobotInst,RobotClass):-
   findall(Act,act_found_on_robot_class(RobotClass,Act),L1),
   member(L1M,L1),
   findall(CapReq,required_cap_for_action(L1M,CapReq),L2),
   member(L2M,L2),
   findall(CapMiss,missing_cap_for_action(L1M,RobotInst,CapMiss),L3),
   (L3 == []),
   findall(A,rdfs_individual_of(A,L1M),L),
   (L == [] ->
      (unique_instance_name(L1M,Inst),
      rdfs_individual_of(CapInst,L2M),
      rdf_assert(Inst, 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',L1M),
      rdf_assert(Inst, cap:'dependsOnCapability',CapInst),
      rdf_assert(RobotInst,act:'hasAction',Inst));
      (rdfs_individual_of(B,L1M),
      rdfs_individual_of(CapInst,L2M),
      rdf_assert(B, 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',L1M),
      rdf_assert(B, cap:'dependsOnCapability',CapInst),
      rdf_assert(RobotInst,act:'hasAction',B))
   ).
```

be available on the vehicle based on its class as well as their dependencies on specific capability classes to what is actually available on the vehicle on the capability level, i.e. which capabilities are instantiated and linked to the vehicle instance. In the event that for an action class all capability dependencies are met through the existence of instantiations of their respective classes, Prolog checks whether the action class is already instantiated and if it is it asserts dependency relations between the action instance and the capability instances into the KB via *dependsonCapability* object properties. In addition, it formulates (asserts) a relation between the vehicle instance and the instantiated action class via a *hasAction* object property. Now, if the action class, whose capability dependencies are met, is not

instantiated in the KB then there is an additional step to the process just described that comes first and is concerned with the creation of a action class instance and then form all the aforementioned relations. Finally, in case not all capability dependencies are met for the action class, that action class is not instantiated and consequently relations are not formed. This is an iterative process until all actions that can be instantiated and linked to the vehicle instance as well as to the capability instances have done so. In this manner the framework identifies the available actions that the vehicle can execute. This concludes the system identification phase.

7.2.3.2 Vehicle System Update Phase

In Section 7.2.3.1 we saw how using Prolog the framework is able to build the representation of the vehicle by formulating appropriate instantiations and relations among components, capabilities, actions and the vehicle itself. However, the vehicle system identification phase in itself is not sufficient for achieving a complete system of assessing and reassessing the vehicles internal state. A careful and observant reader may have already identified why. Nowhere within the system identification phase steps is there anything regarding deleting instances or relations. For example, consider the case in which our vehicle has a broken sonar. In that case the first system identification step will associate the component instance with the vehicle instance but the second step will refrain from instantiating the capabilities that are affected by the non functionality of sonar component. Ergo relations among an existing thing (the vehicle instance) and something non existent (the instance(s) of the capability(ies) affected by the sonar) cannot be formed. Similarly, during the third system identification step, actions that require the affected capabilities so that they can be instantiated, will naturally not be instantiated and relations that involve their instances will not be formed. So far things are in order. In addition, if somehow the sonar becomes operational again, e.g. it has a problematic connection to the vehicle that sometimes works, then the system identification steps will immediately handle the new piece of information and update the KB accordingly. What happens though if a component that has been previously functional brakes? System identification will just ignore it and will not form instantiations and relations that are affected by it. However such instances and relations already exist in the KB from the time where the component was functional. That naturally leads to an inconsistency and a gap towards truly achieving a reliable system. That is when the vehicle *system update* phase comes into play. Snippet 7 illustrates its realization in Prolog while Figure 7.11 illustrates the schematic decomposition of the system update phase and can be used as a visual aid to better understand the Prolog snippets that will follow in this section.

As can be seen by observing Snippet 7 system update is encoded as a Prolog rule whose body comprises seven predicates, six of which are system update predicates and one predicate for finding the non functional components in the KB. Again, the input variables of the rule head are the instance of the robot for which we want to perform system update as well as its class. Predicate *non_functional_components_on_robot* is realized as a Prolog

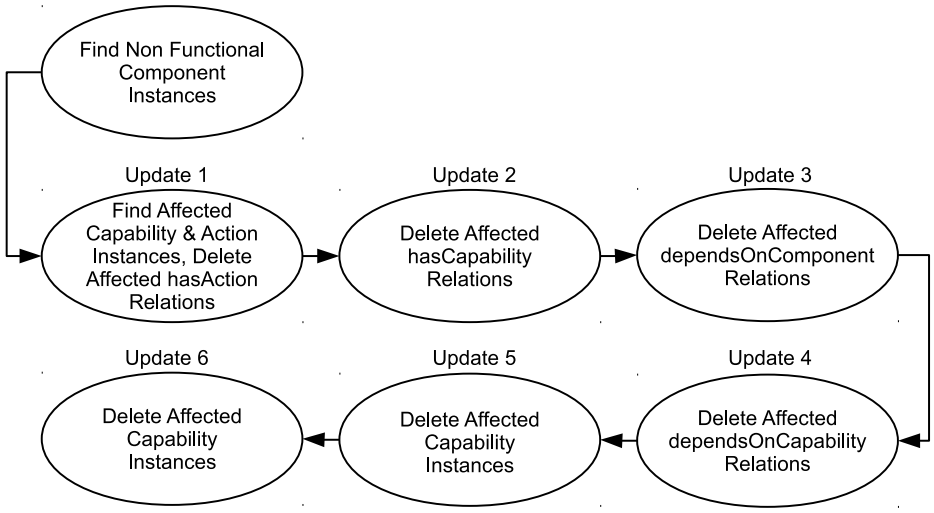


Figure 7.11. Vehicle system update phase decomposition.

Snippet 7 Vehicle system update phase.

```
system_update(RobotInst,RobotClass):-
   non_functional_components_on_robot(RobotInst,BrokenCompList),
   system_update1(RobotInst,RobotClass,BrokenCompList,LAffectedCap,LAffectedAct),
   system_update2(RobotInst,RobotClass,LAffectedCap),
   system_update3(RobotClass,LAffectedCap),
   system_update4(RobotClass,LAffectedAct),
   system_update5(RobotClass,LAffectedCap),
   system_update6(RobotClass,LAffectedAct).
```

rule and is illustrated in Snippet 8. More specifically, the rule simply finds all components

Snippet 8 Non functional components on the robot (vehicle).

```
non_functional_components_on_robot(RobotInst,BrokenCompList):-
   findall(BrokenComp,(owl_has(RobotInst,comp:'hasComponent',BrokenComp),
   owl_has(BrokenComp,comp:'functionalComponent',Value),
   strip_literal_type(Value,false)),BrokenCompList).
```

(component instances) in the KB whose *functionalComponent* object property is false an puts them in the *BrokenCompList* variable which is bounded as a list containing instances of broken (non functional) components.

Then, system update proceeds with the predicate *system_update1* which is also realized as a Prolog rule and is illustrated in Snippet 9. The first two input variables of the rule are given while the third one is previously calculated by the *non_functional_components_on_robot* rule. Input variables *LAffectedCap* and *LAffectedAct* are initially unbound and are intended for storing the list of affected capabilities and actions respectively after the rule execution. Now, as can be observed by looking at Snippet 9, Prolog will check whether there are non functional components in *BrokenCompList* and for each one, if any, it will find all the affected capabilities and put them in the *LAffectedCap* list, if any. In addition, Prolog will check whether *LAffectedCap* contains elements , i.e. whether there are affected capabilities, and for each one, if any, it will find all the affected actions and put them

Snippet 9 Calculation of affected capabilities and actions based on non functional components. In the event that there are affected actions, *hasAction* relations between the vehicle instance and the instances of such actions are deleted.

```
system_update1(RobotInst,RobotClass,BrokenCompList,LAffectedCap,LAffectedAct):-
   (BrokenCompList \= [] ->
      (member(BrokenCompListM,BrokenCompList),
      findall(AffectedCap,owl_has(AffectedCap,_,BrokenCompListM),LAffectedCap));
      true),
   (LAffectedCap \= [] ->
      (member(LAffectedCapM,LAffectedCap),
      (\+ rdfs_individual_of(LAffectedCapM,RobotClass) ->
      findall(AffectedAct,owl_has(AffectedAct,_,LAffectedCapM),LAffectedAct));
      true); true),
   (LAffectedAct \= [] ->
      (member(LAffectedActM,LAffectedAct),
      (\+ rdfs_individual_of(LAffectedActM,RobotClass) ->
      rdf_retractall(RobotInst,act:'hasAction',LAffectedActM));
      true); true).
```

in the *LAffectedAct* list, if any. Finally, Prolog will check whether *LAffectedAct* contains elements , i.e. whether there are affected actions, and for each one, if any, it will delete all *hasAction* relations between the vehicle instance and the affected action instances, if any. This effectively represents that the vehicle is no longer capable of executing the affected action(s). This is an iterative process until all affected capabilities and actions are identified and all *hasAction* relations of those actions affected are deleted, if any.

System update then proceeds with the predicate *system_update2* which is illustrated in Snippet 10. Given the instance of the vehicle, its class, as well as the list of affected

Snippet 10 Deletion of *hasCapability* relations between the vehicle instance and instances of affected capabilities, if any.

```
system_update2(RobotInst,RobotClass,LAffectedCap):-
   (LAffectedCap \= [] ->
      (member(LAffectedCapM,LAffectedCap),
      (\+ rdfs_individual_of(LAffectedCapM,RobotClass) ->
      rdf_retractall(RobotInst,cap:'hasCapability',LAffectedCapM));
      true); true).
```

capabilities (capability instances) from *system_update1*, Prolog deletes all relations between the vehicle instance and the instances of the affected capabilities, if any, in an identical manner to the deletion of *hasAction* relations shown in Snippet *system_update1*. With this, all relations among the vehicle, the affected capabilities and actions have now been deleted.

Next in line for deletion are the relations between affected capability instances and non functional component instances as well as relations between affected action instances and affected capability instances, if any. These are no others than relations formed via *dependsOnComponent* and *dependsOnCapability* object properties as shown in Snippet 11.

Finally, what remains is the deletion of capability and action instances from the KB in the event that there are non functional components that affect them. Snippet 12 illustrates

Snippet 11 Deletion of *dependsOnComponent* and *dependsOnCapability* relations involving non functional component, affected capability and action instances, if any.

```
system_update3(RobotClass,LAffectedCap):-
   (LAffectedCap \= [] ->
      (member(LAffectedCapM,LAffectedCap),
      (\+ rdfs_individual_of(LAffectedCapM,RobotClass) ->
      rdf_retractall(LAffectedCapM,comp:'dependsOnComponent',_));
      true); true).
system_update4(RobotClass,LAffectedAct):-
   (LAffectedAct \= [] ->
      (member(LAffectedActM,LAffectedAct),
      (\+ rdfs_individual_of(LAffectedActM,RobotClass) ->
      rdf_retractall(LAffectedActM,cap:'dependsOnCapability',_));
      true); true).
```

this process. This concludes the vehicle system update phase.

Snippet 12 Deletion of capability and action instances affected by non functional component(s), if any.

```
system_update5(RobotClass,LAffectedCap):-
   (LAffectedCap \= [] ->
      (member(LAffectedCapM,LAffectedCap),
      (\+ rdfs_individual_of(LAffectedCapM,RobotClass) ->
      (owl_has(LAffectedCapM,'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',A),
      rdf_retractall(LAffectedCapM,
      'http://www.w3.org/1999/02/22-rdf-syntax-ns#type,A)));
      true); true).
system_update6(RobotClass,LAffectedAct):-
   (LAffectedAct \= [] ->
      (member(LAffectedActM,LAffectedAct),
      (\+ rdfs_individual_of(LAffectedActM,RobotClass) ->
      (owl_has(LAffectedActM,'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',A),
      rdf_retractall(LAffectedActM,
      'http://www.w3.org/1999/02/22-rdf-syntax-ns#type,A)));
      true); true).
```

7.3 Adaptive Mission Planning and Execution

According to the Oxford English dictionary, *planning* is the act of deciding how to do something. As such, planning is a fundamental activity of human beings and is used daily as a means of achieving one's goals. From simple and short term goals (e.g. how to go to work) to more complex and long term ones (e.g. how to save money to buy a house), planning is, in its essence, indicative of intelligence. By acting in the real world, humans may have to deal with situations that are *new* to them and/or with consequences of their actions that were *unexpected*. In addition, the real world tends to be *dynamic* and *uncertain* and new information, which was not available at the time of planning, may be subsequently available to them. In such cases, adapting or refining the initial course of actions (plan) to accommodate the "unexpected", the "new", and the "uncertain" comes naturally to most people.

Regarding autonomous vehicles, and by extension AUVs intended for real world applications, mission planning[3] can be a very resource demanding and time consuming process that is affected by the size and the complexity of the domain and the problem that needs solving. Similarly to planning in the case of humans, the challenges that an autonomous vehicle has to face are domains that are dynamic, partially known and uncertain. Unexpected changes in the domain and/or the state of the vehicle (e.g hardware faults) can jeopardize a mission. In addition, changes in mission objectives imposed either by humans or resources availability or the external environment should be treated in a meaningful and effective manner. As such, both hardware faults and changes in high level mission priorities during mission execution require adaptive mission planning and execution approaches. These approaches firstly enable robustness under a variety of unexpected events, and secondly ensure that mission goals are satisfied to the maximum extent possible. Adaptive mission planning and execution approaches are the key to robust mission planning and execution which in turn is one of the key requirements towards *persistently autonomous vehicles*. That is, autonomous vehicles that minimize requests for assistance from an operator when they get stuck through a lack of cognitive ability to deal with a situation.

We have mentioned earlier in this chapter that the architecture of mission planning and execution is based on the work of the PANDORA project. As in the case of PANDORA, the system comprises one PDDL planner and one executor. Architecture-wise, as opposed to PANDORA, we divide planning and execution information into two distinct ontologies, i.e. a planning and an execution ontology who exchange information (see Sections 7.3.2, 7.3.3). This distinction makes the system modular and more flexible especially in cases where different planning approaches need to be implemented by other users. Regarding the planning ontology the full PDDL domains and problems reside in the planning ontology and as such they can be reconstructed for the planner. This reconstruction is present in PANDORA and by extension to the standalone planning and execution framework ROSPlan that materialised from the PANDORA project. However, in ROSPlan PDDL domain files are fed into the system as input and then extracted to be used by a planner. In our case PDDL files can be directly generated from the ontology. We had presented the idea of reconstructing PDDL files from a planning ontology back in 2014 and 2015 [178], [4]. In addition, we do not use ROSPlan, we have developed an implementation of our own and we have developed reasoning modules based on Prolog (see Section 7.2.3) that the executor consults while monitoring the execution of actions enabling adaptation. More specifically, in the event of hardware faults our approach favours adaptive execution to adaptive planning, by initially searching for *semantically equivalent* actions to be executed[4]. If none is executable then it resorts to replanning given the current state of both the vehicle and the world in conjunction with remaining mission goals (adaptive planning). Prioritizing adaptive execution over adaptive planning is important as mission planning can be very demanding in terms of computational resources [179]. However, when a request for changing

[3]Mission planning refers to planning missions that involve autonomous underwater operations.

[4]Note that the notion of semantically equivalence is not present neither in PANDORA nor in ROSPlan.

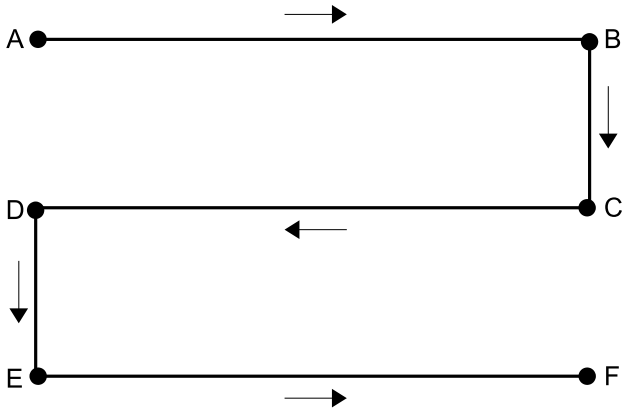


Figure 7.12. Exemplar lawnmower trajectory.

high level mission priorities is issued to the vehicle, the framework performs replanning due to effectively invalidating the prior plan in the vast majority of cases. For mission planning and replanning we use the temporal PDDL planner OPTIC [144].

7.3.1 Generating Mission Plans

In order to generate mission plans, every PDDL planner requires as input a `domain` and a `problem` to solve within the `domain` (see Section 4.5.1). That is, irrespective of whether the domain is a temporal or a non-temporal one. In Section 7.3.1.1 we present a temporal PDDL domain in the context of persistently autonomous operations while in Section 7.3.1.3 we present PDDL problems in the same context. The temporal aspect of the domain is the one presented in the work of Cashmore et. al. [152].

7.3.1.1 Temporal PDDL Domain

While reading this section it is worthwhile referring to Section 4.5.1 where we presented the building blocks of temporal domains under the PDDL2.1 specification as well as various definitions about planning, durative actions etc. Snippet 13 illustrates various `types`, `predicates` and `functions` in PDDL syntax in the domain of detecting, classifying, reacquiring and inspecting physical objects in the external environment.

In our domain we have four `types` of objects: `lawnmowerpoints`, `objectpoints`, `inspectionpoints` and `vehicles`. Lawnmowerpoints are waypoints that a vehicle visits when traversing a lawnmower search trajectory[5] (see Figure 7.12 for an exemplar lawnmower search trajectory) to detect objects in the environment. Objectpoints are waypoints that a vehicle visits and reacquires (physical) objects while inspectionpoints are waypoints around (physical) objects that a vehicle visits and inspects the objects.

[5]There is no formal definition of a lawnmower search trajectory because there are literally infinitely many ways in which someone can maw their lawn. For instance, one can go in straight lines or circles or zigzags or in any way. However lawnmower search is an established straight line search pattern in the underwater domain of the form shown in Figure 7.12.

Regarding the domain predicates shown in Snippet 13, predicate `at_op ?v...` denotes that some vehicle `v` is at objectpoint `op` while `reachable_op ?op1 ?op2...` denotes that some objectpoint `op1` is reachable from some objectpoint `op2`. That is, the vehicle is able transition from objectpoint `op1` to objectpoint `op2`. Vehicle `v`, objectpoints `op`, `op1` and `op2` are variables, also known as parameters which is indicated by the `?` notation preceding them. Effectively this means that they (the variables) can be instantiated to represent different vehicles and objectpoints respectively. The same goes for the remaining predicates but with respect to lawnmowerpoints and inspectionpoints.

Snippet 13 Temporal PDDL domain types, predicates and functions for detecting, classifying, reacquiring and inspecting objects.

```
(:types Lawnmowerpoint Objectpoint Inspectionpoint Vehicle)
(:predicates
        (at_op ?v - Vehicle ?op - Objectpoint)
        (reachable_op ?op1 ?op2 - Objectpoint)
        (at_ip ?v - Vehicle ?ip - Inspectionpoint)
        (reachable_ip ?ip1 ?ip2 - Inspectionpoint)
        (at_lp ?v - Vehicle ?lp - Lawnmowerpoint)
        (reachable_lp ?lp1 ?lp2 - Lawnmowerpoint)
)
(:functions
        (visited_lp ?lp - Lawnmowerpoint)
        (visited_op ?op - Objectpoint)
        (visited_ip ?ip - Inspectionpoint)
        (distance_lp ?lp1 ?lp2 - Lawnmowerpoint)
        (distance_op ?op1 ?op2 - Objectpoint)
        (distance_ip ?ip1 ?ip2 - Inspectionpoint)
        (obj_detection_complete ?lp - Lawnmowerpoint)
        (obj_classification_complete ?v - Vehicle)
        (reacquired_obj ?op - Objectpoint)
        (inspected_obj ?ip - Inspectionpoint)
        (cnt_reacquired_obj ?v - Vehicle)
        (cnt_visited_lp ?v - Vehicle)
        (remaining_energy ?v - Vehicle)
        (consumed_energy ?v - Vehicle)
        (energy_consumption_rate_moving ?v - Vehicle)
        (energy_consumption_rate_still ?v - Vehicle)
        (prob_obj ?op - Objectpoint)
        (ent_obj ?op - Objectpoint)
        (prob_obj_quotient_sum ?v - Vehicle)
        (ent_obj_quotient_sum ?v - Vehicle)
        (det_obj_time ?v - Vehicle)
        (class_obj_time ?v - Vehicle)
        (reacq_obj_time ?v - Vehicle)
        (insp_obj_time ?v - Vehicle)
        (mult_fact_time ?v - Vehicle)
        (mission_duration)
)
```

With respect to the domain functions shown in Snippet 13, functions `visited_lp ?lp...`, `visited_op ?op...` and `visited_ip ?ip...` respectively indicate whether lawnmowerpoint `lp`, objectpoint `op` and inspectionpoint `ip` have been visited. Furthermore, distance functions `distance_lp ?lp1 ?lp2...`, `distance_op ?op1 ?op2...` and `distance_ip ?ip1 ?ip2...` return the distance between lawnmowerpoints `lp1` and `lp2`,

objectpoints op1 and op2 and inspectionpoints ip1 and ip2 respectively. Function obj_detection... indicates whether the object detection phase is complete up to lawnmowerpoint lp whereas function obj_classification... indicates whether the object classification phase has been completed by vehicle v. Function reacquired_obj ?op... indicates whether a detected object in the environment has been reacquired from objectpoint op while function inspected_obj ?ip... indicates whether a reacquired object in the environment has been inspected from inspectionpoint ip. Function cnt_reacquired_obj ?v... is a counter function that returns the number of objects that have been reacquired by vehicle v and function cnt_visited_lp ?v... a counter function that returns the number of lawnmowerpoints that have been visited by vehicle v. Moreover, function remaining_energy ?v... returns the remaining energy of vehicle v while consumed_energy ?v... returns the consumed energy of the vehicle. Naturally, as a vehicle operates it consumes energy (the consumed energy increases and the remaining energy decreases). Function energy_consumption_rate_moving ?v... represents the (per meter) rate at which the vehicle consumes energy whilst moving while function energy_consumption_rate_still ?v... represents the (per second) rate at which the vehicle consumes energy while maintaining its position. How these functions are used will be explained later on in this section where we present durative actions. Now, functions prob_obj ?op... and ent_obj ?op... return the probability of a detection being an object of interest and the entropy related to that detection respectively. Again, we refrain from explaining functions prob_obj_quotient... and ent_obj_quotient... for now as they can be better explained when describing the actions that use them in the remainder of this section. Finally, function det_obj_time ?v... represents the time that the vehicle needs to detect objects, function class_obj_time ?v... represents the time that the vehicle needs to classify objects, function reacq_obj_time ?v... represents the time that the vehicle needs to reacquire an object from some location while function insp_obj_time ?v... represents the time that the vehicle needs to inspect an object from some location. Function mult_fact_time ?v... represents a multiplication factor for time when estimating the duration of actions. Its usage will be explained in the next paragraphs where we analyse durative actions. Finally, mission_duration is a function that returns the time the vehicle spent executing domain actions.

We have encoded four discrete durative actions in our domain, one for each of the four phases, i.e do_hover_detection, do_classify, do_reacquire and do_inspect. Action do_hover_detection is an action that is intended for plans where the goal is to detect objects in the environment and is illustrated in Snippet 14. The parameters of the action are a vehicle v, and two lawnmowerpoints, from and to. The duration of the action is equal to to the distance between the two lawnmowerpoints from,to multiplied by a multiplication factor for time plus the time the vehicle needs to detect objects. The multiplication factor for time in conjunction with the distance between the points effectively allows the action to model the time the vehicle needs for moving from one location to another. For instance, if the vehicle moves with 3 m/s the multiplication factor would be 0.333 and as such if

Snippet 14 Temporal PDDL domain do_hover_detection durative action.

```
(:durative-action do_hover_detection
 :parameters (?v - Vehicle ?from ?to - Lawnmowerpoint)
 :duration (= ?duration (+(*(distance_lp ?from ?to)
         (mult_fact_time ?v)) (det_obj_time ?v)))
 :condition (
                 and (at start (at_lp ?v ?from))
                 (at start (reachable_lp ?from ?to))
                 (at start (= (visited_lp ?to) 0))
                 (at start (= (obj_detection_complete ?to) 0))
                 (at start (> (remaining_energy ?v) (* (distance_lp ?from ?to)
                 (energy_consumption_rate_moving ?v))))
               )
 :effect (
           and (at start (not (at_lp ?v ?from)))
           (at end (at_lp ?v ?to))
           (at end (decrease (remaining_energy ?v) (* (distance_lp ?from ?to)
           (energy_consumption_rate_moving ?v))))
           (at end (increase (consumed_energy ?v) (* (distance_lp ?from ?to)
           (energy_consumption_rate_moving ?v))))
           (at end (increase (visited_lp ?to) 1))
           (at end (increase (obj_detection_complete ?to) 1))
           (at end (increase (cnt_visited_lp ?v) 1))
           (at end (increase (mission_duration) ?duration))
         )
)
```

the distance between locations `from,to` is, say, 10 meters then the vehicle will need 3.33 seconds to traverse it. However, the action is not only about moving from a location to another its also about detecting objects the duration of which we model with `det_obj_time...` which is added to the time the vehicle needs to traverse the distance between the lawnmowerpoints. We will consider this time to be zero (0) however. This is due to the fact that the detections occur while the vehicle moves from one lwanmowerpoint to another and as such the time it needs to detect objects is irrelevant since it is already included within the time the vehicle needs to traverse the distance between the lawnmowerpoints. Another thing that we would like to comment on is that the vehicle is not expected to move constantly with the same speed. There are accelerations and decelerations. Having said that, it is up to the modeller to chose a value for the multiplication factor that will reflect this. Back to the example with the vehicle speed, instead of choosing a value of 0.333 a better choice would be a higher value in order to formulate an upper bound for duration. Such a value can be either chosen empirically or learned from data. Now, back to the analysis of the `do_hover_detection` action shown in snippet 14, the `precondition` of the action `condition...` is that the vehicle `v` needs to be at lawnmowerpoint `from` at the start of the interval of the action (the point at which the action is applied), lawnmowerpoint `to` must be reachable from lawnmowerpoint `from` and lawnmowerpoint `to` must not have been visited at the start of the action's interval (for a reminder about durative actions and temporal annotations see Section 4.5.1). In addition, at the start of the action's interval the object detection phase must not have been completed up to lawnmowerpoint `lp` while

the remaining energy of the vehicle must be greater or equal (>=) than the distance travelled between lawnmowerpoints `from` and `to` multiplied by the energy consumption rate that the vehicle has while moving. Recall from earlier in this section where we presented the `energy_consumption_rate_moving...` function we stated that it represents the rate at which the vehicle consumes energy while moving. As such, multiplying this with the distance travelled will provide an estimate of the consumed energy. As in the case of the multiplication factor for duration that we analysed earlier, the energy consumption rate can be chosen empirically or learned from data. The `effect` of the vehicle executing the action is that it (the vehicle) is no longer at lawnmowerpoint `from` at the start of the durative action, i.e. the effect is that the the vehicle leaves its initial location as soon as the action starts executing. Moreover, at the end of the durative action, the effect of the vehicle being at lawnmowerpoint `to` will take place, while its remaining energy will decrease relatively to the distance travelled between lawnmowerpoints `from` and `to` multiplied by the energy consumption rate that the vehicle has while moving. Now, in a similar fashion, the action estimates the vehicles total consumed energy by increasing `consumed_energy` by the same amount it has decreased the remaining energy. These two representations are redundant and one can be substituted for the other. However we provide both because they represent different styles. Continuing with the effects, of the `do_hover_detection` action, at the end of the action lawnmowerpoint `to` will be marked as visited, the object detection phase will be marked as completed up to lawnmowerpoint `to`, the number of visited lawnmowerpoints will be increased by one and the duration of the mission will increase by the same amount as the duration of the action.

Let us now proceed with the durative `do_classify` action that is illustrated in Snippet 15. Action `do_classify` is intended for plans where the goal is to classify object detections. The `parameters` of the action are a vehicle `v` and a lawnmowerpoint `lp`. The duration of the actions is estimated to be the time that the vehicle needs to classify the objects. Notice that there is no time multiplication factor as in the case of the `do_hover_detection` action since as we will see next the vehicle just holds its position to classify objects and does not traverse any distance. Continuing with the analysis of the action, its `precondition` (`condition...`) is that the vehicle needs to: be at lawnmowerpoint `lp` at the start of the action's interval (the point at which the action is applied) as well as to have visited a number of lawnmowerpoints as the start of the action's interval. The notation `# of lawnmowerpoints` is just a place-holder for demonstration purposes. It is not a valid syntax. To be valid it needs to be replaced by a number. Finally, at the start of the action's interval the object classification phase must have not been completed by the vehicle while the vehicle must have remaining energy that is greater or equal (>=) than the rate at which the vehicle consumes energy while maintaining its position multiplied by the time that the vehicle needs to classify the objects. That is, the per second rate. Note that this is different compared to what we presented for the the `do_hover_detection`. Here we have no transition and as such we will not be using `energy_consumption_rate_moving...` but `energy_consumption_rate_still...`. Note that this rate needs to be determined empir-

ically or be learned from data. The `effect` of executing `do_classify` is that the vehicle

Snippet 15 Temporal PDDL domain do_classify durative action.

```
(:durative-action do_classify
 :parameters (?v - Vehicle ?lp - Lawnmowerpoint)
 :duration (= ?duration (class_obj_time ?v))
 :condition (
                 and (at start (at_lp ?v ?lp))
                 (at start (= (cnt_visited_lp ?v) # of lawnmowerpoints))
                 (at start (= (obj_classification_complete ?v) 0))
                 (at start (>= (remaining_energy ?v) (* (class_obj_time ?v)
                 (energy_consumption_rate_still ?v))))
               )
 :effect (
           and (at end (at_lp ?v ?lp))
           (at end (increase (obj_classification_complete ?v) 1))
           (at end (decrease (remaining_energy ?v) (* (class_obj_time ?v)
           (energy_consumption_rate_still ?v))))
           (at end (increase (consumed_energy ?v) (* (class_obj_time ?v)
           (energy_consumption_rate_still ?v))))
           (at end (increase (mission_duration) ?duration))
         )
)
```

at the end of the action's duration remains at its position (at lawnmowerpoint `lp`) and that the object classification phase has been completed by the vehicle. Moreover, at the end of the action the remaining energy will decrease relatively to the time that the vehicle needs to classify the objects multiplied by the rate at which the vehicle consumes energy while maintaining its position. In a identical fashion the `do_classify` action has as an effect for increasing the consumed energy at the end of the action's interval. Finally, at the end of the action's interval the duration of the mission will increase by the same amount as the duration of the action. The next action to present and analyse is the `do_reacquire` durative action which is illustrated in Snippet 16.

Durative action `do_reacquire` is intended for plans where the goal is to reacquire objects in the environment and has three parameters: a vehicle `v` and two objectpoints `from` and `to`. The estimation of the duration of the action is identical to the one presented for the `do_hover_detection` action (see Snippet 14) but instead of adding the time that the vehicle requires to make detections we add the time the vehicle requires to reacquire an object. The value for the time that the vehicle needs to reacquire an object can be chosen empirically or learned from data. Now, for the action to be applicable, at the start of the action's interval (the point at which the action is applied), the vehicle needs to be at objectpoint (location) `from`, objectpoint `to` must be reachable from objectpoint `from` and objectpoint `to` must have not been visited. Moreover, at the start of the action's interval, the object must have not been reacquired. As a final precondition the vehicle must have energy greater or equal ($>=$) than the following: the distance between the objectpoints `from,to` multiplied by the energy consumption rate of the vehicle while moving plus the rate at which the vehicle consumes energy while maintaining its position multiplied by the time the vehicle needs to reacquire an object. This is the most complex expression so far

Snippet 16 Temporal PDDL domain do_reacquire durative action.

```
(:durative-action do_reacquire
 :parameters (?v - Vehicle ?from ?to - Objectpoint)
 :duration (= ?duration (+(*(distance_op ?from ?to)
 (mult_fact_time ?v)) (reacq_obj_time ?v)))
 :condition (
                 and (at start (at_op ?v ?from))
                 (at start (reachable_op ?from ?to))
                 (at start (= (visited_op ?to) 0))
                 (at start (= (reacquired_obj ?to) 0))
                 (at start (>= (remaining_energy ?v) (+(* (distance_op ?from ?to)
                 (energy_consumption_rate_moving ?v)) (*(reacq_obj_time ?v)
                 (energy_consumption_rate_still ?v)))))
               )
 :effect (
           and (at start (not (at_op ?v ?from))
           (at end (at_op ?v ?to))
           (at end (decrease (remaining_energy ?v) (+(* (distance_op ?from ?to)
           (energy_consumption_rate_moving ?v)) (*(reacq_obj_time ?v)
           (energy_consumption_rate_still ?v)))))
           (at end (increase (consumed_energy ?v) (+(* (distance_op ?from ?to)
           (energy_consumption_rate_moving ?v)) (*(reacq_obj_time ?v)
           (energy_consumption_rate_still ?v)))))
           (at end (increase (visited_op ?to) 1))
           (at end (increase (reacquired_obj ?to) 1))
           (at end (increase (cnt_reacquired_obj ?v) 1))
           (at end (increase (prob_obj_quotient_sum ?v)
           (/(prob_obj ?to) (prob_obj ?from))))
           (at end (increase (ent_obj_quotient_sum ?v)
           (/(ent_obj ?to) (ent_obj ?from))))
           (at end (increase (mission_duration) ?duration))
         )
)
```

in terms of calculating the energy. It has two components: i) the first is the energy that the vehicle is estimated to require in order to transition from one objectpoint to the other and ii) the energy it is estimated to consume in order to reacquire the object. That is, when the vehicle will stop at objectpoint `to` it will maintain its position for some time in order to reacquire the object. As opposed to `det_obj_time...` that we presented in the detection action, `reacq_obj_time...` is not zero and as such it needs to be determined either empirically or learned from data. The effect of executing the `do_reacquire` action is that the vehicle has left its initial location at the start of the action (`(at start (not (at_op ?v ?from)))`) and moved to the location of objectpoint `to`. Additionally, at the end of the action's interval, the vehicle's remaining and consumed energy will change according to what we have presented above as being the energy the vehicle is estimated to require to perform the action. Furthermore, at the end of the action's interval, the effect of the objectpoint `to` being marked as visited will take place, the effect of the object as being reacquired will take place and the number of reacquired objects will increase by one. Also, `at end (increase (prob_obj_quotient_sum ?v) (/(prob_obj ?to) (prob_obj ?from)))` increases, at the end of the action's interval, the sum of probability quotients by the quotient of dividing the probability of the object at location (objectpoint) `to` being an object of interest by the probability of the object at location (objectpoint) `from` being an object of interest. The effect `at end (increase (ent_obj_quotient_sum ?v) (/(ent_obj ?to) (ent_op ?from)))` achieves the same outcome but with entropy instead of probability. Both these effects regarding probability and entropy are related to high level mission priorities which are described in Section 7.3.1.2. Finally, at the end of the action's interval the duration of the mission will increase by the same amount as the duration of the action.

The last action that is encoded in our domain is the durative action `do_inspect` which is illustrated in Snippet 17. `Do_inspect` is intended for plans where the goal is to inspect objects of interest and has three parameters: a vehicle `v` and two inspectionpoints `from` and `to`. The duration of the action is estimated in the same manner as the duration of the `do_reacquire` action shown in Snippet 16. The only difference is that instead of using the time that the vehicle needs to reacquire an object (`reacq_obj_time...`) we use the time that the vehicle needs to inspect an object of interest from an inspectionpoint (`insp_obj_time...`) and instead of using the distance between two objectpoints we use the distance between two inspectionpoints. The action precondition (`condition...`) is that at the start of the action's interval (the point at which the action is applied) the vehicle needs to be at inspectionpoint `from`, inspectionpoint `to` must be reachable from inspectionpointpoint `from` and inspectionpoint `to` must not have been visited. As a final precondition the vehicle must have energy greater or equal ($>=$) than the following: the distance between the inspectionpoints `from,to` multiplied by the energy consumption rate of the vehicle while moving plus the rate at which the vehicle consumes energy while maintaining its position multiplied by the time the vehicle needs to inspect an object from an inspectionpoint. Similar to the case of the `do_reacquire` action the energy that the vehicle is estimated to consume for executing the inspection action has two components: i) the first is the energy

Snippet 17 Temporal PDDL domain do_inspect durative action.

```
(:durative-action do_inspect
 :parameters (?v - Vehicle ?from ?to - Inspectionpoint)
 :duration ( = ?duration (+(*(distance_ip ?from ?to) (mult_fact_time ?v))
 (insp_obj_time ?v)))
 :condition (
                 and (at start (at_ip ?v ?from))
                 (at start (reachable_ip ?from ?to))
                 (at start (= (visited_ip ?to) 0))
                 (at start (>= (remaining_energy ?v) (+(* (distance_ip ?from ?to)
                 (energy_consumption_rate_moving ?v)) (*(insp_obj_time ?v)
                 (energy_consumption_rate_still ?v)))))
               )
 :effect (
           and (at start (not (at_ip ?v ?from)))
           (at end (at_ip ?v ?to))
           (at end (decrease (remaining_energy ?v) (+(* (distance_ip ?from ?to)
           (energy_consumption_rate_moving ?v)) (*(insp_obj_time ?v)
           (energy_consumption_rate_still ?v)))))
           (at end (increase (consumed_energy ?v) (+(* (distance_ip ?from ?to)
           (energy_consumption_rate_moving ?v)) (*(insp_obj_time ?v)
           (energy_consumption_rate_still ?v)))))
           (at end (increase (visited_ip ?to) 1))
           (at end (increase (inspected_obj ?to) 1))
           (at end (increase (mission_duration) ?duration))
         )
)
```

that the vehicle will require in order to transition from one inspectionpoint to the other and ii) the energy it will consume in order to inspect the object. That is, when the vehicle will stop at inspectionpoint `to` it will maintain its position for some time in order to inspect the object from that position. `Insp_obj_time...` needs to be specified by the user. The effect of executing the `do_inspect` action is that the vehicle has left its initial location at the start of the action (`(at start (not (at_ip ?v ?from)))`). Moreover, at the end of the action's interval the vehicle moves to the location of the inspectionpoint `to` and the vehicle's remaining and consumed energy change based on the energy estimate that we presented above while analysing the preconditions of the action. In addition, at the end of the action's interval the effect of marking inspectionpoint `to` as visited takes place as is marking the object as inspected from the same location. Finally, at the end of the action's interval the duration of the mission will increase by the same amount as the duration of the action.

7.3.1.2 High Level Mission Priorities

High level mission priorities are taken into consideration for the reacquisition of objects. Changes in high level mission priorities can occur in real time and vehicles should be able to accommodate and act upon them. Addressing the aforementioned challenge is a non-trivial task. We assume the operator is able to communicate changes in mission priorities to the AUV along the low-bandwidth acoustic channel, for instance in response to a changing situation that the vehicle cannot monitor. We have considered three high level mission

priorities in total.

The first, which we call energy-efficient reacquisition, is intended for generating mission plans that minimize the consumed energy of the AUV or equivalently maximize the remaining energy of the AUV for reacquiring and inspecting objects. That is, plans for reacquiring and inspecting objects in a manner that the smallest possible distance is travelled given that: i) the rate at which the vehicle consumes energy while moving is assumed to be the same for all actions of the same type, ii) the rate at which the vehicle consumes energy while maintaining its position is assumed to be the same for all actions and iii) quantities such as the time that the vehicle needs to classify objects or the time that the vehicle needs to inspect an object from a position etc. are considered to be the same for their respective action types. Minimizing energy consumption is achieved by doing two things. The first has already been described in Section 7.3.1.1 and it involves functions `remaining_energy` and `consumed_energy` in conjunction with distance functions, energy consumption rate functions as well as time functions and their role as part of the preconditions and effects of actions do_reacquire and do_inspect. The second is to use the following metric within the PDDL problem definition[6]: `metric minimize consumed_energy ?v` or equivalently `metric maximize remaining_energy ?v`.

The second high level mission priority is intended for generating mission plans in which the AUV reacquires and inspects detected objects that are the most probable to be objects of interest first, we call this probability-efficient reacquisition, while concurrently minimizing the energy that it consumes (this is the energy-efficient reacquisition that we described as the first high level mission priority). This effectively formulates a multi-objective optimization problem in which we want to have an energy-efficient plus probability-efficient reacquisition which combined, constitute the second high level mission priority. To solve this problem we employ the weighted sum scalarizing method that scalarizes a set of objectives into a single objective by multiplying each objective with a weight. The weighted sum method is given by the following formulas:

$$min \quad F(x) = \sum_{i=1}^{n} w_i f_i(x) \tag{7.1}$$

$$s.t. \quad w_i \geq 0, \quad i \in 1,...,n \tag{7.2}$$

$$\sum_{i=1}^{n} w_i = 1 \tag{7.3}$$

where $F(x)$ is the single objective, $f_i(x)$ is the set of objectives and w_i the weights. We have already presented the energy-efficient reacquisition objective. Now, the probability-efficient reacquisition objective in our case is formulated as minimizing the sum of probability quotients (`prob_obj_quotient_sum ?v`) as shown in Section 7.3.1.1. At this point, an example will be really helpful in clarifying how this computation is made.

Let the outcome of classification be three objects (A, B, C) with probabilities of A = 0.53, B = 0.69 and C = 0.84, with each probability associated with each object representing

[6]PDDL problems will be investigated in Section 7.3.1.3.

how likely it is for the respective object to be an object of interest. Also, let the vehicle be at some location from which it needs to start moving towards the objects in order to reacquire them. Probability-efficient reacquisition requires that the objects are reacquired in a probability efficient manner, that is, higher probability objects should be reacquired first, i.e. before lower probability objects in a succession of reacquisitions. Even though the vehicle is at some initial location *init* which is not associated with any kind of object or probability we assign an arbitrarily large number to that location that represents a "probability" so that the computation of probability-efficient reacquisition can take place. That is, minimizing the sum of probability quotients with each quotient being calculated as shown in effect of the `do_reacquire` action in Snippet 16. That is, dividing the probability of the object at the location that the vehicle will transition to with the probability of the object at the location that the vehicle will transition from. As we mentioned earlier, the initial location of the vehicle is not associated with any object or probability, however we assign an arbitrarily large number to represent a "probability", say 1000. Let us now calculate the sum of probability quotients for each possible reacquisition order which is shown in Table 7.3. As can

Reacquisition Sequence	**Sum of Quotients**
init $\rightarrow$ A $\rightarrow$ B $\rightarrow$ C	2.5198
init $\rightarrow$ A $\rightarrow$ C $\rightarrow$ B	2.4068
init $\rightarrow$ B $\rightarrow$ A $\rightarrow$ C	2.3537
init $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A	1.8490
init $\rightarrow$ C $\rightarrow$ A $\rightarrow$ B	1.9336
init $\rightarrow$ C $\rightarrow$ B $\rightarrow$ A	1.5904

Table 7.3. Sum of probability quotients for each possible reacquisition sequence.

be observed by looking at Table 7.3 the minimum value for the sum of probability quotients corresponds to the sequence init $\rightarrow$ C $\rightarrow$ B $\rightarrow$ A in which the vehicle will reacquire targets. This sequence, is the sequence in which higher probability objects are reacquired before lower probability objects. In our case $0.84 > 0.69 > 0.53$.

Remember that we need both energy-efficient plus probability-efficient reacquisition combined in one objective. In order to apply the weighted sum to form a single objective we need to normalize each objective so that it is expressed in the same range. The linear normalization is given by:

$$f_N(x) = (f(x) - oldMin)\frac{newRange}{oldRange} + newMin \tag{7.4}$$

where, $f_N(x)$ is the normalized objective, $f(x)$ is the original one, $oldMin$ is the minimum value of $f(x)$, $newRange$ ($newMax - newMin$) is the range of $f_N(x)$ that we desire, $oldRange$ ($oldMax - oldMin$) is the range of $f(x)$ and $newMin$ is the desired minimum value of $f_N(x)$.

The third high level mission priority in intended for generating mission plans in which the AUV reacquires and inspects detected objects that have the highest entropy first, we call this entropy-efficient reacquisition, while concurrently minimizing the the energy that

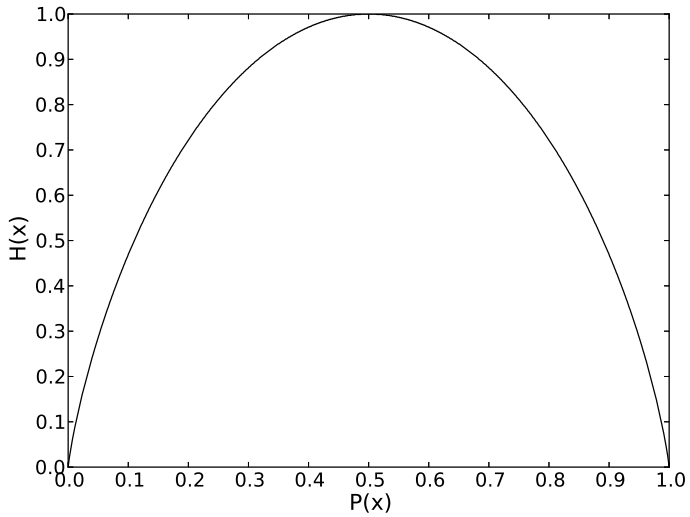


Figure 7.13. Entropy versus Probability

it consumes (energy-efficient reacquisition). Again, this formulates a multi-objective optimization problem in which we want to have an energy-efficient plus entropy-efficient reacquisition which combined, constitute the third high level mission priority. The entropy-efficient reacquisition objective is formulated as minimizing the sum of entropy quotients (`ent_obj_quotient_sum ?v`). The process of calculating the entropy quotients is the same as the one described for probabilities but this time instead of probabilities we use entropies. In our case entropy H is an indication of the uncertainty level about an object being an object of interest ($H(OBJ_i)$) and is related to the probability of an object being an object of interest ($P(OBJ_i)$) as follows [145]:

$$H(OBJ_i) = -P(OBJ_i)log_2P(OBJ_i) - (1 - P(OBJ_i))log_2(1 - P(OBJ_i)) \qquad (7.5)$$

This third high level mission priority is ideal for cases where we want to prioritize high uncertainty reduction and high information gain first in an energy-efficient manner. Figure 7.13 illustrates the relationship between entropy and probability. At this point we would like to clarify something, inspection always happens in an energy efficient manner. Entropy and probability are only considered during reacquisition.

Regarding the presentation of high level mission priorities in this section one might wonder about the minimization of mission execution time since, after all, we have presented a temporal domain with durative actions in Section 7.3.1.1. The formulation of such a high level mission priority is not prohibited by the temporal domain and the durative actions we have presented. However, the minimization of the total execution duration is already subsumed under the energy-efficient high level mission priority. More specifically, by observing how we estimate energy consumption and the duration of actions (see Section 7.3.1.1) one can easily identify that the core element for both is the distance travelled. For instance, in the case of the reacquisition action the multiplication factor for time will be the

same for every action of the same type as will be the estimate for the time that the vehicle requires to reacquire an object[7]. As such, reacquiring objects in different orders will yield different distances for each action instantiation and different total mission execution durations. In the same manner, the energy consumption rate for the vehicle while moving is assumed to be the same for every reacquisition action as is the energy consumption rate for the vehicle while maintaining its position and the estimated time that the vehicle requires to reacquire an object. Again, reacquiring objects in different orders will yield different distances for each action instantiation and different consumed energy for each mission. In both cases, that of duration and that of consumed energy it is the minimization of distance that will yield the optimized reacquisition both in terms of duration and energy. As such, by minimizing for the estimated consumed energy of the vehicle or by maximizing for the remaining energy of the vehicle we subsume the minimization of the estimated execution duration. This also applies for classification actions where there are no transitions in space for the vehicle and as such we do not utilize distances as the basis for estimating durations and energy consumptions. By observing the classification action in Section 7.3.1.1 one can identify that both duration and energy consumption depend on the estimated time that the vehicle requires to classify objects. Regarding energy consumption in particular, it also depends on the energy consumption rate of the vehicle while it maintains its position. However, because for every classification action that rate is assumed to be the same, both duration and energy consumption are driven by the same quantity. As such, minimizing energy consumption or maximizing the remaining energy subsumes the minimization of estimated execution duration.

7.3.1.3 PDDL Problems

According to the Oxford English dictionary, a `problem` is defined as "a situation, person, or thing that needs attention and needs to be dealt with or solved". In the context of generating mission plans we similarly define a `problem` as "a situation that requires attention and needs to be dealt with or solved to achieve some goal". Every such situation comprises elements that are in some state which is different from the state that is dictated by the goal that is set. For a reminder about the structure of problems refer to Section 4.5.1. In this section we provide four snippets illustrating the PDDL problem definitions for detecting, classifying, reacquiring and inspecting objects of interest given the temporal domain described in Section 7.3.1.1.

Let us first begin with the problem of detecting objects in a rectangular area for which we utilize a lawnmower pattern trajectory search with three legs. Let the first leg be from lawnmowerpoint `lp1` to lawnmowerpoint `lp2`, the second from `lp3` to `lp4` and the third from `lp5` to `lp6`. Also let `auv` be the vehicle that will be used while `lp7` be the location from which the vehicle starts. These effectively constitute the `objects` in our problem as can

[7]Obviously we are not referring to the whole duration of the reacquisition action which also involves the transition from some location to the location of the object where reacquisition will take place. We refer to the time the vehicle needs to reacquire the object when it (the vehicle) is in position.

be observed by looking at Snippet 18. The reason for declaring lp7 as a lawnmowerpoint

Snippet 18 PDDL lawnmower object detection problem objects and initial state.

```
(:objects
        lp1 lp2 lp3 lp4 lp5 lp6 lp7 - Lawnmowerpoint
        auv - Vehicle
)
(:init
        (at_lp auv lp7)
        (= (visited_lp lp7) 1)
        (reachable_lp lp7 lp1)
        (= (distance_lp lp7 lp1) 5.00)
        (reachable_lp lp1 lp2)
        (= (distance_lp lp1 lp2) 20.00)
        (= (visited_lp lp1) 0)
        (= (obj_detection_complete lp1) 0)
        ...
        ...
        (reachable_lp lp5 lp6)
        (= (distance_lp lp5 lp6) 20.00)
        (= (visited_lp lp5) 0)
        (= (obj_detection_complete lp5) 0)
        (= (visited_lp lp6) 0)
        (= (obj_detection_complete lp6) 0)
        (= (cnt_visited_lp auv) 0)
        (= (remaining_energy auv) 30000.00)
        (= (consumed_energy auv) 0)
        (= (energy_consumption_rate_moving auv) 3)
        (= (mult_fact_time nessie1) 4)
        (= (det_obj_time nessie1) 0)
        (= (mission_duration) 0)
)
```

is purely because the planner would not be able to form a plan in which the vehicle could transition from its starting location to lp1 given that the parameters of the do_hover_detection action are a vehicle and two lawnmowerpoints. An alternative would be to have included some sort of positioning action in the domain declaration which would take as parameters a vehicle, a location outside the lawnmower trajectory and a lawnmowerpoint. As such, that action would be placed first in the action sequence (plan) that the planner would form and then it would proceed with a sequence of do_hover_detection actions instead of formulating detection plans just with do_hover_detection actions. However this choice of ours does not affect the planning process in any other way and by no mean does it affect the essence or the outcome of the process in the first place. Having said that, let us now dig into the initial state as it is depicted in Snippet 18. Initially the vehicle is at lp7 which has consequently been visited. In order to be able to encode the lawnmower trajectory we use a series of visited_lp and obj_detection_complete functions as well as reachable_lp predicates. More specifically, we have encoded that lp1 is only reachable from lp7, lp2 from lp1, lp3 from lp2, ..., lp6 from lp5. In this manner the sequence of lawnmowerpoints that form the lawnmower pattern will be preserved. Moreover, in order to represent the reality, we have encoded that none of the actual lawnmowerpoints has been visited and that the object detection phase has not been completed up to any lawn-

mowerpoint by utilizing initial values for the `visited_lp` and `obj_detection_complete` functions at zero. The counter function of visited lawnmowerpoints is also initialized at zero. Furthermore we have encoded the distance between consecutive lawnmowerpoints (in meters) as well as the energy at which the vehicle consumes energy while moving so that the `remaining_energy` and `consumed_energy` functions can calculate the remaining and consumed energy of the vehicle for the whole plan execution. In addition, we have chosen a value of four for the time multiplication factor used in estimating the duration of actions. Note that the aforementioned values are for demonstration purposes and, as we mentioned before need to be chosen empirically or learned from data. Also, for two different vehicles values are expected to be different. We are now ready to proceed with the goal state which is illustrated in Snippet 19. By observing Snippet 19 we can see that,

Snippet 19 PDDL lawnmower object detection problem goal state.

```
(:goal
        (and
        (= (visited_lp lp1) 1)
        (= (obj_detection_complete lp1) 1)
        (= (visited_lp lp2) 1)
        (= (obj_detection_complete lp2) 1)
        (= (visited_lp lp3) 1)
        (= (obj_detection_complete lp3) 1)
        (= (visited_lp lp4) 1)
        (= (obj_detection_complete lp4) 1)
        (= (visited_lp lp5) 1)
        (= (obj_detection_complete lp5) 1)
        (= (visited_lp lp6) 1)
        (= (obj_detection_complete lp6) 1)
        (>= (remaining_energy auv) 0)
        )
)
```

as expected, the goal is for every lawnmowerpoint to have been visited, object detection to have been completed in the whole search area. In addition the remaining energy of the vehicle must be greater or equal ($>=$) to zero to indicate that given the initial remaining vehicle energy set at some maximum value, the vehicle is able to perform the lawnmower pattern. That is, it might not have enough energy to do so. In that case the problem becomes unsolvable.

Let us now present the problem of classifying detections. This is a simple problem with respect to its structure as can be observed by looking at Snippet 20. The only `objects` required for the problem are the vehicle `auv` and lawnmowerpoint `lp6`. Lawnmowerpoint `lp6` is the location where the vehicle stopped when it finished searching for objects. This is reflected in the problem's `initial` state where the number of visited lawnmowerpoints is six, i.e. the number of lawnmowerpoints that were provided for the detection problem. Furthermore, as part of the initial state we encode that object classification phase must have not been completed before. This is meant to indicate that classification occurs only once. We also, encode the time the vehicle needs to classify objects (in secs), the remaining energy of the vehicle as well as the energy consumption rate at which the vehicle consumes energy

Snippet 20 PDDL object classification problem.

```
(:objects
        lp6 - Lawnmowerpoint
        auv - Vehicle
)
(:init
        (at_lp auv lp6)
        (= (cnt_visited_lp auv) 6)
        (= (obj_classification_complete auv) 0)
        (= (class_obj_time auv) 5)
        (= (remaining_energy auv) 30000)
        (= (energy_consumption_rate_still auv) 1.5)
        (= (mission_duration) 0)
)
(:goal
        (and
        (= (obj_classification_complete auv) 1)
        (>= (remaining_energy auv) 0)
        )
)
```

while maintaining its position. Finally, we instantiate the mission duration at zero. The values chosen for the energy consumption rate, the remaining energy and the time required to classify an object are for demonstration purposes. Finally, the goal is, as expected, for the vehicle to perform the classification of detections and the remaining energy of the vehicle after classification to be greater or equal ($>=$) to zero. We assume that the vehicle is equipped with a probabilistic classification system that yields classified objects with some probability.

Moving on, we have the problem of reacquiring the objects that the classification phase has yielded. For the sake of our running example let us assume that the outcome of classification was two objects of interest found at objectpoint locations `op1` and `op2`. With respect to our problem we have four PDDL objects as illustrated in Snippet 21. The two objectpoints that we just mentioned plus a vehicle `auv` and another objectpoint `op3` which in reality is the location from where the vehicle will start its object reacquisition phase. Similar to the case of the detection problem where `lp7` was encoded as a lawnmowerpoint, `op3` is encoded as an objectpoint due to the the `do_reacquire` action parameters being a vehicle and two objectpoints. Therefore, the vehicle initially needs to be at an "objectpoint" (`op3`) to be able to transition to either `op1` or `op2` to reacquire the first object. Again, this does not affect the planning essence or outcome in any way. The remainder of the `initial` state is that `op3` has already been visited while `op1` and `op2` have not, and consequently the objects at those locations have not been reacquired. Furthermore, we encode that from `op3` the vehicle can transition to any other objectpoint but it cannot transition back to `op3` since `op3` is not reachable from any other objectpoint. Distances between objectpoints are also calculated based on their positions, hence they are known. In addition, the probabilities of each object being an object of interest are also known from the classification phase and we additionally convert them to entropy based on equation 7.5. Moreover, the values

Snippet 21 PDDL reacquisition problem objects and initial state.

```
(:objects
        op1 op2 op3 - Objectpoint
        auv - Vehicle
)
(:init
        (at_op auv op3)
        (= (visited_op op1) 0)
        (= (visited_op op2) 0)
        (= (visited_op op3) 1)
        (= (reacquired_obj op1) 0)
        (= (reacquired_obj op2) 0)
        (reachable_op op3 op1)
        (reachable_op op3 op2)
        (reachable_op op1 op2)
        (reachable_op op2 op1)
        (= (distance_op op3 op1) 6.05)
        (= (distance_op op3 op2) 4.32)
        (= (distance_op op1 op2) 7.19)
        (= (distance_op op2 op1) 7.19)
        (= (prob_obj op1) 0.606)
        (= (ent_obj op1) 0.967)
        (= (prob_obj op2) 0.755)
        (= (ent_obj op2) 0.803)
        (= (prob_obj op3) 1000.0)
        (= (ent_obj op3) 1000.0)
        (= (cnt_reacquired_obj auv) 0)
        (= (prob_obj_quotient_sum auv) 0)
        (= (ent_obj_quotient_sum auv) 0)
        (= (remaining_energy auv) 30000)
        (= (consumed_energy auv) 0)
        (= (energy_consumption_rate_moving auv) 3)
        (= (energy_consumption_rate_still auv) 1.5)
        (= (mult_fact_time nessie1) 4)
        (= (reacq_obj_time nessie1) 2)
        (= (mission_duration) 0)
)
```

of `prob_obj_quotient_sum auv` and `ent_obj_quotient_sum auv` are initially zero, the vehicle has not consumed any energy, the remaining energy is at some maximum value, and the counter function of reacquired objects is initialized at zero. Finally, the rates at which the vehicle consumes energy while moving and while keeping its position are initialized as is the time multiplication factor for estimating action duration, the time that the vehicle requires to reacquire each object and the mission duration. With respect to the `goal` state of the problem as well as the high level mission priority, we encode that the remaining energy of the vehicle should be greater or equal (>=) to zero and that all objects must be reacquired in an energy-efficient manner (see Snippet 22). Also refer to section 4.5.1 for a reminder about plan metrics.

Snippet 22 PDDL reacquisition problem goal state and energy efficiency as the high level mission priority.

```
(:goal
        (and
        (>= (remaining_energy auv) 0)
        (= (cnt_reacquired_obj auv) 2)
        )
)
(:metric minimize (consumed_energy auv))
```

The last problem to present is the one of inspecting reacquired objects who are deemed to be objects of interest. For inspection we create six, equally distributed, inspection points around every object of interest which the vehicle visits in order to inspect the object from each one of them. The PDDL objects as well as the initial state for the inspection problem of one object of interest is illustrated in Snippet 23. Inspectionpoints `ip1-ip6` are the inspectionpoints around the object while `ip7` is the "inspectionpoint" at which the vehicle starts its mission moving towards the first inspectionpoint similar to the phases of detection and reacquisition with `lp7` and `op3` respectively. Regarding the `initial` state of the inspection problem, we encode that the vehicle is at `ip7` which is visited. Also, we encode that: i) every inspectionpoint is reachable from any other inspectionpoint, ii) `ip1-ip6` are not visited, iii) the object has not been inspected from the inspectionpoint positions. Finally, we encode: i) the distances between all inspectionpoints, ii) the remaining energy being at some maximum value, iii) the consumed energy being at zero, iv) the rates at which the vehicle consumes energy while moving and while keeping its position, iv) the time multiplication factor for estimating action duration and the time that the vehicle requires to inspect each object from each inspectionpoint. Since with the inspection problem we want to inspect an object it is only logical to encode in the `goal` state that the object must be inspected by the vehicle from each inspection point and that the remaining energy of the vehicle must be enough to do so (see Snippet 24). In addition, we instruct the planner to generate a plan with energy efficiency in mind by utilizing `metric minimize (consumed_energy auv)`.

Snippet 23 PDDL inspection problem objects and initial state.

```
(:objects
        ip1 ip2 ip3 ip4 ip5 ip6 ip7- Inspectionpoint
        auv - Vehicle
)
(:init
        (at_ip auv ip7)
        (= (visited_ip op7) 1)
        (= (distance_ip ip7 ip1) 2.34)
        (= (distance_ip ip1 ip7) 2.34)
        (= (reachable_ip ip7 ip1)
        (= (reachable_ip ip1 ip7)
        (= (inspected_obj ip1) 0)
        ...
        ...
        (= (visited_ip ip6) 0)
        (= (distance_ip ip6 ip7) 2.63)
        (= (distance_ip ip7 ip6) 2.63)
        (= (reachable_ip ip6 ip7)
        (= (reachable_ip ip7 ip6)
        (= (inspected_obj ip6) 0)
        (= (remaining_energy auv) 30000)
        (= (consumed_energy auv) 0)
        (= (energy_consumption_rate_moving auv) 3)
        (= (energy_consumption_rate_still auv) 1.5)
        (= (mult_fact_time nessie1) 4)
        (= (insp_obj_time nessie1) 2)
        (= (mission_duration) 0)
)
```

Snippet 24 PDDL inspection problem goal state and energy efficiency as the high level mission priority.

```
(:goal
        (and
        (= (inspected_obj ip1) 1)
        (= (inspected_obj ip2) 1)
        (= (inspected_obj ip3) 1)
        (= (inspected_obj ip4) 1)
        (= (inspected_obj ip5) 1)
        (= (inspected_obj ip6) 1)
        (>= (remaining_energy auv) 0)
        )
)
(:metric minimize (consumed_energy auv))
```

7.3.2 The Planning Ontology

Typically a PDDL planner utilizes information encoded in a PDDL `domain` and a PDDL `problem` file in .pddl format in order to generate mission plans (see Section 7.3.1). In a framework that is intended for persistently autonomous operations, information management, exchange and sharing is critical for successful mission planning and execution. Ergo, the knowledge representation structure used by the planner and the one used by the rest of the framework should be integrated through a common basis to maximize efficiency. For this purpose we have created a planning ontology that is capable of encoding the syntax and the semantics of PDDL domains and problems as well as high level mission priorities in the context of persistently autonomous operations. In this manner the user can query for information that was previously only accessible by the planner. The framework uses the planning ontology to reconstruct the domain and problems that we saw in Sections 7.3.1.1 and 7.3.1.3 dynamically and send them to the planner in order for the planning process to commence. The ontology also imports the world and the vehicle ontologies and by extension the actions, the capabilities and the components ontologies. The reason for doing so is to make all the necessary information available to the planning ontology. Figure 7.14 illustrates the class hierarchy of the ontology with the ontologies that it imports omitted for better readability. By having a closer look at the ontology we can conceptually divide it into two main parts: i) classes that are related to PDDL planning domains and ii) classes that are related to PDDL planning problems. Let us first start with the classes as well as instances and properties related to PDDL planning domains.

Class *PDDLDomain* is a generic class that is intended for encoding planning domains. Given the domains we are interested in we can encode specialized subclasses to describe it; in our case subclass *PDDLAutOpDomain* which encodes the syntax and semantics of the domain of autonomous operations described in Section 7.3.1.1. In order to do so we created an instance of the domain, *PDDLAutOpDomain1*, and related it to domain type, predicate, function and action instances via *hasPDDLDomainType*, *hasPDDLDomainPredicate*, *hasPDDLDomainFunction* and *hasPDDLDomainAction* object properties as shown in Table 7.4. For every domain type, predicate, function and action we have one relation to the domain. That is, 40 relations it total. Now, each domain type, predicate and function instance is related to its "value" via a *value* string data property that represents its PDDL syntax and semantics. For instance, we have *PDDLAutOpDomainType1 value* `Lawnmowerpoint` or *PDDLAutOpDomainPredicate1 value* `at_lp ?v - Vehicle ?lp - Lawnmowerpoint` and so on. Regarding action instances, they are related to their parameters, duration, preconditions and effects via *parameter*, *duration*, *precondition* and *effect* string data properties. Using this representation the framework is able to reconstruct the PDDL domain in full performing appropriate queries for all the necessary building blocks. Before proceeding with the classes, instances and properties that are related to PDDL planning problems let us first emphasize a core difference between encoding domain and problem definitions inside an ontology.

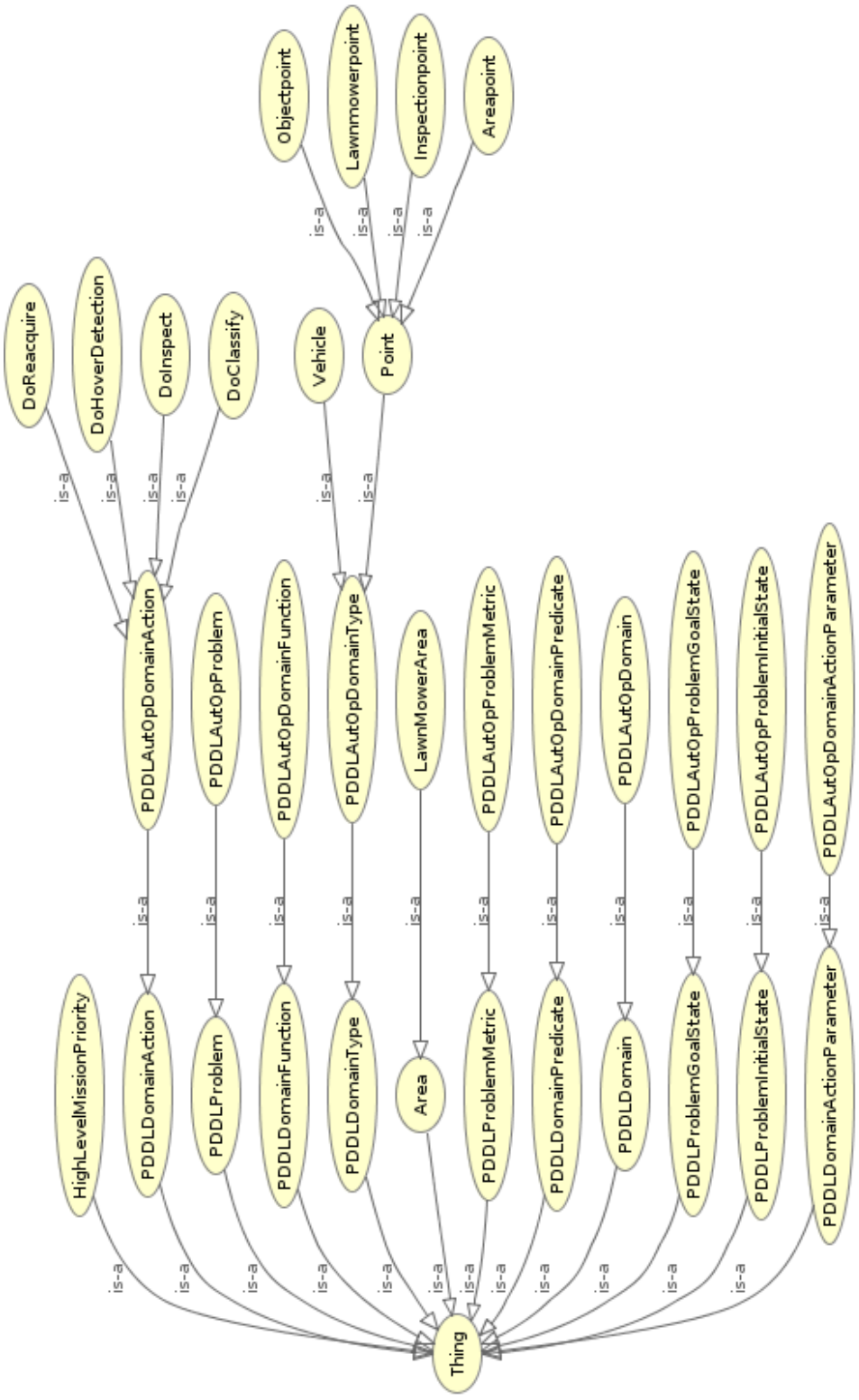


Figure 7.14. Planning ontology class hierarchy. We have intentionally omitted the world, components, capabilities, actions and vehicle ontologies for better readability.

As opposed to domain definitions, problem definitions are dynamic. For example, object types, predicates, action duration estimates, preconditions and effects, etc. do not change. As such we can statically assert them inside the ontology before starting a mission so that the system can dynamically reconstruct them and make them available to the planner. What changes though are the objects that bound the variables/parameters of domains and these are part of the problem. If we want to perform reacquisition of objects for example, we are not in the position to formulate the reacquisition problem until we have the objects. In the event that the objects are known beforehand we can assert them inside the

Domain Instance	Object Property	Building Block Instance
PDDLAutOpDomain1	hasPDDLDomainType	PDDLAutOpDomainType1 - PDDLAutOpDomainType4
	hasPDDLDomainPredicate	PDDLAutOpDomainPredicate1 - PDDLAutOpDomainPredicate6
	hasPDDLDomainFunction	PDDLAutOpDomainFunction1 - PDDLAutOpDomainFunction18
	hasPDDLDomainAction	DoHoverDetection1 DoClassify1 DoReacquire1 DoInspect1

Table 7.4. Relations between the domain instance and all its types, predicates, functions and actions which are referred to as building block instances.

planning ontology and use them to reconstruct the problem by encoding distances, probabilities etc. In the event though that we first need to detect and classify them so that we have all the necessary information, the reacquisition problem cannot be encoded in full inside the planning ontology. Consequently it cannot be reconstructed. An additional example is when a high level mission priority changes. The definition of the problem in the ontology must change as well so that it can be reconstructed accurately for the planner. Therefore, on top of dynamically reconstructing the planning problem from the ontology as in the case of the planning domain we additionally perform its encoding (dynamically) at runtime. Let us now proceed with the classes, instances and properties that are related to PDDL planning problems.

Class *PDDLProblem* is a generic class and is intended for encoding planning problems. Given the problems that we are interested in within a domain, we can encode specialized subclasses to describe it. That is, one (sub)class per problem. Equivalently, we can perform a categorization that groups all problems within a domain and encode only one problem class for that domain and have multiple instances that describe additional problems. The later is what we have chosen for our planning ontology with respect to problems. That is, we have one instance of the *PDDLAutOpProblem* class per problem. Each such instance is related to autonomous operations problem object, initial and goal state as well as high level mission priority and problem metric instances via *hasPDDLProblemObject*, *hasPDDLInitialState*, *hasPDDLGoalState*, *hasPDDLProblemPriority* and *hasPDDLProblemMetric* object properties respectively. For example, for the initial state of a problem we have *PDDLAutOpProblem1 hasPDDLInitialState PDDLAutOpProblemInitialState1* while for the goal state we have *PDDLAutOpProblem1 hasPDDLGoalState PDDLAutOpProblemGoalState1* and so on. Now, each initial and goal state instance as well as problem metric instance is related to its syntax and semantics via *value* string properties. As for high level mission priority instances, they are related to their names via *name* string data

properties. Further, high level mission priority instances are related to a *priorityChanged* boolean data property. The purpose of the aforementioned data property is to notify the executor that monitors the data property of a high level mission priority change. If a change has occurred then the executor initiates a new planning-execution cycle. How a high level mission priority and the *priorityChanged* data property are updated within the planning ontology is explained in Section 7.3.2.1. Moving on with the analysis of the planning ontology, class *Area* is a generic class that is intended for encoding search areas inside the (planning) ontology and can be specialized based on the type of search we want to perform; in our case *LawnmowerArea*. For every lawnmower area we create an instance of the class and relate it to four area points, i.e. four instances of the *Areapoint* class, via *consistsOf* object properties. These area points correspond to the edges of the rectangular area and are related to instances of the *Position3D*[8] class via *position* object properties which in turn are related to the 3D coordinates of the points via *north*, *east* and *depth* float data properties. Furthermore each lawnmower area instance has three data properties: *index*, *spacing* and overlap. Index takes integer values in the range of $[1,4]$ and is used for indicating which of the four area points will be the used as the initial point of the first lawnmower leg. Moreover, spacing is used for defining the spacing between the lawnmower legs and is expressed in meters while with overlap we define the overlap that the legs should have with each other. Overlap is expressed as a percentage. Lawnmower area related information are passed onto a script for generating lawnmowerpoints via instantiating the *Lawnmowerpoint* class shown in Figure 7.14 as appropriate. That is, each lawnmowerpoint instance is linked to *Pose6D* instances via *pose* objects properties while each *Pose6D* instance is linked to its *Position3D* and *Orientation3D* instances via *position* and *orientation* object properties respectively. To complete the representation, each *Position3D* instance is linked to its 3D position via *north*, *east*, *depth* float data properties while each and *Orientation3D* instance is linked to its 3D orientation via *roll*, *pitch*, *yaw* float data properties. Recall from Section 7.2.1 that orientation does not refer to a waypoint (lawnmowerpoint, objectpoint, inspectionpoint) per se, but rather to the orientation the vehicle should have when visiting this waypoint. Objectpoints as well as inspectionpoints are instantiated in the same manner given the detection of objects that we want to inspect. This concludes our analysis of defining planning problems within the planning ontology and by extension it concludes the analysis of the planning ontology itself.

7.3.2.1 Updating High Level Mission Priorities with Prolog

Initially, before any priority change is issued to the vehicle, the planning ontology encodes a high level mission priority for which the *priorityChanged* data property is *false*. Whenever a high level mission priority change is issued to the vehicle a set of Prolog rules which are tasked with updating the priority inside the planning ontology are triggered. The new, desired priority is first checked against the current priority and if it is the same then nothing

[8]Recall that we are importing the world ontology in our planning ontology which gives us access to all its classes and properties.

happens and the *priorityChanged* data property remains *false*. As such, the executor that monitors high level mission priorities continues executing the current plan. However, in the event that the new priority is different from the one encoded inside the planning ontology then the new priority becomes the current while at the same time the *priorityChanged* data property is switched to *true*. In this case, the executor will stop executing the current plan and update the planning problem as appropriate (including metrics) so that a new planning phase can be initiated and a new plan be created. Once the new plan is created then the *priorityChanged* data property is switched back to *false*. This is done in order for the executor to be notified that the plan that it currently holds for execution is up to date with respect to high level mission priorities. The process as a whole is repeated every time a high level mission priority change is issued to the vehicle and as was mentioned in the beginning of this section it is realised as a set of Prolog rules.

7.3.3 The execution ontology

The execution ontology is interconnected with the planning ontology and stores archival information about planning and execution outcomes. These include but are not limited to: time spent planning, mission plans, estimated duration of actions and plans, actual duration of actions and plans, action execution status (executed, pending, not executed) and outcome (successful, unsuccessful), estimated and actual start and end time points for actions and plans. While this piece of information is very important for post mission analysis it is also used by the planning ontology to update its status accordingly in real time (e.g. objects that have already been inspected do not need to be considered in the planning process any more). At this point we would like to emphasize that the estimated action duration may be different than the actual duration. Let us now analyse the execution ontology, the class hierarchy of which is illustrated in Figure 7.15, in more detail.

Class *MissionPlan* is a generic class that is intended for storing the mission plans generated by the planner. As can be observed by looking at the class hierarchy of the execution ontology the aforementioned class is specialized by the *AutOpMissionPlan* class and its four respective subclasses; *AutOpMissionDetectionPlan*, *AutOpMissionClassificationPlan*, *AutOpMissionReacquisitionPlan* and *AutOpMissionInspectionPlan* which store detection, classification, reacquisition and inspection plans intended for autonomous operations. Given a plan that is generated by the planner the frameworks formulates the appropriate class instantiation, e.g. *AutOpMissionInspectionPlan1* for an inspection plan. Each plan of course consists of actions that are represented by the instantiations of their respective mission action classes which are specializations of the *MissionAction* generic class. Given the aforementioned inspection plan instance for example, the framework instantiates six inspection actions: *AutOpMissionDoInspectAction1 – AutOpMissionDoInspectAction6*. Furthermore, the plan instance is linked to each action instance it consists of via *consistsOfMissionAction* object properties and linked to its generation time, generation duration and date as well as estimated and actual: execution duration, start and end time points via *genera-*

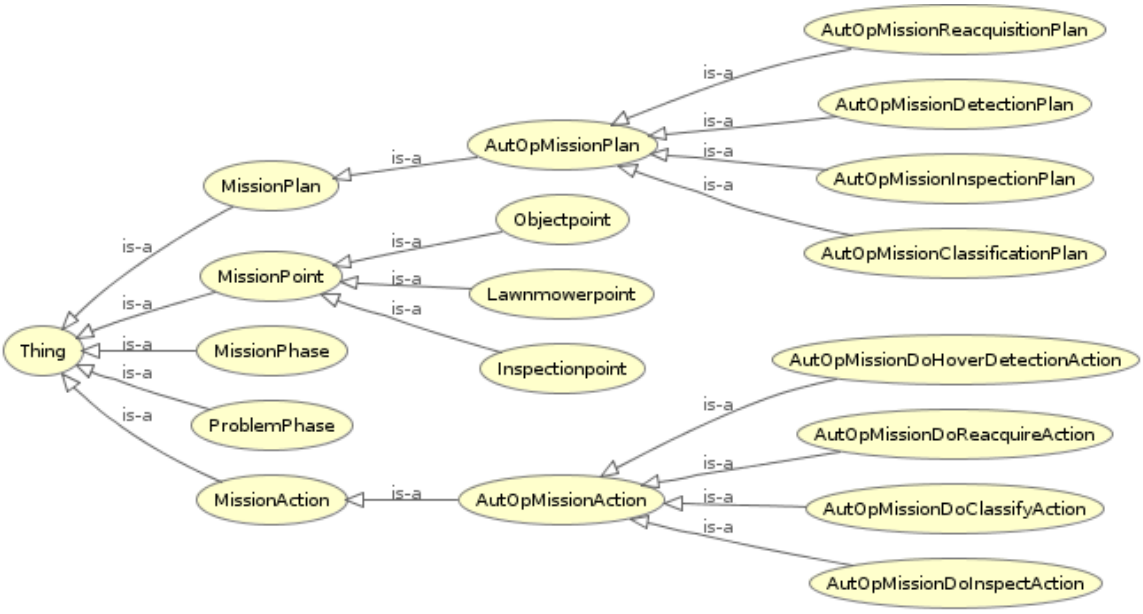


Figure 7.15. Execution ontology class hierarchy.

tionTime (float), *generationDuration* (float), *generationDate* (string), *estimatedExecutionDuration* (float), *actualExecutionDuration* (float), *estimatedStartTimePoint* (string), *actualStartTimePoint* (string), *estimatedEndTimePoint* (string) and *actualEndTimePoint* (string) data properties[9] respectively. Each instantiated action in the execution ontology comprises instantiated parameters. Instantiated parameters are planning problem objects that are also part of the planning problem ontology and were used to generate the plan at hand. However, the execution ontology has its own *Lawnmowerpoint*, *Objectpoint* and *Inspectionpoint* classes (see Figure 7.15) which the framework utilizes in order to create a duplicate representation of the aforementioned parameters. The reason for doing so is that we want to archive all action instances inside the execution ontology and by extension all instantiated parameters. Without this duplicate representation we would not be able to achieve this since every time there is a planning problem of the same nature, old waypoint instances (lawnmowerpoint, objectpoint, inspectionpoint instances) are deleted from the planning ontology and are replaced with the new ones. Ergo, the information would be lost. The relation of action instances to their parameters are formed depending on the number and sequence of the parameters in the action definition via *actionFirstParameter*, *actionSecondParameter*, ..., *actionNParameter*. Moreover, each action instance is linked to its execution status, execution outcome, execution duration and execution start and end time points via *actionStatus* (string), *actionOutcome* (string), *actionEstimatedDuration* (float), *actionActualDuration* (float), *actionEstimatedStartTimePoint* (string), *actionActualStartTimePoint* (string), *actionEstimatedEndTimePoint* (string), *actionActualEndTimePoint* (string) data properties. Note that the *actionEstimatedDuration* is instantiated, for each action, once the plan is generated. However, the *actionActualDuration* property is instantiated, for each action, after

[9] In parenthesis next to each data property we present the build-in data types of each one.

each action has executed since their actual duration is unknown at the time of generating the plan that involves them. The same goes for estimated and actual start and end time points. Regarding *actionStatus*, initially each action is marked as non executed, during execution is marked as pending and at the end of the execution as executed. Regarding the outcome of each action (*actionOutcome*), it is assigned after each action has executed and it is either successful or unsuccessful depending on the outcome. Finally, class *MissionPhase* is intended for keeping track of the phase in which mission execution is (e.g. classification, reacquisition etc.) while class *ProblemPhase* logs the nature of the problem that gave birth to the latest plan that was formed by the planner (e.g. inspection problem etc.).

7.3.4 Dispatching Actions

The utilization of a temporal planning model in general, allows for the dispatch of actions from the executor on time. That is, each action can be dispatched at the time that the planner has assigned to it in the generated plan [152]. However as Cashmore et. al. [152] argue, dispatching actions on time may potentially lead to implications as follows. First, in the event that a set of actions or even a single action in a plan underruns[10] then the system should be able to accommodate a pause-in-state until the estimated duration and the actual one are synchronized. That is, remain in the same state until the estimated and the actual durations match. If this is not possible then the plan will fail. Second, in the event that a set of actions or even a single action in a plan overruns[11] then the plan will again fail since the dispatch on time approach will dispatch the next action prematurely leading to situations where the effects of the previous action may have not already taken place. The approach for dispatching actions in this line of work is dispatch on completion to avoid the implications mentioned above. However, pause-in-state is implemented in the framework since it is required for accommodating changes in high level mission priorities for which new plans need to be formulated and the vehicle needs to pause in its current state to devise them.

7.4 Summary and Discussion

This chapter was concerned with presenting a framework for persistently autonomous (underwater) operations. A list of contributions made in this chapter is as follows:

- A framework for persistently autonomous underwater operations based on KnowRob and the development of a set of ontologies and reasoning modules in order to accommodate persistently autonomous underwater operations effectively extending KnowRob to the underwater domain.

- Extension of the approach followed in KnowRob for modelling components with the

[10] An action has underran if its estimated duration is higher that its actual one when executed.
[11] An action has overran if its estimated duration is lower that its actual one when executed.

inclusion of modelling their health status so that we could model component faults at runtime.

- Introduction of the notion of semantic equivalence in capabilities and actions allowing for adaptive execution over taking action on the planning level where possible.

Let us now elaborate on the above list. Architecturally, the framework is divided into three main interacting parts: i) knowledge representation, ii) reasoning and iii) planning and execution.

Regarding knowledge representation and reasoning, this is based on KnowRob which utilizes ontologies and Prolog respectively. However, given that KnowRob was developed in the scope of household robots, we have developed our own set of ontologies and reasoning modules in order to accommodate persistently autonomous (underwater) operations extending in this manner KnowRob to the underwater domain. More specifically, the framework comprises a world ontology for modelling the environment in which an AUV performs autonomous operations. Regarding the modelling of AUVs and their internal state, the framework comprises a components, a capabilities, an actions and a vehicle ontologies. These ontologies are interconnected in a bottom up fashion with the components ontology lying at the very basis, followed by the capabilities ontology at a higher modelling level, followed by the actions ontology and finally by the vehicle ontology. This bottom up approach is based on the bottom up approach of KnowRob. However, we have enriched the modelling of components by modelling their health status which indicates whether components are functional or non functional. In this manner, we are able to model component faults at runtime. As such, a vehicle may start with a set of capabilities given the existence and the health status of components but this (set) can change in the presence of component faults. By extension, the initial set of actions that the vehicle can perform will be affected given a reduced set of capabilities. Another, modelling extension we have implemented is the existence of semantically equivalent capabilities in the presence of redundant components. Semantically equivalent capabilities represent alternative ways of achieving the same type of capability. The notion of semantic equivalence is also extended to actions in the presence of semantically equivalent capabilities for the same type of action. This assessment of the vehicle internal state is dynamic and in real time as the vehicle operates and is made possible with reasoning modules implemented in Prolog.

Our framework is also complemented by a mission planning and execution system which is based on the PANDORA project. As in the case of PANDORA, the system comprises one PDDL planner and one executor. However as opposed to PANDORA, we divide planning and execution information into two distinct ontologies, i.e. a planning and an execution ontology who exchange information. This distinction makes the system modular and more flexible especially in cases where different planning approaches need to be implemented by other users. In addition, we do not only encode planning problems inside the planning ontology but we additionally encode the planning domain which along with planning problems are dynamically reconstructed and fed into the PDDL planner to

generate mission plans. In this manner the complete information available to planner is made accessible to the user through querying the planning ontology. Further, we consider changes in high level mission priorities during mission execution which the framework can accommodate by taking action on the planning level (adaptive planning). The framework is also designed to take action on the planning level in the event that component faults appear in the vehicle. However, the framework favours adaptive execution over adaptive planning where possible. That is, in cases where no critical component has deprived the vehicle of a capability type that is in turn critical for an action type. If this is the case, the vehicle has to devise alternative plans (adaptive planning) to satisfy mission goals to the maximum extent possible.

Chapter 8

Persistently Autonomous Mine Countermeasures

Underwater Mine Countermeasures (MCM) lie within the spectrum of underwater operations and deal with the identification of mined areas to be avoided as well as localization and neutralization of individual mines [180]. The purpose of MCM operations within the Unmanned Underwater Vehicles (UUVs) context can be summarized into the following: "to field a common set of unmanned, modular MCM systems operated from a variety of platforms or shore sites that can quickly counter the spectrum of threat mines assuring access to naval forces with minimum mine risk" [181]. MCM can be broken down into the following four phases: i) Detect (D), ii) Classify (C), iii) Identify (I) and iv) Neutralize (N) [181]. These phases can be either performed by one or more vehicles in one or more passes. For instance, in the case of one vehicle and two passes, the approach to MCM could be that in the first pass the vehicle performs Detection (D) and Classification (C) of Mine-Like Objects (MLOs) using appropriate sensors (e.g. side-looking sonar). In the second pass, the vehicle performs Identification (I) of mines using electro-optic sensors (e.g. cameras) and neutralization (N) of those deemed to be mines using some neutraliser (e.g. a stationary bomblet that is placed in the area and is remotely detonated later using an acoustic command [181]). The aforementioned scenario can be described as DC IN (four phases in two passes). In most cases, depending on the number of vehicles and their capabilities, MCM operations can be realized as different combinations of the four phases. An alternative approach to MCM, which we follow in this chapter, is to perform Detection (D), Classification (C), Reacquisition (R) and Inspection (In) of mines maintaining however the four-phase, two-pass approach. That is, DC RIn.

As was mentioned in the introduction of this thesis (see Section 1.1.1), persistently autonomous vehicles are the key to addressing a series of open challenges in autonomous underwater operations. These include vehicle failures, partially known and dynamic environments, and minimization of human in the loop occurrences. Such challenges are also dominant in the underwater MCM environment. In addition, the impact of misaddressing them can be highly costly; not only with respect to funds (e.g. losing a vehicle) but also

with respect to human lives due to the highly dangerous nature of mines. Consequently, when using AUVs for autonomous underwater MCM operations, it is of utmost importance that these vehicles demonstrate persistence in their autonomy [179].

8.1 Extending and Adjusting Framework Ontologies for MCM

The purpose of this section is to document all the necessary adjustments and extensions performed to the ontologies of the persistently autonomous operations framework that was presented in Chapter 7 so that they fit the MCM setting. By no means does this section substitute the analysis made in Chapter 7 to which the reader should refer in the event that he/she wants to recall how the framework ontologies are organised and how things bind together forming the bigger picture. Where we feel that the extensions and the adjustments are not straightforward, we provide additional information.

8.1.1 MCM World Ontology

The MCM world ontology models the environment in which the AUV platform will operate executing MCM missions. In comparison to the generic world ontology illustrated in Figure 7.2, the MCM world ontology contains two additional classes, *Mlo* and *Mine*, as shown in Figure 8.1. As the AUV proceeds with searching for mines in an area, the *Circle* class

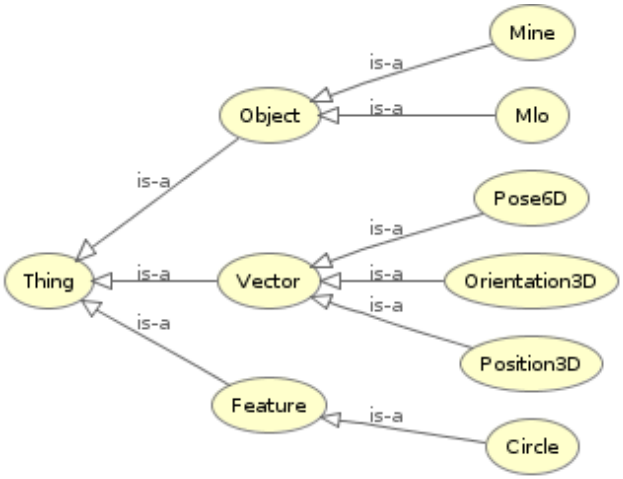


Figure 8.1. MCM world ontology class hierarchy.

is used for instantiating circle feature detections based on which the classification system will create and assert *Mlo* (Mine-Like Object) instances into the ontology. Such assertions associated with some probability and entropy. The *Mine* class on the other hand is intended for instantiating mines based on *Mlo* instances that are deemed to be mines after the AUV has reacquired them.

8.1.2 MCM Components, Capabilities, Actions and Vehicle Ontologies

Regarding the components ontology for MCM, it remains unchanged compared to the components ontology illustrated in Figure 7.4. That is, as far as hardware components are concerned. In order to extend it to the MCM setting though we have extended the *DetectionModule*, *ClassificationModule* and *ReacquisitionModule* classes (see Figure 7.4 of Section 7.2.2.1) with *MineClassificationModule*, *MineDetectionModule* and *MloReacquisitionModule* classes respectively (see Figure 8.2). This was done in order to be able to

Figure 8.2. MCM components ontology class hierarchy.

instantiate the necessary detectors, classifiers as well as MLO reacquisition modules that the AUV is equipped with and are required for the MCM setting.

With respect to the capabilities ontology for MCM, we have extended the *DetectionCapability*, *ClassificationCapability* and *ReacquisitionCapability* classes shown in Figure 7.5 with *MineDetectionCapability*, *MineClassificationCapability* and *MloReacquisitionCapability* classes respectively. In addition, as in the case of the capabilities ontology shown in Figure 7.5, we have extended the aforementioned MCM related capability classes with additional classes to represent *semantically equivalent* capabilities in the MCM setting. The outcome of the extended MCM capabilities ontology is shown in Figure 8.3. This extension of the capabilities ontology is necessary given that we want to represent capabilities within an MCM setting and not some generic autonomous operations scenarios. At this point we would like to emphasize that the dependency relations shown in Table 7.1 are also extended accordingly. For instance, the primary mine detection capability of the vehicle now depends on the forward looking sonar and the mine detection module while the secondary mine detection capability on the forward looking camera and the mine detection module and so on.

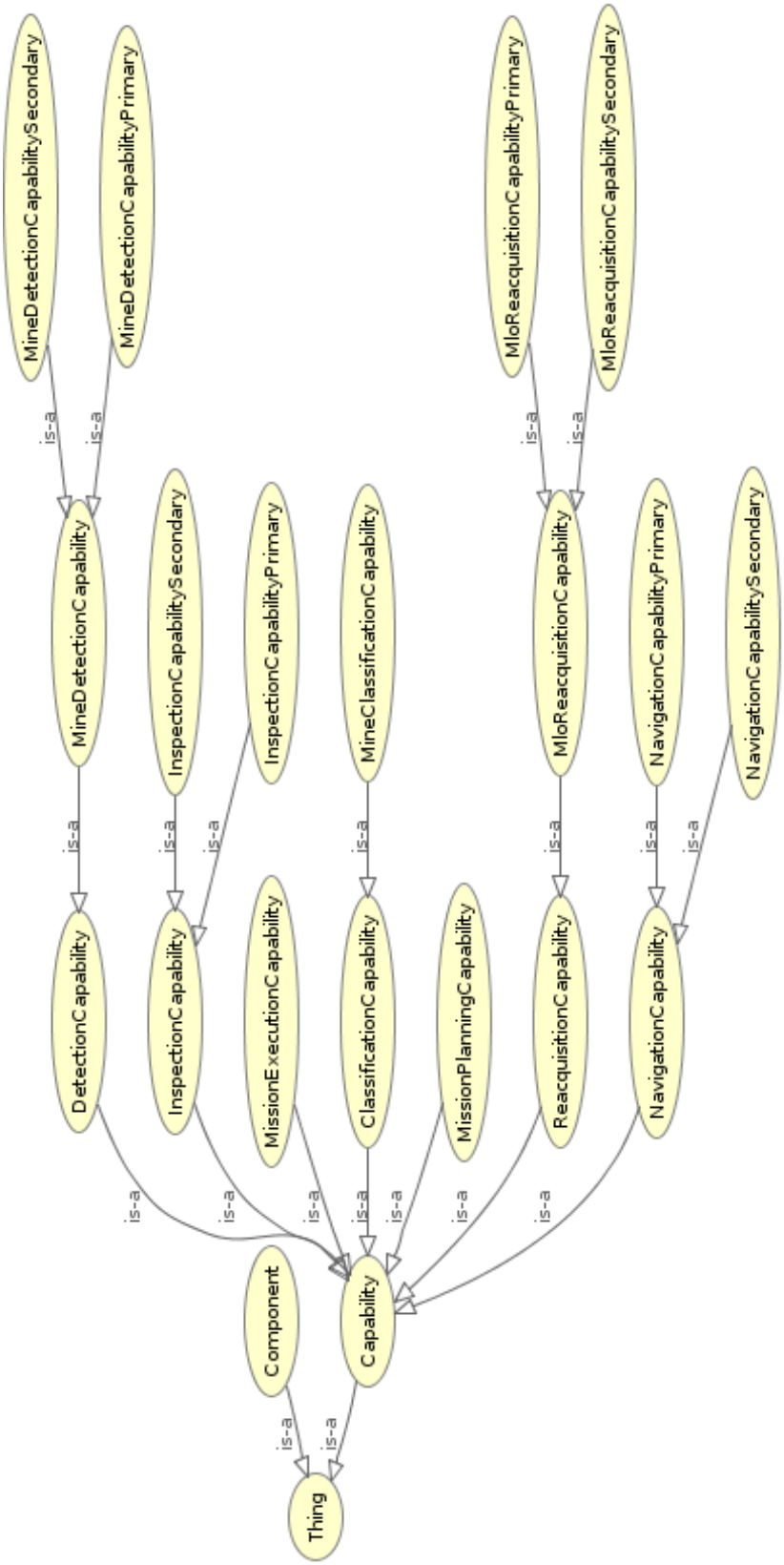


Figure 8.3. MCM capabilities ontology class hierarchy. We have intentionally omitted the extended MCM components ontology for better readability.

Similarly, with respect to the actions ontology for MCM, we have extended the actions *DoHoverDetection*, *DoClassify*, *DoReacquire* and *Doinspect* action classes shown in Figure 7.6 with *DoHoverMineDetection*, *DoClassifyMine*, *DoReacquireMlo* and *DoInspectMine* classes respectively. Moreover, as in the case of the actions ontology shown in Figure 7.6 we have extended the aforementioned MCM related action classes with additional classes to represent *semantically equivalent* actions in the MCM setting. Figure 8.4

illustrates the outcome of this extension. Again, similar to the case of capabilities, the de-

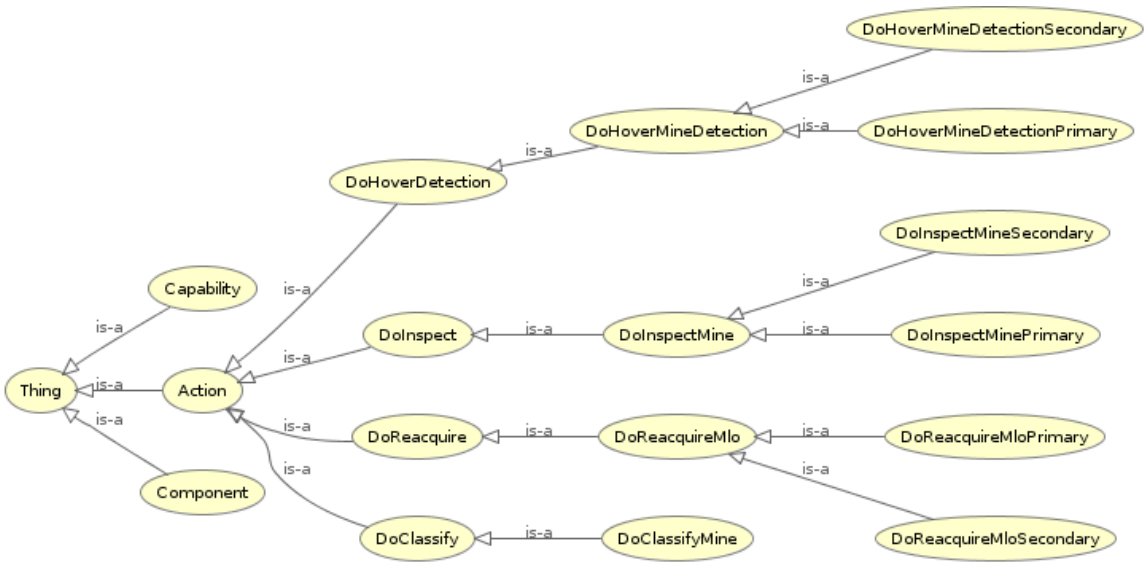


Figure 8.4. MCM actions ontology class hierarchy. We have intentionally omitted the extended MCM components and capabilities ontologies for better readability.

pendency relations shown in Table 7.2 are extended accordingly. For instance, the primary MLO reacquisition action now depends on the primary MLO reacquisition capability and on the primary navigation capability while the mine classification capability now depends on the mine classification capability and so on.

Finally, regarding the vehicle ontology, we modified it by importing the extended components, capabilities and actions ontologies as opposed to the standard ones. In addition, we updated the *hasComponent*, *hasCapability* and *hasAction* class level relations within the ontology so that we can accurately represent what is expected of the vehicle to have at its disposal. This was done so that the Prolog vehicle system identification and system update phases will be able to build, monitor and update the vehicle internal state correctly during MCM operations.

8.1.3 MCM Planning and Execution ontologies

In order to be able to model the MCM planning domain and problems as well as log and monitor the planning and execution outcomes as part of an adaptive mission planning and execution setting we have also adjusted the mission planning and execution ontologies. Section 8.1.3.1 is concerned with the MCM planning ontology while Section 8.1.3.2 with the MCM execution ontology.

8.1.3.1 MCM Planning Ontology

In comparison to the planning ontology shown in Figure 7.14, the MCM mission planning ontology shown in Figure 8.5 features some changes to the class names, that were referring to autonomous operations in general, to reflect the MCM setting.

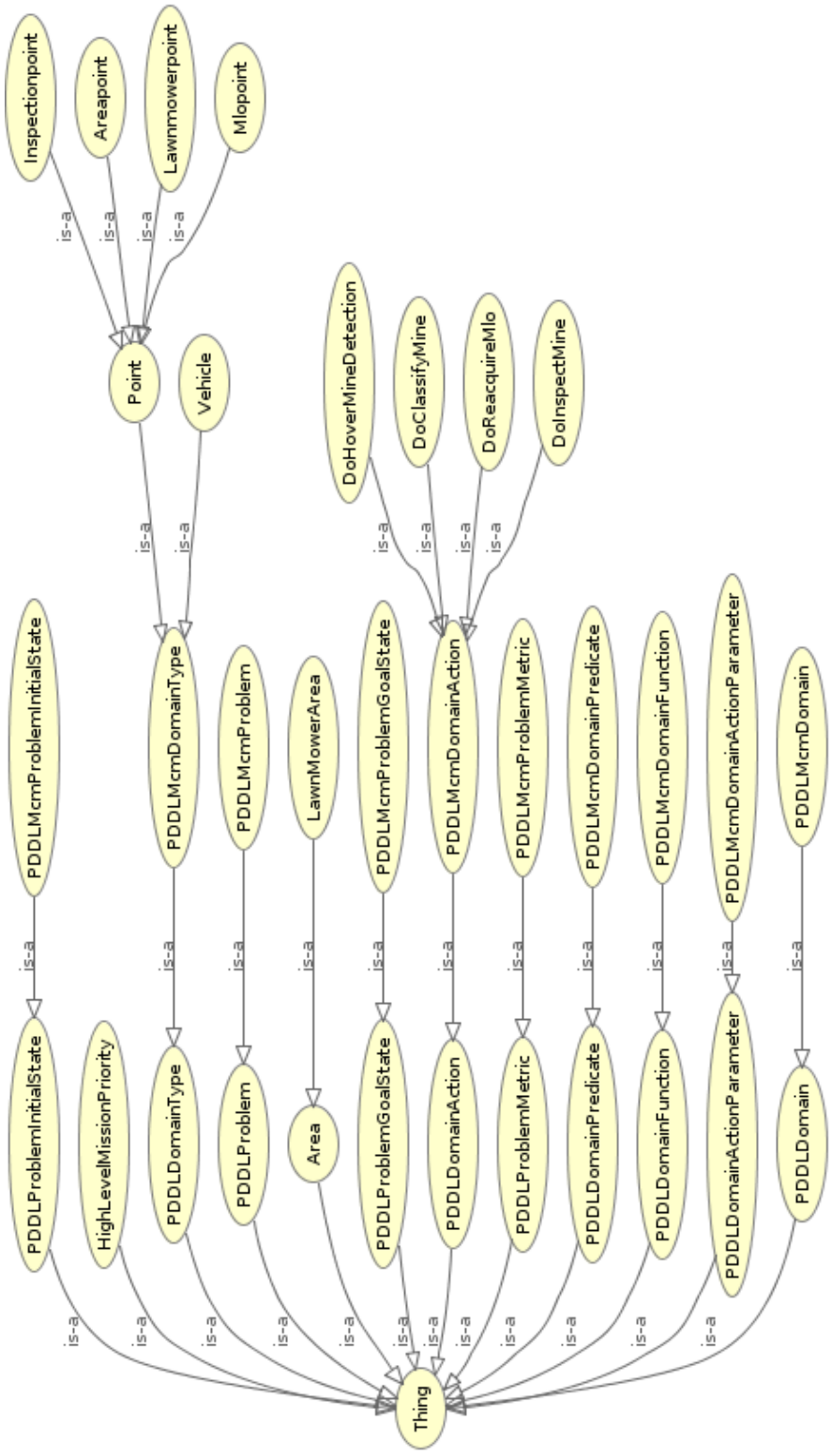


Figure 8.5. MCM planning ontology class hierarchy. We have intentionally omitted the extended MCM world, components, capabilities, actions and vehicle ontologies for better readability.

No core change has been made to the class hierarchy and no change at all has been made to the nature of ontology properties. What changed though is the domain and problem definitions inside the planning ontology to support the syntax and semantics of temporal MCM domains and problems. For instance, in defining the DoReacquireMlo action we create an instance of the *DoReacquireMlo* class and relate it to its parameters, duration, precondition and effect via *parameter*, *duration*, *precondition* and *effect* string data properties as in the case of the DoReacquire action reconstructed from the autonomous operations planning ontology. However, the parameters are no longer a vehicle and two objectpoints but a vehicle

and two mine-like object points (Mlopoints). The same goes for the duration, precondition and effect of the action and all actions as well as everything included within the MCM domain and problems (e.g. predicates, functions, initial and goal states etc.) which we adjust accordingly.

Recall form Section 7.3.2 that the planning ontology, except for defining PDDL domains and problems, is used for reconstructing them so that they become available for the planner to generate mission plans. Consequently, this also applies to the MCM planning ontology. Snippet 25 illustrates the PDDL MCM domain types, predicates and functions in PDDL syntax as they are reconstructed form the MCM planning ontology. When compared to the types, predicates and functions of the autonomous operations planning ontology the similarity is very obvious. In fact, the only differences are that: i) instead of objectpoints we now have mlopoints which are waypoints that a vehicle visits and reacquires mine-like objects and ii) instead of predicates and functions related to objects and objectpoints we now have predicates and functions in which the subjects are mine-like objects and mlopoints as well as mines. Moverover, inspectionpoints now represent points around mines that the vehicle visits in order to inspect the mines. Let us now proceed with the temporal MCM domain actions reconstructed from the MCM planning ontology. As one might expect, temporal MCM domain durative actions are variations of the autonomous operations temporal domain durative actions that were presented in snippets 14, 15, 16 and 17 and can be found in Section 7.3.1.1. To avoid unnecessary repetition we only present the reconstructed *do_reacquire_mlo* action shown in snippet 26. The remaining temporal MCM domain durative actions, i.e. *do_hover_mine_detection*, *do_classify_mine* and *do_inspect_mine*, can be obtained by substituting the autonomous operations temporal domain types, predicates and functions inside the *do_hover_detection*, *do_classify* and *do_inspect* actions with the temporal MCM domain types, predicates and functions.

Durative action `do_reacquire_mlo` is intended for plans where the goal is to reacquire mine-like objects in the environment and has three parameters: a vehicle `v` and two mlopoints `from` and `to`. The estimated duration of the action is identical to the one presented for the `do_reacquire` action (see Snippet 16) but instead of adding the time that the vehicle requires to reacquire an object in general we specialise this as the time time the vehicle requires to reacquire a mine-like object. Again, he value for the time that the vehicle needs to reacquire a mine-like object can be chosen empirically or learned from data. For the action to be applicable, at the start of the action's interval (the point at which the action is applied), the vehicle needs to be at mlopoint (location) `from`, mlopoint `to` must be reachable from mlopoint `from` and mlopoint `to` must have not been visited. Moreover, at the start of the action's interval, the mine-like object must have not been reacquired from the mlopoint `op`. As a final precondition the vehicle must have energy greater or equal ($>=$) than the following: the distance between mlopoints `from,to` multiplied by the energy consumption rate of the vehicle while moving plus the rate at which the vehicle consumes energy while maintaining its position multiplied by the time the vehicle needs to reacquire a mine-like object. The effect of executing the `do_reacquire_mlo` action is that the vehicle has left its

Snippet 25 Temporal PDDL MCM domain types, predicates and functions reconstructed from the MCM planing ontology.

```
(:types Lawnmowerpoint Mlopoint Inspectionpoint Vehicle)
(:predicates
        (at_mlop ?v - Vehicle ?mlop - Mlopoint)
        (reachable_mlop ?mlop1 ?mlop2 - Mlopoint)
        (at_ip ?v - Vehicle ?ip - Inspectionpoint)
        (reachable_ip ?ip1 ?ip2 - Inspectionpoint)
        (at_lp ?v - Vehicle ?lp - Lawnmowerpoint)
        (reachable_lp ?lp1 ?lp2 - Lawnmowerpoint)
)
(:functions
        (visited_lp ?lp - Lawnmowerpoint)
        (visited_mlop ?mlop - Mlopoint)
        (visited_ip ?ip - Inspectionpoint)
        (distance_lp ?lp1 ?lp2 - Lawnmowerpoint)
        (distance_mlop ?mlop1 ?mlop2 - Mlopoint)
        (distance_ip ?ip1 ?ip2 - Inspectionpoint)
        (mine_detection_complete ?lp - Lawnmowerpoint)
        (mine_classification_complete ?v - Vehicle)
        (reacquired_mlo ?mlop - Mlopoint)
        (inspected_mine ?ip - Inspectionpoint)
        (cnt_reacquired_mlo ?v - Vehicle)
        (cnt_visited_lp ?v - Vehicle)
        (remaining_energy ?v - Vehicle)
        (consumed_energy ?v - Vehicle)
        (energy_consumption_rate_moving ?v - Vehicle)
        (energy_consumption_rate_still ?v - Vehicle)
        (prob_mlo ?mlop - Mlopoint)
        (ent_mlo ?mlop - Mlopoint)
        (prob_mlo_quotient_sum ?v - Vehicle)
        (ent_mlo_quotient_sum ?v - Vehicle)
        (det_obj_time ?v - Vehicle)
        (class_mlo_time ?v - Vehicle)
        (reacq_mlo_time ?v - Vehicle)
        (insp_mine_time ?v - Vehicle)
        (mult_fact_time ?v - Vehicle)
        (mission_duration)

)
```

Snippet 26 PDDL temporal MCM domain do_reacquire_mlo durative action reconstructed from the MCM planning ontology.

```
(:durative-action do_reacquire_mlo
 :parameters (?v - Vehicle ?from ?to - Mlopoint)
 :duration (= ?duration (+(*(distance_mlop ?from ?to)
 (mult_fact_time ?v)) (reacq_mlo_time ?v)))
 :condition (
                 and (at start (at_mlop ?v ?from))
                 (at start (reachable_mlop ?from ?to))
                 (at start (= (visited_mlop ?to) 0))
                 (at start (= (reacquired_mlo ?to) 0))
                 (at start (>= (remaining_energy ?v) (+(* (distance_mlop ?from ?to)
                 (energy_consumption_rate_moving ?v)) (*(reacq_mlo_time ?v)
                 (energy_consumption_rate_still ?v)))))
)

 :effect (
           and (at start (not (at_mlop ?v ?from)))
           (at end (at_op ?v ?to))
           (at end (decrease (remaining_energy ?v) (+(* (distance_mlop ?from ?to)
           (energy_consumption_rate_moving ?v)) (*(reacq_mlo_time ?v)
           (energy_consumption_rate_still ?v)))))
           (at end (increase (consumed_energy ?v) (+(* (distance_mlop ?from ?to)
           (energy_consumption_rate_moving ?v)) (*(reacq_mlo_time ?v)
           (energy_consumption_rate_still ?v)))))
           (at end (increase (visited_mlop ?to) 1))
           (at end (increase (reacquired_mlo ?to) 1))
           (at end (increase (cnt_reacquired_mlo ?v) 1))
           (at end (increase (prob_mlo_quotient_sum ?v) (/(prob_mlo ?to)
           (prob_mlo ?from))))
           (at end (increase (ent_mlo_quotient_sum ?v) (/(ent_mlo ?to)
           (ent_mlo ?from))))
           (at end (increase (mission_duration) ?duration))
         )
)
```

initial location at the start of the action ((at start (not (at_mlop ?v ?from))) and moved to the location of mlopoint to. Additionally, at the end of the action's interval, the vehicle's remaining and consumed energy will change according to what we have presented above as being the energy the vehicle requires to perform the action. Furthermore, at the end of the action's interval, the effect of the mlopoint to being marked as visited will take place, the effect of the object as being reacquired will take place and the number of reacquired objects will increase by one. Also, at end (increase (prob_mlo_quotient_sum ?v) (/(prob_mlo ?to) (prob_mlo ?from))) increases, at the end of the action's interval, the sum of probability quotients by the quotient of dividing the probability of the mine-like object being at location (mlopoint) to being a mine by the probability of the mine-like object at location (mlopoint) from being a mine. Finally, at end (increase (ent_mlo_quotient_sum ?v) (/(ent_mlo ?to) (ent_mlo ?from))) achieves the same outcome but with entropy instead of probability.

Regarding the reconstructed MCM problems from the planning ontology, they are variations of the detection, classification, reacquisition and inspection problems that were presented in snippets 18-24 and can be found in Section 7.3.1.3. As such, and to avoid unnecessary repetition, we only present an exemplar reconstructed PDDL *MLO reacquisition problem* (see snippet 27). The remaining MCM problems, i.e. *detection*, *classification* and *inspection* of mines, can be obtained by substituting the autonomous operations objects, predicates and functions inside snippets 18-24 with the MCM objects, predicates and functions given the area that needs surveying, the mine-like objects found in the environment and the desired MCM high level mission priorities. Now, according to the exemplar reacquisition problem in snippet 27, the planner must generate a plan in which the AUV will reacquire two MLOs (located at their respective mlopoints) without spending more energy than it has available (goal state). The initial state indicates that the AUV starts at mlop3, the MLO at this location has been reacquired and the distances between mlopoints (MLOs) are known. Associated probabilities, entropy, remaining energy as well as consumed energy are also known. Furthermore, the energy consumption rates for the vehicle while moving and while maintaining its position are also provided as is the multiplication factor for time and the time the vehicle requires to reacquire a mine-like object. Again, as we stated before, the energy consumption rates as well as the multiplication factor for time and the reacquisition time need to be chosen empirically or learned from data. Finally, the metric maximize (remaining_energy auv) shown in Snippet 27 indicates that the high level priority of the mission is for the AUV to reacquire the MLOs located at mlop1 and mlop2 in a way that the remaining energy on the vehicle is maximized, i.e. the shortest possible distance is travelled.

8.1.3.2 MCM Execution Ontology

As in the case of the MCM planning ontology the MCM execution ontology features some class name changes when compared to the autonomous operations execution ontology pre-

Snippet 27 Exemplar PDDL MLO reacquisition problem reconstructed from the planning ontology. Energy efficiency is the chosen high level mission priority.

```
(:objects
        mlop1 mlop2 mlop3 - Mlopoint
        auv - Vehicle
)
(:init
        (at_mlop auv mlop3)
        (= (visited_mlop mlop1) 0)
        (= (visited_mlop mlop2) 0)
        (= (visited_mlop mlop3) 1)
        (= (reacquired_mlo mlop1) 0)
        (= (reacquired_mlo mlop2) 0)
        (reachable_mlop mlop3 mlop1)
        (reachable_mlop mlop3 mlop2)
        (reachable_mlop mlop1 mlop2)
        (reachable_mlop mlop2 mlop1)
        (= (distance_mlop mlop3 mlop1) 6.05)
        (= (distance_mlop mlop3 mlop2) 4.32)
        (= (distance_mlop mlop1 mlop2) 7.19)
        (= (distance_mlop mlop2 mlop1) 7.19)
        (= (prob_mlo mlop1) 0.606)
        (= (ent_mlo mlop1) 0.967)
        (= (prob_mlo mlop2) 0.755)
        (= (ent_mlo mlop2) 0.803)
        (= (prob_mlo mlop3) 1000.0)
        (= (ent_mlo mlop3) 1000.0)
        (= (cnt_reacquired_mlo auv) 0)
        (= (prob_mlo_quotient_sum auv) 0)
        (= (ent_mlo_quotient_sum auv) 0)
        (= (remaining_energy auv) 30000)
        (= (consumed_energy auv) 0)
        (= (mission-duration) 0)
        (= (energy_consumption_rate_moving auv) 3)
        (= (energy_consumption_rate_still auv) 1.5)
        (= (mult_fact_time nessie1) 10)
        (= (reacq_mlo_time nessie1) 2)
        (= (mission_duration) 0)

)
(:goal
        (and
        (>= (remaining_energy auv) 0)
        (= (cnt_reacquired_mlo auv) 2)
        )
)
(:metric minimize (consumed_energy auv))
```

sented in Section 7.3.3. Class name changes were performed in order to adjust the autonomous operations execution ontology to the MCM setting. For the same reason, the MCM execution ontology also features some class extensions.

More specifically, as can be observed by looking at Figure 8.6, we now have the *McmMissionPlan* class and its various subclasses adjusted to the MCM setting instead of the *AutOpMissionPlan* class and its respective subclasses that are shown in Figure 7.15. Moreover

Figure 8.6. MCM execution ontology class hierarchy.

the MCM execution ontology now features a *McmMissionAction* class instead of an *AutOpMissionAction* class which in turn is extended by subclasses that are intended for holding information related to the actions executed during MCM missions. In fact, the information that is being logged is the same as in the case of the autonomous operations execution ontology (see Section 7.3.3). Having said that, we would like to emphasize that both the object and the data properties that were presented in Section 7.3.3 remain unchanged, not only with respect to the MCM execution ontology action classes but with all classes. They (the properties) are just applied among instances that originate from the adjusted and extended classes. Moving on with the changes, classes *ProblemPhase* and *MissionPhase* are extended with *McmProblemPhase* and *McmMissionPhase* classes respectively. Finally with respect to mission points, class *MissionPoint* is extended with a *McmMissionPoint* class which is in turn is extended with all necessary point classes for MCM missions.

8.2 High Level MCM Mission Priorities

Different high level MCM mission priorities can be taken into consideration for the reacquisition of mines, and may be affected by changes in the situation on the surface. For example, a ship may be en-route to the area where MCM is conducted, and probability-efficient mine clearance given the AUVs remaining energy may be required. Recall from Section 7.3.1.2 that our framework supports three high level mission priorities: i) energy-efficiency ii) a combination of energy-efficiency and probability-efficiency iii) a combination of energy-efficiency and entropy-efficiency. The aforementioned high level mission priorities are also applicable in the MCM setting with the objects of interest now being mine-like objects and mines. As such, no adjustment needs to be made to the framework to accommodate the MCM setting. For example, in the case of a combination of energy-efficient and probability-efficient reacquisition in the MCM setting the planner will attempt to generate a plan in which MLOs that are the most likely to be mines will be reacquired and inspected first while concurrently the vehicle's consumed energy is minimized. This effectively formulates a multi-objective optimization problem whose solution is already described in Section 7.3.1.2.

In the event that the high level mission priority is energy efficiency, reacquisition is an Open Loop Symmetric Travelling Salesman Problem (OSTSP) [4]. Openness is due to the vehicle not being required to return to its initial position while symmetry is due to the assumption of ours that for travelling from some mine-like object point MLO_i to some other mine-like object point MLO_j the vehicle requires the same amount of energy as for travelling from MLO_j to MLO_i. That is, due to the energy consumption being proportionate to the distance travelled and the fact that travelling from MLO_i to MLO_j is the same distance as travelling from MLO_j to MLO_i. In contrast, for the remaining high level mission priorities, the reacquisition problem is an Open Loop Travelling Salesman Problem (OTSP) [182]. This is due to the manner in which transitions from one mine-like object to the other are evaluated when probability or entropy are involved[1]. For instance, let us assume that MLO_i has a 0.69 probability of being a mine while MLO_j a 0.93 probability. In this event, transitioning from MLO_i to MLO_j will yield $\frac{0.93}{0.69} = 1.348$ while transitioning from MLO_j to MLO_i will yield $\frac{0.69}{0.93} = 0.74$.

8.3 Underwater Simulator

8.3.1 Simulator Overview

For the purpose of testing our framework in persistently autonomous underwater MCM operations we used the UWSim underwater simulator [183]. UWSim is an open source tool that was developed by the IRSLab which is part of Jaume-I University in Castellon. UWSim offers a flexible, extensible and easy to understand simulator environment in which

[1] See the effect of the *do_reacquire_mlo* action in snippet 26.

researchers can simulate and test their work regarding not only underwater but also surface[2] vehicles and missions.

One of the main features of the simulator is that we can represent any underwater environment with an XML file comprising all the necessary elements to do so. At the very basis of the XML lies the inclusion of a 3D scene that represents the core environment and needs to be in a format that can be read by OpenSceneGraph (OSG) [184] in order to be rendered by the simulator. Depending on the environment that we are interested in, we can add additional objects to refine it. For instance, we can have a 3D scene that represents a shipwreck of archaeological interest and later on add in the XML file references to archaeological artifacts such us amphorae in various sizes, shapes and positions which must be in a format that can be read by OSG. In addition, features like water visibility and water color can be chosen and configured at will.

Another main feature of the simulator is that it can support multiple vehicles as well as manipulators via abstract classes that can be specialized accordingly. We focus on vehicles since our work is not related to manipulation. Each vehicle consists of two things: i) its 3D model and ii) its description. Regarding its 3D model, the supported formats are the same as the formats for 3D scenes. With respect to its description, this is written in the Unified Robot Description Format (URDF) which is an XML format for describing robots. This, among others, includes the kinematic and dynamic characteristics of the vehicle. The simulator also offers a series of sensors a fraction of which is listed below [185]:

- **Camera & Range Camera:** The camera produces a continuous stream of images and can be placed in any desirable angle while the range camera does the same but with range images. That is, it produces images that show the distance to specific points in a scene from another specific point (depth images).

- **Range Sensor & Multibeam Range Sensor:** The range sensor yields a distance measurement to the nearest object as long as the object is within the direction that it points and within the range of the sensor. It's multibeam alternative is an array of range sensors that yields distance measurements at specific angle increments.

- **Pressure Sensor:** The pressure sensor yields a pressure measurement that represents the pressure a vehicle receives when submerged.

- **Doppler Velocity Log:** The Doppler Velocity Log (DVL) estimates the linear speed at which the vehicle is moving.

- **Inertial Measurement Unit:** The Inertial Measurement Unit (IMU) estimates the orientation of the vehicle using the world frame as reference.

- **Global Positioning System:** The Global Positioning System (GPS) calculates the coordinates of the vehicle with respect to the world frame. It is important to mention

[2]By surface, we mean the surface of the sea.

that in order to get a GPS position measurement the vehicle needs to be very close to the surface. This is done in order to simulate an actual GPS which would not be able to get satellite signals when submerged.

- **Force Torque Sensor:** The force torque sensor provides an estimate of the force and torque applied/received at the area where it is installed.

Further to the listed sensors, the simulator is equipped with some default position and velocity sensors for vehicles that provide six Degrees Of Freedom (6DOF) poses. That is, x, y, z or equivalently north, east, depth and r, p, y for roll, pitch and yaw. Yet another feature of UWSim is that it integrates the Bullet physics engine [186] for simulating physics.

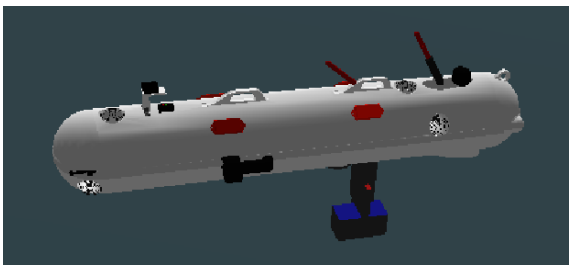
All the aforementioned features are made accessible externally via ROS since UWSim is distributed as a ROS package. This, for instance, gives us the capability to issue commands to the vehicle to move or develop our own vision algorithms based on which the vehicle will detect something or write our own controllers and so on. Finally, the simulator provides the capability of developing various widgets. Widgets are portions of the simulated environment display that can be used to show important information to the user. For instance, one can develop a camera widget which will display what the simulated camera sees or a navigation widget that will display navigation related information to the user.

8.4 Experimental Setup

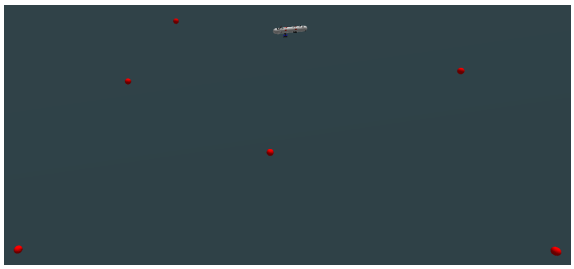
Experiments were conducted in a simulated MCM environment inside UWSim. In order to build the simulated MCM environment we first created 3D scene representing the ocean in which the experiments would take place. We then imported the scene in an MCM XML file and parametrized it accordingly with respect to water visibility, color etc. Dealing with sea currents during MCM is outside the scope of this thesis and as such we set the sea currents parameter at zero. In order to model the mines in our environment we built six 3D spherical objects which we imported in our MCM XML file and placed them randomly within a 22×12 meter area. The vehicle that we chose to use in our MCM operations is a simulated version of the Nessie V AUV [187]. Abiding to the requirements of UWSim, we created the 3D model of the vehicle and then formulated its description in URDF in which we included its dynamic and kinematic characteristics. Finally, we imported the vehicle into the MCM XML, thus completing the representation of the MCM simulated environment. Figure 8.7 illustrates the simulated MCM environment. Regarding vehicle sensors, Nessie V is equipped with all the sensors required for operating autonomously and for performing MCM operations, e.g. (a forward looking sonar, cameras, a DVL, a gyroscope etc.).

Before proceeding with the experiments we would like to discuss some choices with respect to the following functions:

- `mult_fact_time...`
- `det_obj_time...`,`class_mlo_time...`,`reacq_mlo_time...`,`insp_mine_time...`



(a) Simulated Nessie V. Attached underneath the vehicle is a forward looking sonar.



(b) Experimental setup illustrating simulated Nessie V and six mines (red spherical objects).

Figure 8.7. Simulated MCM environment.

- `energy_consumption_rate_moving...`, `energy_consumption_rate_still...`

that are used as part of temporal domain durative action definitions. Regarding detection actions and detection problems we will be using a value of 2.5 for the (per meter) multiplication factor for time. In addition we will be using a value of zero seconds as the detection time of objects as we explained in section 7.3.1.1. The choice of 2.5 is made empirically in the scope of formulating an upper bound for the estimated duration of detection actions. Regarding classification actions and problems we will be using a value of 25 seconds for `class_mlo_time...` which is again chosen empirically so that an upper bound for the estimated duration of classification actions will be formulated. Regarding reacquisition actions and problems the multiplication factor for time is chosen empirically to be again 2.5 while the reacquisition time for an object is chosen to be 2 seconds in the same scope as in the case of detection actions and problems. Now, in the case of inspection actions and problems the empirically chosen value for the time multiplication factor is 10 while the value of the inspection time for each object from an inspectionpoint is chosen to be 1 second in the same scope as in the case of detection actions and problems as well as reacquisition actions and problems. As far as energy consumption rates are concerned, the values chosen are 3 and 1.5 for an energy consumption rate of the vehicle whilst moving and an energy consumption rate for the vehicle whilst maintaining its position respectively. As opposed to the empirically chosen values, the values for the energy consumption rates were not chosen empirically. They were chosen in the scope of reflecting that the vehicle will consume more energy whilst moving compared to whilst maintaining its position. This non empirical choice is due to the fact that we did not have access to data of Nessie's energy consumption profile and we proceeded with making an assumption. These numbers are expected to be different for different vehicles. Also, note that multiplication factors for time are also dependent on the vehicle used. However, an observant reader may have noticed that the time multiplication factor value for inspection actions is different than the value chosen for detection actions. This is due to the fact that while searching to detect objects in a lawnmower pattern a vehicle tends to perform less accelerations and decelerations as well as re-orienteering as it tends to traverse longer distances in straight lines. As such, the (per meter) multiplication factor for time is going to be smaller when compared to the (per meter) multiplication factor for time that applies to inspection actions which tend to be over

much smaller distances and at lower speeds. Finally, note that in formulating upper bounds for action durations we allow each action to underrun and not overrun. Had we chosen lower bounds and consequently allowing for actions to overrun, it would lead to some plans potentially failing in the presence of time deadlines. However, this choice of ours has a different implication. That is, in the presence of time deadlines some problem instances may become unsolvable where in fact they may have been solvable in reality. Having said that, in this thesis we do not focus on time deadlines and as such this discussion is at this point purely made in the scope of understanding the choices that we made and the rationale behind them.

8.5 Energy-Efficient MCM

The energy-efficient MCM scenario that is investigated in this section is our core scenario and it involves detection, classification, reacquisition and inspection of mines in two passes using one AUV. The first pass includes the phases of detection and classification of MLOs while the second pass, the phases of reacquisition and inspection. The purpose of this scenario is to demonstrate how semantic knowledge residing inside the ontologies is used in planning and executing the aforementioned MCM mission successfully in an energy-efficient manner.

8.5.1 Results

Each MCM phase corresponds to one problem reconstructed from the planning ontology. As such we have four problems in total. The initial information provided is four lawnmower area points instantiated inside the MCM world ontology along with spacing and overlap information that the framework utilizes in order to generate lawnmower trajectory points both inside the MCM planning ontology and the MCM execution ontology[3]. Figure 8.8a illustrates the outcome of this process schematically. Moreover, the initial problem phase which is detection is encoded inside the MCM execution ontology and the high level mission priority is energy efficiency, since the high level mission priority is to plan and execute MCM in an energy-efficient manner.

First pass - detection: Given the detection problem phase and the generated lawnmowerpoints the framework reconstructs the detection planning problem which along with the reconstruction of the MCM domain are fed into the planner. The planner then devises a plan to traverse the lawnmowerpoints in order to search for mines. Table 8.1 illustrates the devised plan while Table 8.2 illustrates the distances between lawnmowerpoints. Moreover, Table 8.3 illustrates the estimated execution duration, estimated energy consumption, plan generation duration and the distance to be travelled for the detection plan shown in Table 8.1 as well as the number of states evaluated by the planner in formulating the plan. As can be observed by looking at Table 8.1 the actions are situated in time with a reference

[3]For an explanation of this duplicate representation see Section 7.3.3

Table 8.1. Plan for detection of mines. Numbers on the left hand-side represent estimated start time points while numbers on the right hand-side (in brackets) represent estimated durations for each durative action. By estimated time points we refer to the points in time after each plan starts executing with a reference to 0.000 seconds as being the start of the plan execution.

First pass - detection
0.000 (do_hover_mine_detection nessie init lp1) [25.650]
25.650 (do_hover_mine_detection nessie lp1 lp2) [50.000]
75.650 (do_hover_mine_detection nessie lp2 lp3) [10.000]
85.650 (do_hover_mine_detection nessie lp3 lp4) [50.000]
135.650 (do_hover_mine_detection nessie lp4 lp5) [10.000]
145.650 (do_hover_mine_detection nessie lp5 lp6) [50.000]

Table 8.2. Distances between lawnmowerpoints.

Distance (in meters)
init-lp1: 10.26, lp1-lp2: 20, lp2-lp3: 4
lp3-lp4: 20, lp4-lp5: 4, lp5-lp6: 20

to the start time point for the plan as being 0.000 and they have an estimated duration. In addition, as can be seen in Table 8.3, the time it took the planner to generate the plan is 0.02 seconds meaning that the generation was very fast given also that the number of states that the planer evaluated in the process of devising the plan was 13. Note that the result with respect to generation duration is very similar with the results for generating inspection plans as they were presented by Cashmore et. al. in [152]. Now, once generated, the plan is instantiated inside the MCM execution ontology and linked with its generation time, generation duration, date as well as estimated execution duration and estimated start and end time points (see Sections 7.3.3, 8.1.3.2). For each `do_hover_mine_detection` action an instance of the *DoHoverMineDetection* class inside the MCM execution ontology is also created and linked to the plan instance. Each instantiated action is linked to its parameters which are the vehicle instance and the instances of its two lawnmowerpoints. Furthermore, each instantiated action is linked to its execution status, execution outcome, execution duration and execution start and end time points. Initially, each `do_hover_mine_detection` action instance is marked as non executed and it has no execution outcome. In addition, it is linked to the estimated execution duration as well as the execution start and end time. Regarding the actual execution duration as well as the actual start and end time points, they will be assigned after the action has executed since they are unknown at the time of generating the plan. The same applies for the execution outcome. Now, before execution, the framework checks whether these actions can be executed based on the system identification and system update phases described in Sections 7.2.3.1 and 7.2.3.2 respectively. This process is continuous since a component can break at any point during a mission affecting the vehicles available capabilities and actions. In this core MCM scenario of ours we do not simulate any component faults. These are considered in the scenario investigated in Section 8.8. The vehicle is now ready to proceed with the execution of the detection plan. Figure

Table 8.3. Plan statistics for the detection plan shown in Table 8.1.

First pass - detection
Estimated Execution Duration: 195.650 secs
Estimated Energy Consumption: 234.780 units
Plan Generation Duration: 0.02 secs
Distance to Travel: 78.26 meters
Number of Evaluated States: 13

8.8b illustrates how the detection phase unfolds as `do_hover_mine_detection` actions are being executed. As the mission progresses the framework will update each action representation inside the MCM execution ontology with respect to its actual duration as well as start and end time points, its execution status and outcome. In addition, the MCM mission phase now becomes detection. At the same time the vehicle performs mine detections. At the end of the detection phase the framework updates the plan representation inside the execution ontology with respect to the actual start and end time points and duration.

Mine detection is a vital part of an MCM mission since the success of the subsequent classification phase is greatly affected by its quality. Approaches for detecting mines are beyond the scope of this thesis and as such the assumption made is that for a real world application, an efficient Automatic Target Recognition (ATR) module is available to the system. Nevertheless, for the purpose of presenting the usage of our framework in performing MCM, we used a circle detector for detecting circular features based on the Hough transform provided by the OpenCV library [188]. During this detection phase of the first pass, as circles are detected, they are asserted into the MCM world ontology (instances of the *Circle* class) along with their positions and radii. This leads to multiple detections of the same objects (circles) being asserted into the KB with slightly different positions given that Nessie will perform such detections from varying distances. This effectively affects the detection outcomes slightly. Nonetheless this will be dealt with by the classification phase that follows.

First pass - classification: After the lawnmower trajectory points have been traversed and the detection phase has been completed, the vehicle proceeds with the second phase which is the classification of detections. In order to do so, the MCM problem phase is initially updated to classification. This effectively, instructs the framework to reconstruct a classification problem from the MCM planning ontology and along with the MCM domain to generate the simple one action plan. Table 8.4 illustrates the aforementioned plan while Table 8.5 illustrates the estimated execution duration, estimated energy consumption and plan generation duration for the classification (single action) plan shown in Table 8.4 as well as the number of states evaluated by the planner in formulating the plan. Again as can be seen by observing Table 8.4, the single action is situated in time with a reference to the start time point for the plan as being 0.000 and it has an estimated duration. Again, as in the case of the detection plan the generation time for the single action plan is very small. Now, once generated, the classification plan is instantiated inside the MCM execution ontology

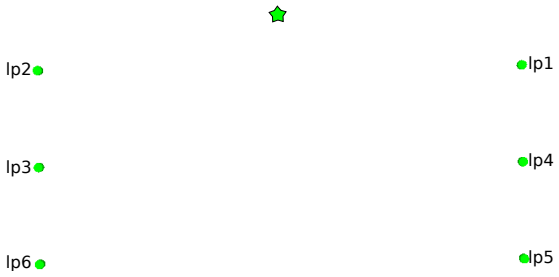
Table 8.4. Plan for classification of mines. The number on the left hand-side represents the estimated start time point while the number on the right hand-side (in brackets) represents the estimated estimated duration for the durative classification action.

First pass - classification
`0.000 (do_classify_mine nessie lp6) [25.000]`

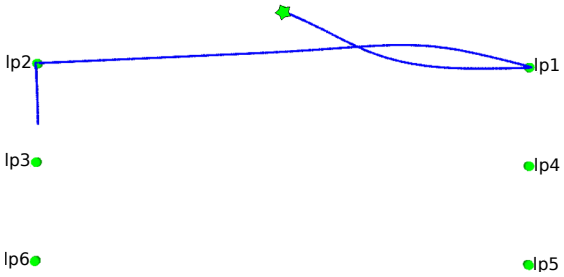
Table 8.5. Plan statistics for the classification plan shown in Table 8.4.

First pass - classification
Estimated Execution Duration: 25.000 secs
Estimated Energy Consumption: 37.500 units
Plan Generation Duration: 0.01 secs
Number of Evaluated States: 3

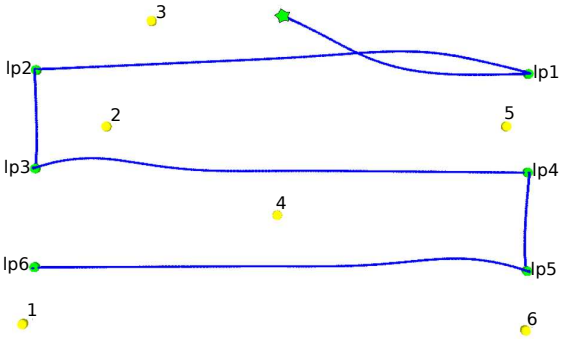
and is linked to its generation time, generation duration, date as well as estimated execution duration and estimated start and end time points (see Sections 7.3.3, 8.1.3.2). Furthermore, it is linked to the single action it consists of which the framework also instantiates inside the MCM execution ontology. The classification action is initially marked as non executed and it has no execution outcome. Moreover, it is linked to its parameters which are the vehicle instance (nessie) and lp6 which is the position where the detection phase has ended. Initially, as in the case of detection actions, the classification action is linked to its estimated execution duration as well as the execution start and end time points. Regarding the actual duration as well as the actual start and end time points, they will be assigned after the action has executed since they are unknown at the time of generating the plan. The same applies for the execution outcome. Finally, we assume that the vehicle is equipped with a classifier that is able to classify MLOs with some confidence level in the form of a probability associated with every MLO. Each such probability can be then transformed into an entropy measurement based on equation 7.5. In order to simulate such a classification system and given the multiple circle detections inside the KB, the detected circles are clustered using the affinity propagation clustering method [189]. Affinity propagation is a method that considers measures of similarity between pairs of data points and is very efficient with uneven cluster sizes. Using affinity propagation, cluster centroids are classified as MLOs and are instantiated inside the MCM world ontology given that the classification action is executable, i.e the classification module is up and running. Except for their positions, MLOs are assigned a random probability within the interval of $[0.50, 0.99]$ based on which the framework calculates entropy which is also assigned to each MLO. Moreover, the reacquisition status of each MLO is initially set to *false* which represents that no MLO has not been reacquired yet. Figure 8.9 illustrates the results of clustering and consequently classification of MLOs. In addition, Figure 8.8c illustrates the completion of the detection phase and the placement of MLOs inside the simulated MCM environment after the completion of the classification phase. While the vehicle executes its `do_classify_mine` action the MCM mission phase is updated to classification whereas when classification finishes the representation of the `do_classify_mine` action is updated accordingly. i.e its



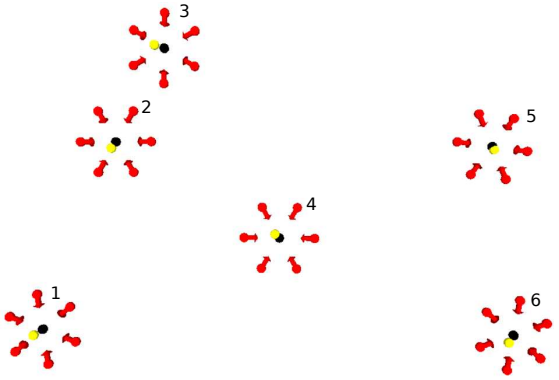
(a) Generated lawnmower trajectory points (lp1-lp6). The green star represents the vehicle's initial position.



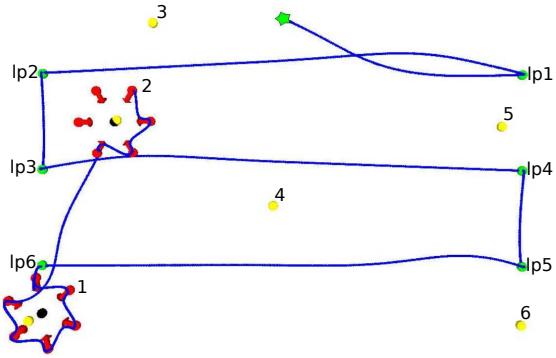
(b) Execution of `do_hover_mine_detection` actions as part of the detection phase.



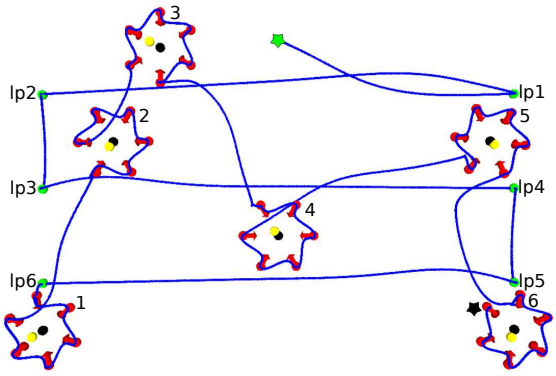
(c) Mine detection phase completion and classification of MLOs (yellow spheres).



(d) Areas 1-6 represent reacquisition and inspection areas. Red spheres with attached arrows represent inspection points, yellow spheres are classified MLOs while black spheres represent mines.



(e) Execution of `do_reacquire_mlo` and `do_inspect_mine` actions in an energy-efficient manner as part of the reacquisition and inspection phases respectively.



(f) Completion of the reacquisition and inspection phases in an energy-efficient manner. The order in which reacquisition and inspection occurred is: 1, 2, 3, 4, 5. The black star represents the mission end point.

Figure 8.8. MCM mission phases and trajectories followed by the vehicle during an energy-efficient MCM mission execution.

execution status, outcome, actual duration as well as actual start and end time points. At the end of the classification phase the framework updates the plan representation inside the execution ontology with respect to the actual start and end time points and duration.

Second pass - reacquisition & inspection: Nessie is now ready to proceed with planning the reacquisition phase. As such the MCM problem phase is updated to reacquisition.

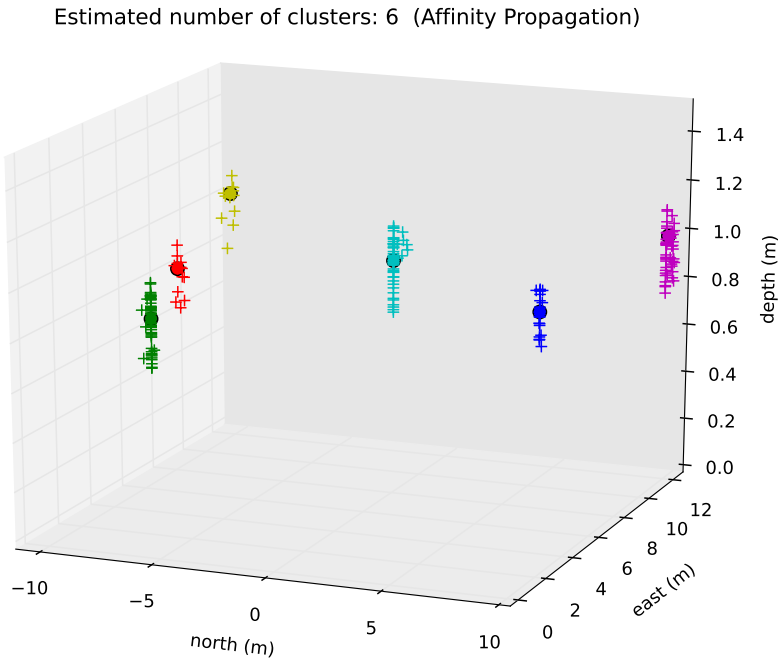


Figure 8.9. Circle detections clustering. Crosses represent detected circular objects while coloured circles represent cluster centroids.

Given this problem phase update and the MLO assertions the framework generates mlopoints both inside the MCM planning ontology and the MCM execution ontology which are marked as unvisited. The framework then proceeds with the reconstruction of the reacquisition problem which along with the MCM domain are passed on to the planner. The goal state of the problem is that the vehicle must reacquire all MLOs and have enough energy to do so. In addition, the reconstructed reacquisition problem contains the following metric: `minimize (consumed_energy auv)`. This instructs the planner to formulate a plan in which MLO reacquisition will be executed in an energy-efficient manner. The reacquisition problem initial state is that none of the MLOs has been reacquired, none of the mlopoints has been visited all mlopoints are reachable form each other and so on. The generated reacquisition plan is illustrated in Table 8.6 (top). Mlopoints mlop1-mlop6 correspond to MLOs 1-6 illustrated in Figure 8.8c while init corresponds to the initial position of the vehicle from which reacquisition will commence. In our case this is the position where the vehicle completed its detection and classification phases. The generated plan will be logged into the MCM execution ontology by the formulation of an instance of the *McmMissionMloReacquisitionPlan* which, as in the case of the phases of detection and classification is given a generation time, date, generation duration, estimated execution duration as well as estimated start and end time points. In addition, the framework creates six *McmMissionDoReacquireMloAction* instances inside the MCM execution ontology, one for each `do_reacquire_mlo` action. Each such action instance is linked to the reacquisition plan instance and is initially marked as as non executed and it has no execution outcome. Moreover, it is linked to its parameters which are the vehicle instance (Nessie) and the two

Table 8.6. Plans for reacquisition of MLOs (top) and inspection of mines (bottom) in an energy-efficient manner. Numbers on the left hand-side represent estimated start time points while numbers on the right hand-side (in brackets) represent estimated durations for each durative action. By estimated time points we refer to the points in time after each plan starts executing with a reference to 0.000 seconds as being the start of the plan execution.

Second pass - reacquisition
0.000 (do_reacquire_mlo nessie init mlop1) [7.850]
7.850 (do_reacquire_mlo nessie mlop1 mlop2) [23.750]
31.600 (do_reacquire_mlo nessie mlop2 mlop3) [13.799]
45.399 (do_reacquire_mlo nessie mlop3 mlop4) [25.875]
71.274 (do_reacquire_mlo nessie mlop4 mlop5) [26.375]
97.649 (do_reacquire_mlo nessie mlop5 mlop6) [23.250]
Second pass - inspection
0.000 (do_inspect_mine nessie init ip1) [5.200]
5.200 (do_inspect_mine nessie ip1 ip2) [16.000]
21.200 (do_inspect_mine nessie ip2 ip3) [16.000]
37.200 (do_inspect_mine nessie ip3 ip4) [16.000]
53.200 (do_inspect_mine nessie ip4 ip5) [16.000]
69.200 (do_inspect_mine nessie ip5 ip6) [16.000]

mlopoint instances that it comprises (e.g. mlop1 and mplop2). Initially, as in the case of detection and classification actions, each reacquisition action is linked to its estimated execution duration as well as the estimated execution start and end time points. Regarding the actual execution duration as well as the actual start and end time points, they will be assigned after each action has executed since they are unknown at the time of generating the plan. The same applies for the execution outcome of each action. Again, each reacquisition action is deemed executable or non executable at runtime given the outcome of the system identification and update phases which are executed continuously and behind the scenes.

As the vehicle starts executing its reacquisition plan the MCM mission phase is updated to reacquisition. Moreover, every time the vehicle reacquires an MLO that is deemed to be a mine it needs to inspect it. Before doing so, the execution status, outcome, (actual) execution duration as well as (actual) start and end time points of the reacquisition action are updated/instantiated accordingly, the MLO is marked as reacquired and the mlopoint (marked) as visited. Furthermore, the framework instantiates the *Mine* class inside MCM world ontology, marks it as uninspected and associates it with its position and radius. Here we make the assumption that there is uncertainty arising from the vehicle sensors. Therefore, the initial positions of the MLOs are inaccurate compared to the positions of the reacquired ones due to their detections being performed from a much longer distance than reacquisition. As such, the positions of the mines are slightly different from the ones of the MLOs before reacquisition.

In order for the vehicle to proceed with planning the inspection phase for that mine the framework updates the MCM problem phase to be inspection and generates six *Inspectionpoint* instances inside the MCM planning ontology and six inside the MCM execution ontology associated with their geometrical information and marked as unvisited. The

framework then proceeds with the reconstruction of the mine inspection problem and MCM domain which feeds into the planner. The goal of the problem is to visit each inspection point and inspect the mine from every inspectionpoint given that there is enough energy to do so. Table 8.6 (bottom) illustrates the generated plan for inspection. "Inspectionpoint" init corresponds to the position from where the vehicle starts travelling towards the first inspection point; in our case the generated plan the vehicle performed reacquisition of the MLO that was deemed to be a mine. The inspection plan is logged into the MCM planning execution ontology by instantiating the *McmMissionMineInspectionPlan* class and the generated instance is given a generation time, date, generation, estimated execution duration as well as estimated start and end time points. In addition, the framework creates six *McmMissionDoInspectMineAction* instances inside the MCM execution ontology, one for each `do_inspect_mine` action. Each such action instance is linked to the inspection plan instance and is initially marked as as non executed and it has no execution outcome. Moreover, as with every action instance so far, it is linked to its parameters which in this case are the vehicle instance (Nessie) and the two inspectionpoint instances that it comprises (e.g. ip1 and ip2). As in the case of detection, classification and reacquisition actions, each inspection action is linked to its estimated execution duration as well as the estimated execution start and end time points. Regarding the actual execution duration as well as the actual start and end time points, they will be instantiated after each action has executed since they are unknown at the time of generating the plan. The same applies for the execution outcome. Again, each inspection action is deemed executable or non executable at runtime given the outcome of the system identification and update phases. Before the execution of inspection the framework appends the remaining inexecuted reacquisition plan at the end of the generated inspection plan. In this manner, the planner does not have to devise a plan for continuing reacquiring MLOs since it has already done so once. As a result, we save the planner from unnecessary computational load, speeding up the planning process altogether. Nessie is now ready to proceed with inspection and the MCM mission phase is now updated to inspection. Additionally, every time Nessie visits an inspectionpoint and performs an inspection from that point that point is marked as visited and inspection as performed from that inspectionpoint. The mine around which the inspection points are placed is marked as inspected when all inspectionpoints have been visited and inspection of the mine has taken place from every single one of them. Once the inspection of the mine finishes the vehicle resumes with the reacquisition plan from where it left it. This interleaving of reacquisition and inspection is continued until all MLOs are reacquired and all mines are inspected given that the vehicle has enough energy to do so. Figure 8.8d illustrates the six reacquisition and inspection areas that correspond to the six mines that where randomly placed in our simulation environment along with the estimated position of the MLOs before reacquisition and the position of those deemed to be mines after. The slight difference in position is also visible. Figure 8.8e illustrates the consecutive execution of `do_reacquire_mlo` and `do_inspect_mine` actions as the MCM mission unfolds while Figure 8.8f illustrates the completion of the reacquisition and inspection phases which also constitutes the comple-

tion of the MCM mission. Table 8.7 illustrates the estimated execution duration, estimated energy consumption, plan generation duration and the distance to be travelled for the reacquisition and inspection plans shown in Table 8.6 as well as the number of states evaluated by the planner in formulating the plans. One interesting observation to make by looking at

Table 8.7. Plan statistics for the plans shown in Table 8.6.

Second pass - reacquisition
Estimated Execution Duration: 120.899 secs
Estimated Energy Consumption: 148.680 units
Plan Generation Duration: 0.38 secs
Distance to Travel: 43.56 meters
Number of Evaluated States: 971
Second pass - inspection
Estimated Execution Duration: 85.200 secs
Estimated Energy Consumption: 32.760 units
Plan Generation Duration: 0.50 secs
Distance to Travel: 7.92 meters
Number of Evaluated States: 1394

Table 8.7 is the the plan generation times are increased when compared to the generation times for the detection and classification plans. This increase is due to the increased number of states that the planner had to evaluate for both devising the reacquisition and inspection plans. Despite that fact, the time to generate plans is still very low. One thing that we would also like to emphasize is that the estimated energy consumption for both plans is the minimum since we are optimizing for energy consumption. Recall that in Section 7.3.1.2 we explained that the minimization of energy consumption or equivalently the maximization of the remaining energy subsumes the minimization of estimated execution duration. As such the estimated execution durations illustrated in Table 8.7 are minimal. Moreover due to the manner in which energy and duration are estimated (see Sections 7.3.1.1, 7.3.1.2) the distance to be travelled is also minimal. The distances between mlopoints and the distances between inspectionpoints are illustrated in Table 8.8.

Table 8.8. Distances between mlopoints (top) and between inspectionpoints (bottom) for the plans shown in Table 8.6.

Distance (in meters)
init-mlop1: 2.34, init-mlop2: 6.45, init-mlop3: 11.16, init-mlop4: 10.11, init-mlop5: 19.79 init-mlop6: 20.02, mlop1-mlop2: 8.7, mlop1-mlop3: 13.42, mlop1-mlop4: 11.18 mlop1-mlop5: 20.94, mlop2-mlop3: 4.72, mlop2-mlop4: 7.91, mlop2-mlop5: 16.03 mlop2-mlop6: 18.93, mlop3-mlop4: 9.55, mlop3-mlop5: 14.86, mlop3-mlop6: 19.81 mlop4-mlop5: 9.75, mlop4-mlop6: 11.02, mlop5-mlop6: 8.5
Distance (in meters)
init-ip1: 0.42, init-ip2: 1.28, init-ip3: 2.22, init-ip4: 2.60, init-ip5: 2.30, init-ip6: 1.41 ip1-ip2: 1.5, ip1-1p3: 2.60, ip1-ip4: 3, ip1-ip5: 2.60, ip1-ip6: 1.5, ip2-ip3: 1.5 ip2-ip4: 2.60, ip2-ip5: 3, ip2-ip6: 2.6, ip3-ip4: 1.5, ip3-ip5: 2.60, ip3-ip6: 3, ip4-ip5: 1.5 ip4-ip6: 2.6, ip5-ip6: 1.5

8.6 Considering Probability and Entropy in Addition to Energy in MCM

In this section we investigate two scenarios that involve detection, classification, reacquisition and inspection of mines in two passes using one AUV as was demonstrated in the energy-efficient MCM scenario of Section 8.5. As such, we have preserved the same experimental setup with the same placement of mines and the same lawnmower area encoded inside the MCM world ontology. In contrast to the energy-efficient MCM however, the purpose of the first scenario is to demonstrate the behaviour of the AUV in reacquiring MLOs while combining energy efficiency and probability efficiency (energy-efficient plus probability-efficient MCM) while the purpose of the second scenario is to demonstrate the behaviour of the AUV while combining energy efficiency and entropy efficiency (energy-efficient plus entropy-efficient MCM). For each scenario the high level mission priority is encoded inside the MCM planning ontology prior to the mission. As we explained earlier in this thesis, combining energy efficiency with probability or entropy efficiency formulates a multi-objective optimization problem (see Section 7.3.1.2). Irrespective of the combination (the high level mission priority), we have chosen a weight of 0.5 ($w = 0.5$) for both objectives to achieve equal contribution of both. Finally, the purpose of this section is to provide a comparative analysis of the three MCM high level mission priorities and their impact in planning and executing MCM missions.

8.6.1 Results

To avoid unnecessary repetition, the phase walk-through that was presented in Section 8.5 will not be presented in this section. In addition, the planning outcomes for the detection, classification and inspection phases are omitted for the same reason. As such, we only present the generated plans for the reacquisition phases under the two different high level mission priorities in Table 8.9 while the probabilities and entropies of each MLO that yielded the different reacquisition plans is shown in Table 8.10. As was explained in Section 8.5, probabilities are random assignments in the interval of $[0.50, 0.99]$ while entropies are calculated based on equation 7.5.

As can be observed by looking at the reacquisition plan under the energy-efficient plus probability-efficient (`Energy-Prob`) high level mission priority, the vehicle tends to visit higher probability MLOs earlier than in the case of energy-efficient reacquisition (see the top of Table 8.6 for a comparison between the reacquisition plans). However, due to the fact that energy efficiency contributes equally to the multi-objective optimization, the planner is also instructed to generate a plan in which the vehicle will also attempt to minimize energy consumption. This leads us to the trajectory illustrated in Figure 8.10a where, for instance, the vehicle does not transition from area 6 to reacquire the MLO in area 3 (MLO 3) but instead transitions to reacquire the MLO in area 5 (MLO 5). That is, despite the fact that the probability of MLO 3 being a mine is greater than the probability of MLO 5

Table 8.9. Plans for reacquisition of MLOs under different high level mission priorities. The plan for energy efficient plus probability efficient reacquisition is shown at the top while the plan for energy efficient plus entropy efficient reacquisition is shown at the bottom. Numbers on the left hand-side represent estimated start time points while numbers on the right hand-side (in brackets) represent estimated durations for each durative action. By estimated time points we refer to the points in time after each plan starts executing with a reference to 0.000 seconds as being the start of the plan execution.

Energy-Prob
`0.000 (do_reacquire_mlo nessie init mlop1) [7.850]`
`7.850 (do_reacquire_mlo nessie mlop1 mlop4) [29.950]`
`37.800 (do_reacquire_mlo nessie mlop4 mlop6) [29.549]`
`67.349 (do_reacquire_mlo nessie mlop6 mlop5) [23.250]`
`90.599 (do_reacquire_mlo nessie mlop5 mlop3) [39.150]`
`129.749 (do_reacquire_mlo nessie mlop3 mlop2) [13.799]`
Energy-Ent
`0.000 (do_reacquire_mlo nessie init mlop2) [18.125]`
`18.125 (do_reacquire_mlo nessie mlop2 mlop3) [13.799]`
`31.924 (do_reacquire_mlo nessie mlop3 mlop4) [25.875]`
`57.799 (do_reacquire_mlo nessie mlop4 mlop5) [26.375]`
`84.174 (do_reacquire_mlo nessie mlop5 mlop6) [23.250]`
`107.424 (do_reacquire_mlo nessie mlop6 mlop1) [52.575]`

Table 8.10. Probabilities and entropies of MLOs.

MLO	1	2	3	4	5	6
Probability	0.96	0.5	0.75	0.8	0.62	0.9
Entropy	0.24	1	0.81	0.72	0.96	0.47

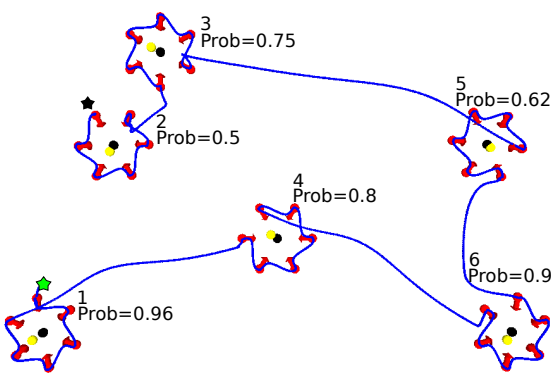
being a mine ($0.75 > 0.62$). Table 8.11 illustrates the estimated execution duration, estimated energy consumption, plan generation duration and the distance to be travelled for the energy-efficient plus probability-efficient reacquisition plan shown at the top of Table 8.9 as well as the number of states evaluated by the planner in formulating the plan. By

Table 8.11. Plan statistics for the energy-efficient plus probability-efficient reacquisition plan shown at the top of Table 8.9.

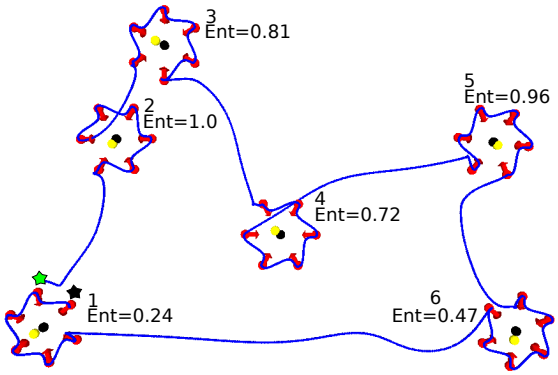
Energy-Prob reacquisition
Estimated Execution Duration: 143.548 secs
Estimated Energy Consumption: 175.9 units
Plan Generation Duration: 1.22 secs
Distance to Travel: 52.62 meters
Number of Evaluated States: 3017

comparing the findings shown in Table 8.11 to the findings illustrated at the top of Table 8.7 (effectively comparing energy-efficient reacquisition with energy-efficient plus probability-efficient reacquisition), we can see that in the second case the vehicle is expected to: i) travel a longer distance, ii) consume more energy and iii) require more time to reacquire targets. In addition, the plan generation duration is increased. Regarding the former three[4] such

[4]distance, energy consumption and execution duration



(a) Energy-efficient plus probability-efficient reacquisition and inspection. The AUV proceeds by reacquiring MLOs and inspecting mines in all the areas in the following order: 1,4,6,5,3,2.



(b) Energy-efficient plus entropy-efficient reacquisition and inspection. The AUV proceeds by reacquiring MLOs and inspecting mines in all the areas in the following order: 2,3,4,5,6,1.

Figure 8.10. Trajectories followed by the vehicle under different high level mission priorities. Start points for each trajectory, denoted by green stars, are the points where the vehicle finished its lawnmower pattern and performed classification to estimate MLOs. Yellow spheres denote the estimated positions of MLOs after classification while black spheres (denote) the positions of mines. Black stars represent mission end points [4].

a finding is expected since we are also considering probability in the optimization and as such only in the best case scenario the findings would agree with the ones presented at the top of Table 8.7. That is, if the probability of each mine-like object being a mine was such that the outcome of generating a plan that would only consider probabilities and not energy would be the same as the plan shown at the top of Table 8.6. With respect to the time the planer requires to generate the energy efficient plus probability efficient reacquisition plan, the increase is due to the fact that the planner has to evaluate more states.

Regarding the reacquisition plan under the energy-efficient plus entropy-efficient (`Energy-Ent`) high level mission priority shown at the bottom of Table 8.9 we can observe that the vehicle tends to visit higher entropy MLOs first as opposed to the energy-efficient reacquisition (see the top of Table 8.6 for a comparison between the reacquisition plans). This is also shown schematically in Figure 8.10b. The transition of the vehicle from its initial position to visit mlop2 and reacquire the MLO in the second area (MLO 2) first given that the MLO in the first area (MLO 1) is so close, is representative of this behaviour. This would not occur if the difference in entropy was not so great due to the fact that energy efficiency contributes equally to the generation of the reacquisition plan. Its contribution is evident by the fact that the vehicle does not transition from the mlop2 to mlop5 to reacquire the MLO in the fifth area but instead visits mlop3 in the third area to reacquire MLO 3. Table 8.12 illustrates the estimated execution duration, estimated energy consumption, plan generation duration and the distance to be travelled for the energy-efficient plus entropy-efficient reacquisition plan shown at the bottom of Table 8.9 as well as the number of states evaluated by the planner in formulating the plan. By comparing the findings shown in Table 8.12 to the findings illustrated at the top of Table 8.7 (effectively comparing energy-

Table 8.12. Plan statistics for the energy-efficient plus entropy-efficient reacquisition plan shown at the bottom of Table 8.9.

Energy-Ent reacquisition
Estimated Execution Duration: 159.999 secs
Estimated Energy Consumption: 195.6 units
Plan Generation Duration: 0.72 secs
Distance to Travel: 59.2 meters
Number of Evaluated States: 1819

efficient reacquisition with energy-efficient plus entropy-efficient reacquisition), we can see that in the second case the vehicle is expected to again: i) travel a longer distance, ii) consume more energy and iii) require more time to reacquire targets. In addition, the plan generation duration is increased. Regarding the former three such a finding is again expected since we are also considering entropy in the optimization and as such only in the best case scenario the findings would agree with the ones presented at the top of Table 8.7. That is, if the entropy related to each mine-like object being a mine was such that the outcome of generating a plan that would only consider entropies and not energy would be the same as the plan shown at the top of Table 8.6. With respect to the time the planer requires to generate the energy efficient plus probability efficient reacquisition plan, the increase is due to the fact that the planner has to evaluate more states.

In order to complete the picture about the two priorities that involve multi-objective optimizations (energy efficiency plus probability efficiency and energy efficiency plus entropy efficiency) we need to report some additional plan generation times that are involved. Recall that we utilize the weighted sum method in order to combine the objectives and linear normalization in order to express them in the same range (see Sections 7.3.1.2, 8.2). This requires the system to calculate the minimum and the maximum of each objective first. For instance, in the case of combining energy with probability the system first needs to calculate the minimum and maximum consumed energies as well as the minimum and maximum sums of probability quotients. In order to do so however the planner needs to solve the problems independently four times and then combine the outcomes on order to formulate the multi-objective optimization problem in which both probability and energy will be expressed in the same range. That is, one for the minimization of energy consumption, one for its maximization, one for the minimization of the sum of probability quotients and one for its maximization and then formulate the multi-objective optimization problem. As such, for the reported plan generation duration in Table 8.11 we need also to consider 5.23 seconds of additional time while for the reported plan generation duration in Table 8.12, 4.5 seconds of additional time. However, even with the additional plan generation durations the overall generation durations for both energy-efficient plus probability-efficient and energy-efficient plus entropy-efficient reacquisitions are still low. That is, low in the sense that such plan generation durations are not expected to impact the applicability of the approach for real time missions in any noteworthy manner.

Now, in order to be able to draw safe conclusions about the impact of the three dif-

ferent high level mission priorities in MCM we additionally generated 10 random geometrical configurations of the six mines within our 22×12 meter area. For each geometrical configuration, after classification, each MLO is again assigned a random probability within the interval $[0.50, 0.99]$ which is also converted to entropy. For each such random geometrical configuration we ran one full MCM mission[5] for each high level mission priority and calculated the average mission statistics that are shown in Table 8.13. As can be seen by observing Table 8.13, the average total estimated plan execution dura-

Table 8.13. Average mission statistics of planning and executing MCM missions under 10 random geometrical configurations of mines using all three high level mission priorities.

High level mission priority	Energy	Energy-Prob	Energy-Ent
Total plan generation duration (secs)	3.69	9.59	8.84
Total estimated plan execution duration (secs)	856.41	875.87	880.25
Total actual plan execution duration (secs)	718.69	731.82	735.45
Total estimated energy consumption (units)	452.02	475.37	478.48
Total KB-related operations duration (secs)	12	14.5	14.6
Total mission duration (secs)	730.69	746.32	750.05

tion for all plans under the energy-efficient (`Energy`) high level mission priority is lower when compared to the average total estimated plan execution duration for all plans under the energy-efficient plus probability-efficient (`Energy-Prob`) and energy-efficient plus entropy-efficient (`Energy-Ent`) high level mission priorities. To avoid any confusion we would like to clarify that we refer to the average combined (total) estimated execution duration of all plans in each mission. That is, the detection, classification, reacquisition and inspection plans. This is expected since in the latter two high level mission priorities (`Energy-Prob`, `Energy-Ent`) we combine energy efficiency with probability and entropy efficiency respectively when reacquiring MLOs. Recall that in Section 7.3.1.2 we explained that the minimization of energy consumption subsumes the minimization of execution duration. As such, by considering entropy or probability in addition to energy will not only have an impact on the energy consumption of the vehicle (energy consumption is increased) but also on the execution duration due to considering different reacquisition plans. That is, in the first case reacquisition plans that minimize energy consumption, in the second case reacquisition plans that minimize energy consumption in conjunction with reacquiring MLOs that are most probable to be mines first and in the third case reacquisition plans that minimize energy consumption in conjunction with reacquiring MLOs that yield the highest information gain first. The results in Table 8.13 clearly show the increase in energy consumption an execution duration. The increase is in fact an increase in the duration and the energy consumption of reacquisition actions for which the different high level mission priorities are considered. That is, irrespective of the high level mission priority chosen for reacquisition; the durations of detection, classification and inspection actions will not be affected.

[5]One full MCM mission comprises detection, classification, reacquisition and inspection operations.

Another interesting observation to be made by looking at Table 8.13 is that the average combined (total) estimated plan execution durations are an over-estimate when compared to the actual ones. That is, the planner estimated that the MCM missions would last longer than they actually did, irrespective of the high level mission priority chosen. This over-estimation is due to the formulation of upper duration bounds for actions as explained in Section 8.4. Needless to say that we could have chosen values for quantities such as the multiplication factor for time that would reduce this over-estimation. In our case the over-estimation is about 17.5%. This allows actions to underrun and not overrun as explained in aforementioned section.

Yet another observation to make when looking at Table 8.4 is that the average combined (total) generation duration for all plans is lower in the case of reacquiring MLOs in an energy efficient manner (`Energy`) as opposed to reacquiring MLOs in an energy-efficient plus probability-efficient (`Energy-Prob`) manner or in an energy-efficient plus entropy-efficient (`Energy-Ent`) manner. This difference is due to the very existence of different high level mission priorities for reacquiring MLOs. More intuitively, irrespective of the high level mission priority chosen for reacquiring MLOs; the average plan generation durations for detection, classification and inspection plans will not change. What will contribute to the increase of the average total generation duration for all plans is the process of performing multi-objective optimizations in the cases of energy-efficient plus probability-efficient and energy-efficient plus entropy-efficient reacquisitions. That is, the system needs to calculate the maximum and the minimum of each objective first in order to be able to use them in the weighted sum approach (see equation 7.4). As such, more planning time is required. However, the reported average total plan generation durations for each MCM mission under the three different high level mission priorities for reacquisition are still low in the sense that durations of the magnitude of a few seconds (3.69, 9.59 and 8.84 seconds) are not expected to have a noteworthy impact on the utilization of the framework for real world applications.

Finally, in Table 8.13 we report the average total duration of KB-related operations for each full MCM as well as the average total mission duration under each high level mission priority. The average total duration of KB-related operations refers to KB-related operations such as the creation, deletion, updating and fetching of instances and properties (knowledge) that do not happen concurrently with the execution of actions. For instance, when traversing a lawnmower trajectory as part of the execution of a detection plan, KB-related operations in the form of instantiations of circle detections inside the world ontology occur concurrently with the execution of the detection actions. As such, they are not included in the average total duration of KB-related operations. In contrast to what we just mentioned, KB-related operations such as the reconstruction of a planning domain and problem do not happen concurrently with action execution. For instance, the reconstruction of a detection problem from the planning ontology occurs before the execution of a detection plan. In fact, the framework cannot devise a plan for detection if the detection (planning) problem and the (planning) domain, are not reconstructed from the planning ontology so that they can be

fed into the planner for detection plan generation. Another example of a KB-related operation that does not occur concurrently with action execution is the assertion inside the KB that a mine has been inspected from a specific inspectionpoint. This KB-related operation occurs in-between inspection action executions. Consequently, the reported average total durations for KB-related operations are added to the average total actual plan execution duration reported in Table 8.13 so that the average total mission duration can be formulated. The small magnitude of the average total durations of KB-related operations that do not happen concurrently with the execution of actions is indicative of the speed at which the KB communicates with the rest of the system.

Going back to the increased energy consumption and duration for MCM missions that combine energy efficiency with probability efficiency or energy efficiency with entropy efficiency as opposed to missions that only consider energy efficiency as the high level mission priority it can be seen that these (increased energy consumption and duration) are compensated in terms of high certainty exploitation and information gain (uncertainty reduction) respectively as shown in Figure 8.11. For the `Energy-Ent` reacquisitions the graph

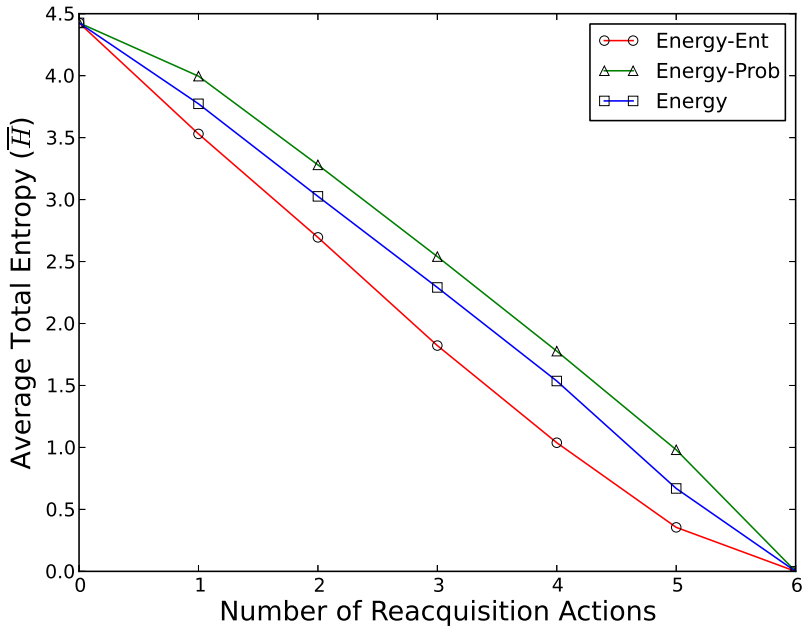


Figure 8.11. Average total entropy reduction over the number of MLO reacquisition actions taken by the vehicle under the three different high level mission priorities.

line is lower compared to the `Energy` and `Energy-Prob` reacquisitions. Additionally, for `Energy-Prob` the graph line is higher than the rest. This means that for `Energy-Ent` the uncertainty about the environment is always lower after every reacquisition compared to the rest while for `Energy-Prob` it is always higher. For the `Energy` reacquisition it is somewhere in-between. The fact that the probabilities of MLOs are in the interval $[0.50, 0.99]$ means that during `Energy-Prob` reacquisitions the AUV tends to visit MLOs with higher probability first (see also Figure 7.13 for a better understanding).

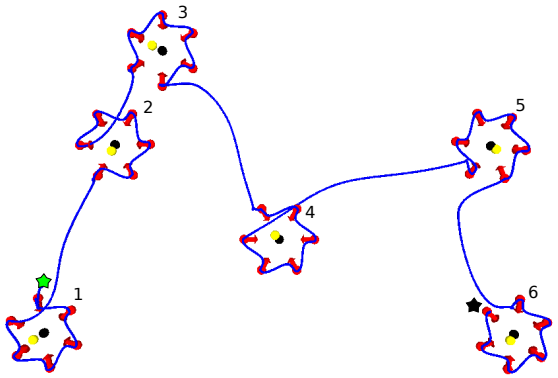
8.7 High Level MCM Mission Priority Changes and Adaptation

So far have we have presented how the framework can be used for planning and execution of MCM missions successfully using the knowledge residing into the framework's ontologies under different high level MCM mission priorities. In this section we test the framework's ability to adapt in the presence of changes in high level MCM mission priorities issued to the vehicle during mission execution. More specifically, we reuse the 10 random geometrical configurations of mines generated for the experiments of Section 8.6 as well as the associated probabilities and entropies for MLOs to initially plan and execute the same missions. However, we now issue high level mission MCM mission priority changes during the interleaved execution of the reacquisition and inspection phases. In doing so, we test the adaptive behaviour of our framework and its impact on mission planning and execution.

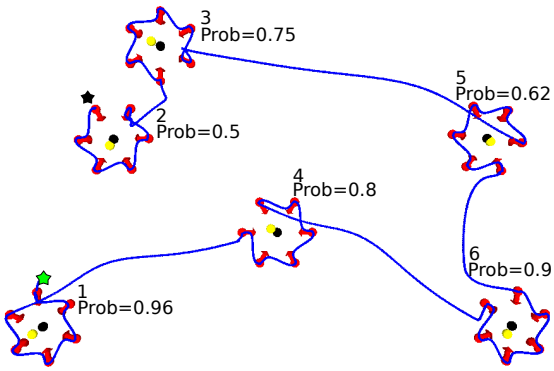
8.7.1 Results

The reuse of the same random geometrical configurations as well as the same probabilities and entropies for each setting enables us to perform a direct comparison between our previous findings and the newly generated ones. Figure 8.12 is representative of the AUV's adaptive behaviour. Figures 8.12a-8.12c illustrate the reacquisition and inspection trajectories followed by the simulated AUV due to planning for reacquisition based on different high level mission priorities. In the mission illustrated in Figure 8.12d, the initial plan for the vehicle was to reacquire MLOs with the high level mission priority being energy efficiency and inspect those deemed to be mines as shown in Figure 8.12a. After visiting areas 1 and 4 however, a request for changing the high level mission priority into energy efficiency plus entropy efficiency took place and the vehicle adapted accordingly. This adaptation was due to replanning caused by invalidating the previously generated plan. A similar adaptation is shown in Figure 8.12e where, a command for switching from energy efficiency plus entropy efficiency as the high level mission priority (Figure 8.12c) to the energy efficiency one is issued to the vehicle.

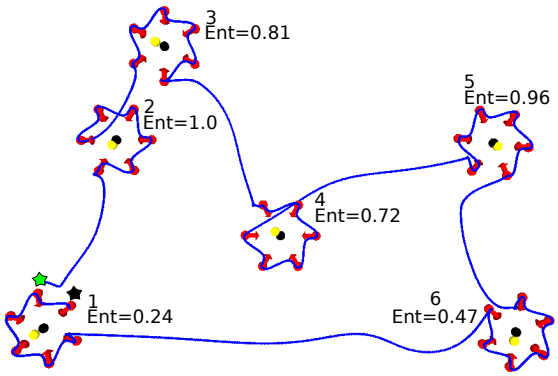
Except for the successful adaptation of planning and execution when issuing high level mission priority changes, we also tested its impact on the average total entropy reduction as the reacquisition of MLOs unfolded during mission execution (see Figure 8.13). As expected, transitioning from energy-efficient plus probability-efficient reacquisition to energy-efficient plus entropy-efficient reacquisition leads to a lower average total entropy (disorder/uncertainty) after each reacquisition action. In contrast, when transitioning from an energy-efficient plus entropy-efficient reacquisition to just energy-efficient reacquisition the average total entropy in the environment is higher after each reacquisition. That is, until entropy becomes zero in the environment for all high level mission priorities which occurs after all reacquisitions have taken place.



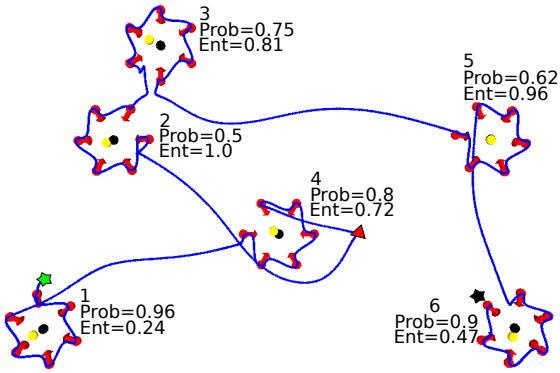
(a) Energy-efficient reacquisition and inspection. The AUV proceeds by reacquiring MLOs and inspecting mines in all the areas in the following order: 1, 2, 3, 4, 5, 6.



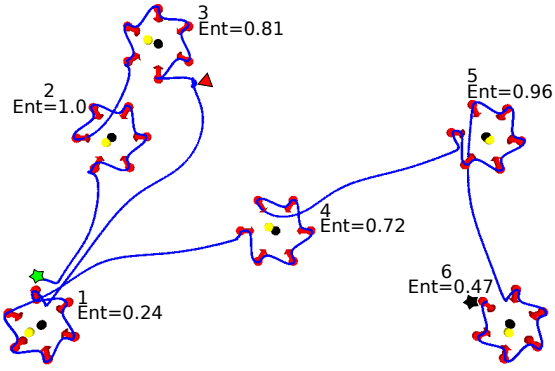
(b) Energy-efficient plus probability-efficient reacquisition and inspection. The AUV proceeds by reacquiring MLOs and inspecting mines in all the areas in the following order: 1,4,6,5,3,2.



(c) Energy-efficient plus entropy-efficient reacquisition and inspection. The AUV proceeds by reacquiring MLOs and inspecting mines in all the areas in the following order: 2,3,4,5,6,1.



(d) Adaptation from energy-efficient plus probability-efficient reacquisition and inspection (areas 1, 4) to energy-efficient plus entropy-efficient reacquisition and inspection (areas 2, 3, 5, 6).



(e) Adaptation from energy-efficient plus entropy-efficient reacquisition and inspection (areas 2, 3) to energy-efficient reacquisition and inspection (areas 1, 4, 5, 6).

Figure 8.12. High level mission priorities and adaptation. Start points for each trajectory, denoted by green stars, are the points where the vehicle finished its lawnmower pattern and performed classification to estimate MLOs. Black stars represent mission end points. Red triangles denote the points in which adaptation occurred.

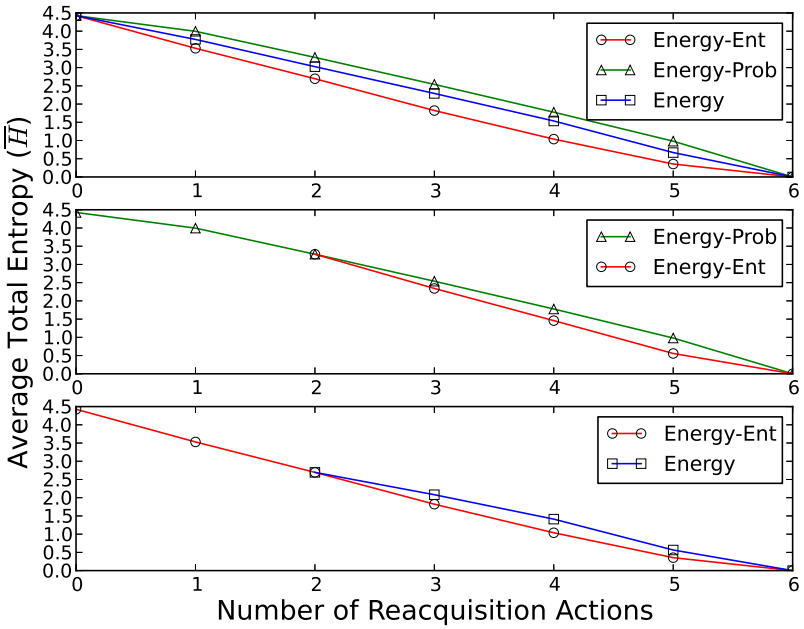


Figure 8.13. Average total entropy reduction over the number of MLO reacquisition actions taken by the vehicle: i) under the three different high level mission priorities (upper graph), ii) when adapting from the `Energy-Prob` to the `Energy-Ent` priority (middle graph), iii) when adapting from the `Energy-Ent` to the `Energy` priority (bottom graph).

8.8 Component Faults and Fault Recovery Through Adaptation

Component faults during mission execution can jeopardise missions and lead to undesirable outcomes whose impact can vary depending on the component that malfunctioned. For instance, if the battery(ies) of the AUV malfunction(s) and we have total power loss during mission execution then the mission cannot continue any more. This is due to the battery of the vehicle being a critical component for the vehicle. On the other hand, there are component faults that the vehicle can overcome subject to the existence of redundant functional components on-board that can be used as a substitute for the faulty component.

The modelling of *semantically equivalent* capabilities and actions in our framework offers a solid basis upon which an AUV executing an MCM mission can rely in order to recover from a fault given that recovery is possible, i.e the component is not critical. Also, it is an undeniable fact that mission planning can be a very demanding process in terms of computational resources at time. As such, our approach favours adaptive execution over adaptive planning in the presence of component faults by initially searching for semantically equivalent actions to be executed. In the event that this is not effectual, i.e. if none is executable, then the framework resorts to replanning given the current state of both the vehicle and the world in conjunction with remaining mission goals (adaptive planning).

8.8.1 Results

In order to test our vehicle's behaviour with respect to component faults we initially simulated a mine detection (software) module fault during the execution of a lawnmower trajectory for mine detection. Recall from Section 8.1.2 and by extension Section 7.2.2.2 that the mine detection capability of the vehicle is available when either mine detection capability (primary and/or secondary) is available to the system. However, both the primary and secondary mine detection capabilities depend on the availability of the mine detection module. Hence, simulating a mine detection module fault leaves the vehicle deprived of its mine detection capability altogether. This in succession, causes both the primary and the secondary mine detection actions to not be available (inexecutable) since both depend on the mine detection module. As such, the vehicle cannot resort to adaptive execution through the execution of a semantically equivalent action. However, due to the the mine detection module being a critical component only for mine detection the framework causes the vehicle to resort to adaptation on the planning level in order to satisfy the remaining mission goals to the maximum extend possible. That is, classification of MLOs given potential circle detections so far as well as reacquisition of MLOs, if any, and inspection of those deemed to be mines. Figure 8.14 illustrates where we simulated the mine detection module fault and the adaptation of the vehicle which led to the continuation of the MCM mission. This adaptation of course would not be possible if the vehicle was not equipped

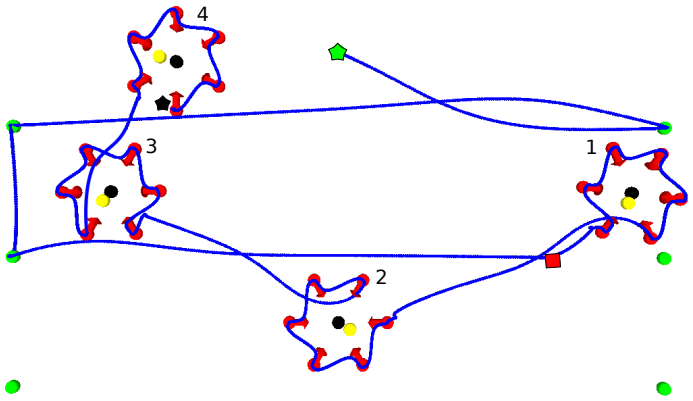


Figure 8.14. Adaptation due to mine detection (software) module fault (red square). The mission start point is represented by the green star while the mission end point is represented by the black one. The geometrical configuration of mines used in this experiment is the same as the configuration illustrated throughout this chapter while the high level mission priority chosen is energy efficiency.

with the components, capabilities, actions and vehicle ontologies as well as the Prolog-based reasoning. That is, the continuous assessment of the vehicle's internal state by the continuously successive repetition of the system identification and system update phases in real time and behind the scenes.

In another experiment we conducted, we simulated a sonar fault during the lawnmower trajectory execution as shown in Figure 8.15. Due to the sonar being a critical component for the primary mine detection capability which in succession is necessary for the primary

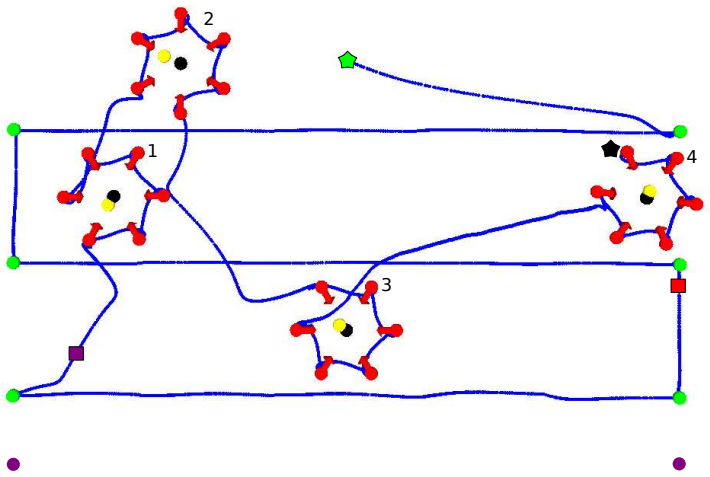


Figure 8.15. Adaptation due to sonar fault (red square). The mission start point is represented by the green star while the mission end point is represented by the black one. Purple spheres represent the mines that the vehicle was not able to detect due to the limited range of the camera while the purple square represents the recovery of the sonar. The geometrical configuration of mines used in this experiment is the same as the configuration illustrated throughout this chapter while the high level mission priority chosen is energy efficiency.

mine detection action the vehicle was forced to search for semantically equivalent actions to continue mission execution. This lead to the vehicle perform an adaptation on the execution level since the secondary mine detection capability which depends on the camera was still available as was, consequently, the secondary mine detection action. However, due to the limited range of the camera the vehicle was not able to perform circle detections that corresponded to some of the mines in the environment as shown in Figure 8.15. As such the classification system classified MLOs only for areas 1-4. Moreover, due to the faulty sonar, the vehicle started moving towards the first MLO executing secondary MLO reacquisition actions until we simulated a sonar recovery. This led to the vehicle switch to the execution of primary MLO reacquisition actions and continue the reacquisitions and inspections until the end.

8.9 Summary and Conclusions

This chapter was concerned with persistently autonomous MCM which constitutes an application of the ontology-based framework presented in Chapter 7 in the MCM setting. As such, this chapter makes the following contributions:

- As opposed to standard MCM missions where energy consumption and/or execution time govern missions we have provided an alternative perspective where criteria such as probability and entropy are also considered.

- We have utilised ontology-based knowledge representation and reasoning not only to plan and execute MCM missions but also drive adaptation both on the planning

and execution level (of missions) in response to real time high level mission priority changes and component faults.

Let us now elaborate on the above contributions and present a summary of the chapter.

We have made some modifications to the framework ontologies in order to accommodate MCM. For example, we extended the generic *Object* class inside the world ontology, with a *Mlo* and a *Mine* class in order to model mine-like objects and mines respectively. Similarly, we extended the generic *ClassificationModule* class with a *MineClassificationModule* in order model a mine classifier and so on. However, due to the flexibility of the framework, such modifications were minimal. It is also important to emphasize that by no means the presented modifications throughout this chapter constituted a modification to the framework's architecture or how the problem of persistently autonomous operations is addressed within the framework.

Experiments were conducted in simulation and concerned MCM in the form of DC RIn using a simulated version of the Nessie V AUV. That is, four phases in two passes with one AUV. In the aforementioned setting we have tested the ability of the framework to plan and execute MCM missions under different high level mission priorities. As such, we have moved from the standard MCM missions where energy consumption and/or execution time govern the mission and provided an alternative perspective in which criteria such as probability and entropy are also considered with promising results. That is, despite an increase in the vehicle's energy consumption and execution time, the combination of an energy-efficient plus probability-efficient reacquisition of mine-like objects benefits MCM in the sense that the vehicle tends to reacquire high probability mine-like objects first while at the same time attempting to minimize energy consumption and execution time. In the case of energy-efficient plus entropy-efficient reacquisitions the vehicle's energy consumption and execution time are also increased as opposed to the standard MCM. However, this is compensated in terms of information gain (uncertainty reduction). That is, the vehicle conducting MCM tends to reacquire higher entropy mine-like objects first while at the same time attempting to minimize energy consumption and execution time. Further, the experimental results demonstrate how an ontology-based knowledge representation and reasoning approach can drive adaptation both on the planning and execution level of missions. More specifically, adaptive planning due to high level mission priority changes is an important feature because it facilitates the need for adaptation due to real time mission requirement changes. The experimental results demonstrate the robustness of the framework in that respect. Moreover, experimental results demonstrate the efficiency of the framework in recovering from faults in critical components through adaptive planning. In this manner, the framework facilitates the continuation of missions and satisfaction of mission goals to the maximum extent possible. On the other hand, the ability of the framework to recover from faults in components for which there are redundancies, through adaptive execution, demonstrates the benefit of encoding semantically equivalent capabilities and actions. That is, preserving computational resources and resorting to adaptive planning only when necessary. This is a highly desirable feature especially in the context of autonomous operations

with AUVs where computational resources are limited.

Chapter 9

Training and Evaluating MLNs for Maritime Situation Awareness

So far in this thesis we have presented our work with respect to persistently autonomous operations with a focus on MCM in the context of maritime defence. However, maritime defence is only one of the two aspects of the maritime defence and security domain with which this thesis is concerned (see Chapter 1 for a reminder about the problem statement and the thesis objectives). The other being maritime security in which maritime situation awareness plays a key role.

Recall that in Chapter 2 we presented knowledge representation and reasoning approaches based on logic, probabilistic graphical models as well as hybrid approaches that unify logic and probability into a single representation. In addition, in Chapter 5 we presented an overview of the maritime situation awareness domain including a number of maritime situation awareness systems. Among the presented pieces of work one that particularly drew our attention was the work of Snidaro et al. [5]. To the best of our knowledge, it is the only work in the field that utilizes MLNs for building maritime situation awareness. However, despite the well researched domain of learning the weights of MLNs [93], [190], [191], [192], [193], the authors set the weights manually. In addition, they do not evaluate the performance of the networks in depth since they only provide exemplar query outcomes without assessing the overall performance of the networks on a test set. As such, in this chapter we present our work with respect to training and evaluating MLNs for maritime situation awareness.

More specifically, in Section 9.1 we show how MLN weights can be learned generatively using likelihood while in Section 9.2 we briefly present Alchemy, a framework for statistical relational learning with MLNs which we used in our experiments. The maritime situation awareness scenario investigated in this chapter concerns the identification of vessels rendezvousing in order to get involved in illegal activities and is based on the work of Snidaro et al. [5] (see Section 9.3). In that respect, the experiments also aim at demonstrating the extent to which contextual information has an impact on the performance of the network in assessing an evolving situation.

For this chapter, the reader is referred to background Chapters 2, 5 for a reminder about MLNs and maritime situation awareness respectively.

9.1 Learning MLN Weights Generatively Using Likelihood

Learning MLN weights, also known as MLN parameter learning, is a function optimization problem [193]. This section is concerned with learning MLN weights generatively using: i) Maximum Likelihood (ML) and ii) Maximum Pseudo-Likelihood (MSL) [193]. An alternative approach, using likelihood as the basis for optimization, is to learn the MLN weights discriminatively by maximizing Conditional Likelihood (CL) [93]. That is, if we know the query predicates of the MLN (l) as well as the evidence predicates (k) prior to learning, we can maximize the CL of l given k instead of maximizing the joint (pseudo) likelihood of k and l which is the case in generative learning.

9.1.1 Maximum Likelihood (ML) Learning

In order to perform ML learning in an MLN we maximize the logarithm of Equation 2.15 (log-likelihood) which is the following:

$$\begin{aligned} f(w) &= \log P_w(X = x) \\ &= \log\left(\frac{1}{Z}\exp\left(\sum_i w_i n_i(x)\right)\right) \\ &= \sum_i w_i n_i(x) - \log Z \end{aligned} \tag{9.1}$$

where, $Z = \sum_x \exp\left(\sum_i w_i n_i(x')\right)$. We assume that the training data are in the form of ground atoms [193]. Let m be the number of all possible ground atoms. As such, the training database can be represented as a vector $x = \{x_1, x_2, ..., x_m\}$ where each element represents a binary truth value $(0, 1)$. Zero corresponds to the respective possible ground atom not being in the training database while one corresponds to the opposite. This effectively imposes the closed world assumption [193]. In addition, x' represents all possible databases. Now, maximization of function 9.1 can be achieved by applying standard *first-order* or *second-order* gradient-based optimization methods. One of the most used first-order methods is the gradient descent (gradient ascent for maximization) while conjugate gradient, Newton's method as well as various quasi-Newton methods such as the Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm are some of the most used second order methods. The interested reader is referred to [194] for additional (technical) information on the gradient-based methods as well as on their applicability to optimization problems.

(Log)-likelihood is a concave function and as such it does not suffer from local optima. To be more precise, a local maximum of a concave function is at the same time a global maximum. The derivative of the log-likelihood (Expression 9.1) with respect to a weight

(partial derivative) is the following [193]:

$$\begin{aligned}\frac{\partial}{\partial w_j}\log P_w(X=x) &= n_j(x) - \frac{1}{Z}\sum_{x'}\exp\Big(\sum_i w_i n_i(x')\Big)n_j(x') \\ &= n_j(x) - \sum_{x'} P_w(X=x')n_j(x') \\ &= n_j(x) - E_w[n_j(x)]\end{aligned} \tag{9.2}$$

where, $n_j(x)$ is the number of true groundings of the jth formula in the training database x while $E_w[n_j(x)]$ is the expected (predicted) number of true groundings of the jth formula according to the current model given the current weight vector $w = (w_1, w_2, ...w_j, ...)$. Ergo, if the model predicts that the number of true grounding of the jth formula is higher than it actually is then w_j needs to decrease while if the prediction is lower the w_j needs to increase. Having said that, one can easily deduct that Expression 9.2 represents the jth component of the gradient. The gradient as a whole is given by Expression 9.2 applied for all formulas.

The problem that arises however, is that counting the true groundings of a formula in the training data exactly is generally intractable. According to Richardson and Domingos [193] this can be solved by counting the true groundings approximately by uniform sampling. In addition, generally intractable is also the counting of the expected number of true groundings which necessarily requires inference over the MLN [193]. According to Richardson and Domingos [193] approximate methods can be used but they tend not to converge, as they state, in reasonable time.

9.1.2 Maximum Pseudo-Likelihood (MSL) Learning

An alternative to ML learning is to use MSL learning. The idea of maximizing pseudo-likelihood originates back in the seventies [195] and the research conducted with respect to MNs. The pseudo-likelihood is given by the following [193]:

$$PL_w(X=x) = \prod_{l=1}^{m} P_w(X_l = x_l | MB(X_l)) \tag{9.3}$$

where, x is again a training database in the form of a vector like in the case of maximum likelihood ($x = \{x_1, x_2, ..., x_m\}$) with x_l representing the binary truth value of the lth possible ground atom with respect to being or not being in the data. Moreover, $MB(X_l)$ represents the truth values of the ground atoms that constitute X_l's Markov Blanket[1] (MB). By taking the logarithm of pseudo-likelihood (Expression 9.3) we get the log-pseudo-likelihood

[1] Since $X_l = x_l$ we refer to x_l's Markov blanket.

which is the following [193]:

$$\begin{aligned} \log PL_w(X=x) &= \log \prod_{l=1}^{m} P_w(X_l = x_l | MB(X_l)) \\ &= \sum_{l=1}^{m} \log P_w(X_l = x_l | MB(X_l)) \end{aligned} \tag{9.4}$$

The derivative of the log-pseudo-likelihood (Expression 9.4) with respect to a weight (partial derivative) is the following [193]:

$$\begin{aligned} \frac{\partial}{\partial w_j} \log PL_w(X=x) = \sum_{l=1}^{m} \big[& n_j(x) - P_w(X_l = 0 | MB(X_l)) n_j(x_{[X_l=0]}) \\ & - P_w(X_l = 1 | MB(X_l)) n_j(x_{[X_l=1]}) \big] \end{aligned} \tag{9.5}$$

where, $n_j(x)$ is the number of true groundings of the jth MLN formula in the training data, $n_j(x_{[X_l=0]})$ is the number of true groundings of the jth MLN formula when we force X_l to be zero ($X_l = 0$) without changing the remaining data while $n_j(x_{[X_l=1]})$ is the number of true groundings of the jth MLN formula when we force X_l to be one ($X_l = 1$) without changing the remaining data. Compared to the case of the log-likelihood in Section 9.1.1 computing the derivatives of log-pseudo-likelihood or log-pseudo-likelihood itself does not require inference over the MLN. For optimization (maximization) we can utilize any first or second-order gradient-based technique as in the case of maximum likelihood.

9.2 The Alchemy Framework

Alchemy is an open source software framework for statistical relational learning with Markov logic networks that was developed by the department of computer science and engineering, University of Washington [196]. The purpose of Alchemy is to assist researchers in the field of SRL and Machine Learning (ML) in building and testing MLNs for their applications in a fast and reliable manner. As such, it offers a variety of algorithms for structure and weight learning as well as inference. For more information on Alchemy the interested reader is referred to [196] where a tutorial as well as various datasets and exemplar MLNs are also provided for familiarization with both the framework and the process of building MLNs for real world applications. In this chapter we use Alchemy for training and testing the performance of MLNs for maritime situation awareness.

9.3 The Rendezvous Scenario

The rendezvous scenario is adjusted from the work presented in Snidaro et al. [5]. As in the case of Snidaro et al. [5] it is concerned with the identification of pairs of vessels

#	Formula
1	$Overlaps(v,y) \Leftrightarrow Overlaps(y,v).$
2	$Meets(v,y) \Leftrightarrow Meets(y,v).$
3	$Proximity(v,y) \Leftrightarrow Proximity(y,v).$
4	$Rendezvous(v,y) \Leftrightarrow Rendezvous(y,v).$
5	$Stopped(v) \wedge (IsIn(v,OpenSea) \vee IsIn(v,IntWaters)) \Rightarrow Suspicious(v)$
6	$Stopped(v) \wedge (IsIn(v,Harbour) \vee IsIn(v,NearCoast)) \Rightarrow \neg Suspicious(v)$
7	$\neg AIS(v) \Rightarrow Alarm(v).$
8	$\neg InsideCorridor(v) \Rightarrow Suspicious(v)$
9	$Humint(v,Smuggling) \Rightarrow Suspicious(v)$
10	$Humint(v,Clear) \Rightarrow \neg Suspicious(v)$
11	$Suspicious(v) \Rightarrow Alarm(v).$
12	$\neg Suspicious(v) \Rightarrow \neg Alarm(v)$
13	$IsIn(v,z) \Rightarrow (z \neg zp) \wedge \neg IsIn(v,zp).$
14	$IsIn(v,z) \wedge IsIn(y,zp) \wedge (z \neg zp) \Rightarrow \neg Proximity(v,y).$
15	$\neg Proximity(v,y) \Rightarrow \neg Rendezvous(v,y).$
16	$Suspicious(v) \wedge Suspicious(y) \wedge (Overlaps(v,y) \vee Meets(v,y)) \wedge Proximity(v,y) \Rightarrow Rendezvous(v,y).$
17	$(Overlaps(v,y) \vee Meets(v,y)) \wedge Proximity(v,y) \Rightarrow Rendezvous(v,y)$
18	$\neg Stopped(v) \neg Stopped(y) \Rightarrow \neg Rendezvous(v,y)$
19	$Before(v,y) \wedge Proximity(v,y) \Rightarrow \neg Rendezvous(v,y)$

Table 9.1. MLN formulas used in the rendezvous scenario presented in Snidaro et al. [5]. Variables v,y represent vessels while *OpenSea*, *IntWaters*, *Harbour*, *NearCoast* are constants with which a zone variable can become bound and *Smuggling*, *Clear* constants with which a report variable can become bound.

rendezvousing in order to get involved in smuggling activities (e.g. drugs, oil, arms[2] etc.). According to Snidaro et al. vessels stopping at open sea or international waters (zones) constitute an indication of suspicion as opposed to vessels stopping near the coast or at harbours (zones). This is also the case for vessels moving outside traffic corridors as well as vessels for which there is a human intelligence report that they are smuggling. In addition, vessels who do not transmit AIS data or are suspicious should raise an alarm to the human operator. Finally, in Snidaro et al. [5] the incident of rendezvous is determined by a combination of the following indicators: vessels being suspicious, vessels being in the same zone, a vessel leaving a zone well before another comes in, vessels meeting in the same zone, e.g. a vessel leaves the harbour just after another vessel arrives and vessels overlapping temporally in the same zone, e.g. vessels being moored in a harbour at the same time. The latter three indicators are temporal ones. Table 9.1 illustrates the rendezvous scenario MLN formulas as they are presented in Snidaro et al. [5] while Table 9.2 illustrates the adjusted rendezvous scenario MLN formulas that we used in our experiments. Before proceeding with the comparison of the two sets of formulas let us first disambiguate a few things. Where present, full stops at the end of formulas in both sets denote a deterministic (hard) constraint which practically means that such formulas are equivalent to the "traditional" first-order logical formulas. Further, the notations $\neg$ and ! are equivalent and denote negation. Finally, where present in the set of formulas of Table 9.2, the $+$ notation preced-

[2]For brevity, smuggling of arms is usually referred to as arms trafficking

#	Formula
1	$Proximity(x,y) \Leftrightarrow Proximity(y,x).$
2	$Rendezvous(x,y) \Leftrightarrow Rendezvous(y,x).$
3	$Stopped(x) \wedge IsIn(x,+z) \Rightarrow Suspicious(x)$
4	$!AIS(x) \Rightarrow Alarm(x)$
5	$AIS(x) \Rightarrow !Alarm(x)$
6	$!InsideCorridor(x) \Rightarrow Suspicious(x)$
7	$InsideCorridor(x) \Rightarrow !Suspicious(x)$
8	$Humint(x,+r) \Rightarrow Suspicious(x)$
9	$Suspicious(x) \Rightarrow Alarm(x)$
10	$!Suspicious(x) \Rightarrow !Alarm(x)$
11	$IsIn(x,z) \wedge (z! = zp) \Rightarrow !IsIn(x,zp).$
12	$(Suspicious(x) \vee Suspicious(y)) \wedge Proximity(x,y) \Rightarrow Rendezvous(x,y)$
13	$(!Stopped(x) \vee !Stopped(y)) \Rightarrow !Rendezvous(x,y)$
14	$!Proximity(x,y) \Rightarrow !Rendezvous(x,y)$

Table 9.2. Adjusted rendezvous scenario MLN formulas. Variables x, y represent vessels while z is a zone variable which can become bound with constants *OpenSea*, *IntWaters*, *Harbour*, *NearCoast* and r is a report variable which can become bound with constants *Smuggling*, *Clear*.

ing a variable represents the grounding of that variable with all available constants for that variable during training yielding in this manner additional formulas. For instance, given the *OpenSea*, *IntWaters*, *Harbour*, *NearCoast* constants, formula #3 will be expanded into the following four formulas:

- $Stopped(x) \wedge IsIn(x, OpenSea) \Rightarrow Suspicious(x)$
- $Stopped(x) \wedge IsIn(x, IntWaters) \Rightarrow Suspicious(x)$
- $Stopped(x) \wedge IsIn(x, Harbour) \Rightarrow Suspicious(x)$
- $Stopped(x) \wedge IsIn(x, NearCoast) \Rightarrow Suspicious(x)$

where, each one will be assigned a different weight. The aforementioned $+$ notation is specific to the Alchemy framework.

Comparing the two sets of MLN formulas one can observe that in the adjusted set we have omitted the explicit representation of temporal relations that refer to vessels, i.e *Overlaps*, *Meets* and *Before*. The reason for doing so is that we have embedded the relations in the *Proximity* predicate. More specifically, in the case of formulas in Table 9.1, predicate *Proximity* does not actually refer to proximity of two vessels but instead to vessels being in the same zone which can be either the *OpenSea*, *IntWaters*, *Harbour* and *NearCoast*. As such, there exists the need for explicit definition of temporal relations among vessels. For example, consider formula #16 in Table 9.1, where *Proximity* needs to be accompanied by temporal relation *Overlaps*. In our case we give *Proximity* its natural meaning which is vessels being close to each other at the same time. Ergo, there is no need to define an explicit *Overlaps* temporal relation. In addition, in this context, we find temporal relations *Meets* and *Before* redundant since according to Snidaro et al. [5], a vessel which is in

some area and then leaves before some other vessel comes in the same area means that the vessels do not meet. In short, temporal relations as they are defined in Snidaro et al. [5] add unnecessary complexity to the set of formulas. The existence of a hard constraint in formula #14 of Table 9.1 is problematic in the sense that it restricts the model in considering vessels close to each other only when they are in the same zone. However, in reality, vessels can be close to each other even when they are in different zones. For instance, consider the case of a vessel being close to the boundaries of the *OpenSea* zone with the *IntWaters* zone but in the *OpenSea* zone and another vessel being close to the boundaries of the *IntWaters* zone with the *OpenSea* zone but in the *IntWaters* zone. With the representation of Snidaro et al. [5] such vessels can never be in proximity because they belong to different zones. This however, is hardly the case.

9.3.1 Context Encoding

As in the work presented in [5], context in the adjusted set of formulas is encoded with predicates $IsIn(x,z)$ and $InsideCorridor(x)$ (see Table 9.4). However, regarding the $IsIn(x,z)$ predicates, we do not group them in pairs and by extension we do not group zones (*OpenSea*, *IntWaters*, *Harbour* and *NearCoast*) when examining whether vessels stopping within them are suspicious or not. Instead, we consider each zone individually. In this manner, we aim at capturing the impact of a vessel stopping at each zone on being suspicious (the vessel) more accurately. Further, a sharp-eyed reader has probably identified that in the case of formula #6 (Table 9.1) the consequent is a negation while in the expanded adjusted set of formulas the consequent is non negated, i.e $\neg Suspicious(v)$ versus $Suspicious(x)$. This might seem as the representation of the opposite in the adjusted set of formulas. However this is not necessarily the case. In the modelling of the rendezvous scenario by Snidaro et al. [5] all the hard-coded formula weights are positive. As such, if we want to express that a vessel stopping at a harbour or near a coast is not likely to be suspicious we need to take the negation of suspicious and assign a positive weight. Alternatively, we can avoid using negation if we assign a negative weight to the formula expressing effectively, the same thing. This is the case in the adjusted set of formulas where weights can be either positive or negative. Finally, with respect to the $InsideCorridor(x)$ predicate, we have added formula #7 in the adjusted set of formulas to encode that vessels moving within a traffic corridor are not likely to be suspicious.

9.3.2 The Artificial Rendezvous Dataset

Unfortunately, we did not have access to data from real world observations of vessels, hence we constructed a synthetic dataset based on the rendezvous scenario characteristics presented at the beginning of Section 9.3. As such, the dataset consists of observations from 1562 vessels, 786 of which are involved in a rendezvous incident. Further, of the total of 1562 vessels, 1144 demonstrate suspicious behaviour while 1104 raise an alarm to the human operator. The additional numbers of vessels involved in rendezvous

Dataset Entry	1	2
Ground Atoms	$!Stopped(V60)$	$Stopped(V10)$
	$!Stopped(V12)$	$Stopped(V22)$
	$IsIn(V60, IntWaters)$	$IsIn(V10, OpenSea)$
	$IsIn(V12, Harbour)$	$IsIn(V22, IntWaters)$
	$!InsideCorridor(V60)$	$Suspicious(V10)$
	$Humint(V60, Clear)$	$!InsideCorridor(V10)$
	$!Suspicious(V60)$	$Humint(V10, Smuggling)$
	$InsideCorridor(V12)$	$Suspicious(V22)$
	$Humint(V12, Clear)$	$!InsideCorridor(V22)$
	$!Suspicious(V12)$	$Humint(V22, Smuggling)$
	$!AIS(V60)$	$AIS(V10)$
	$!Alarm(V60)$	$Alarm(V10)$
	$AIS(V12)$	$AIS(V22)$
	$!Alarm(V12)$	$Alarm(V22)$
	$!Proximity(V60, V12)$	$Proximity(V10, V22)$
	$!Proximity(V12, V60)$	$Proximity(V22, V10)$
	$!Rendezvous(V60, V12)$	$Rendezvous(V10, V22)$
	$!Rendezvous(V12, V60)$	$Rendezvous(V22, V10)$

Table 9.3. Exemplar entries from the artificially generated dataset.

incidents, demonstrating suspicious behaviour and raising an alarm do not add up to the total number of vessels since the aforementioned assessments are not mutually exclusive. The dataset is in the form of ground atoms and ground atoms that are not present in the dataset are considered to be false. In this manner we abide to the closed world assumption that was mentioned in Section 9.1.1. Table 9.3 illustrates two exemplar entries from the artificial dataset. The first entry represents two vessels $(V12, V60)$ that: are not suspicious $(!Suspicious(V12), !Suspicious(V60))$, do not raise an alarm to a human operator $(!Alarm(V12), !Alarm(V60))$ and are not involved in a rendezvous incident $(!Rendezvous(V60, V12), !Rendezvous(V12, V60))$. On the other hand, the second entry represents two vessels $(V10, V22)$ that: are suspicious $(Suspicious(V10), Suspicious(V22))$, raise an alarm to a human operator $(Alarm(V10), Alarm(V2))$ and are involved in a rendezvous incident $(Rendezvous(V10, V22), Rendezvous(V22, V10))$.

9.3.3 Experiments and Results

In order to train the MLN that comprises the formulas shown in Table 9.2 we used the Maximum Pseudo-Likelihood (MSL) learning implementation of the Alchemy framework which is based on optimization using a variant of the BFGS algorithm known as L-BFGS-B. In order to assess the quality of training i.e. how the trained model (MLN) generalises to an independent data set we use k-fold cross validation with $k = 10$. The rationale behind k-fold cross validation is that given a dataset we randomly divide (partition) it into k equal sized subsets. Each time, we train the model on data from $k-1$ subsets (training set) and test it on the data from the remainder subset (validation set). This process is performed repeatedly for a total of k times with each of the k subsets being used for testing (validation) only once.

#	Weight	Formula
1	—	$Proximity(x,y) \Leftrightarrow Proximity(x,v)$.
2	—	$Rendezvous(x,y) \Leftrightarrow Rendezvous(x,v)$.
3	-1.61618	$Stopped(x) \wedge IsIn(x,Harbour) \Rightarrow Suspicious(x)$
4	-1.48836	$Stopped(x) \wedge IsIn(x,NearCoast) \Rightarrow Suspicious(x)$
5	1.92031	$Stopped(x) \wedge IsIn(x,OpenSea) \Rightarrow Suspicious(x)$
6	2.00784	$Stopped(x) \wedge IsIn(x,IntWaters) \Rightarrow Suspicious(x)$
7	1.87642	$!AIS(x) \Rightarrow Alarm(x)$
8	0.689701	$AIS(x) \Rightarrow !Alarm(x)$
9	1.99037	$!InsideCorridor(x) \Rightarrow Suspicious(x)$
10	1.15087	$InsideCorridor(x) \Rightarrow !Suspicious(x)$
11	1.50073	$Humint(x,Smuggling) \Rightarrow Suspicious(x)$
12	-0.66122	$Humint(x,Clear) \Rightarrow Suspicious(x)$
13	1.38086	$Suspicious(x) \Rightarrow Alarm(x)$
14	1.03365	$!Suspicious(x) \Rightarrow !Alarm(x)$
15	—	$IsIn(x,z) \wedge (z! = zp) \Rightarrow !IsIn(x,zp)$.
16	1.22971	$Suspicious(x) \wedge Suspicious(y) \wedge Proximity(x,y) \Rightarrow Rendezvous(x,y)$
17	0.307675	$(!Stopped(x) \vee !Stopped(y)) \Rightarrow !Rendezvous(x,y)$
18	0.6556	$!Proximity(x,y) \Rightarrow !Rendezvous(x,y)$

Table 9.4. Trained MLN with one of the 10 folds.

Table 9.4 illustrates the outcome of training the MLN with one of the 10 folds.
For each validation set we use the MC-SAT algorithm implemented in the Alchemy framework to perform approximate probabilistic inference querying for the probability of: i) suspicious behaviours, ii) an alarm being raised and iii) rendezvous incidents, that is, $P(Suspicious(V_i)|M_{L,C}))$, $P(Alarm(V_i)|M_{L,C})$ and $P(Rendezvous(V_i,V_j)|M_{L,C}))$ respectively. The maximum number of steps chosen for the MC-SAT algorithm is 5000. For each of the aforementioned three queries we produce $k = 10$ Receiver Operating Characteristic (ROC) curves and calculate $k = 10$ Areas Under the Curve (AUC). ROC curves are graphs commonly used for assessing the performance of binary classifiers by plotting the True Positive Rate (TPR) of a (binary) classifier against its False Positive Rate (FPR) under varying thresholds. In our case, the varying thresholds are probabilities outputted by the queries. Regarding the AUC, it represents how well the classifier separates the two classes. At the end of the validation process as a whole, the validation results for each query, i.e. k ROC curves and k AUC for each query, are averaged to produce a single mean (average) ROC curve and a single mean (average) AUC metric. Figures 9.1, 9.2 and 9.3 illustrate the ROC curves and the AUC for each of the $k = 10$ validation sets as well as the mean ROC curve and the mean AUC for the three aforementioned queries.

Let us first interpret our findings with respect to the performance of the MLN in correctly classifying vessels as being suspicious or non suspicious. As can be observed by looking at the left graph of Figure 9.1, AUC ranges between 0.926 and 0.973 among the ten different validation folds with a mean of 0.952 (right graph). This means that the MLN, on average, has a 95.2% probability of ranking a randomly chosen suspicious vessel higher than a randomly chosen non suspicious vessel. An AUC value of one means that the classifier is perfect since this corresponds to a TPR of one and an FPR of zero while an AUC

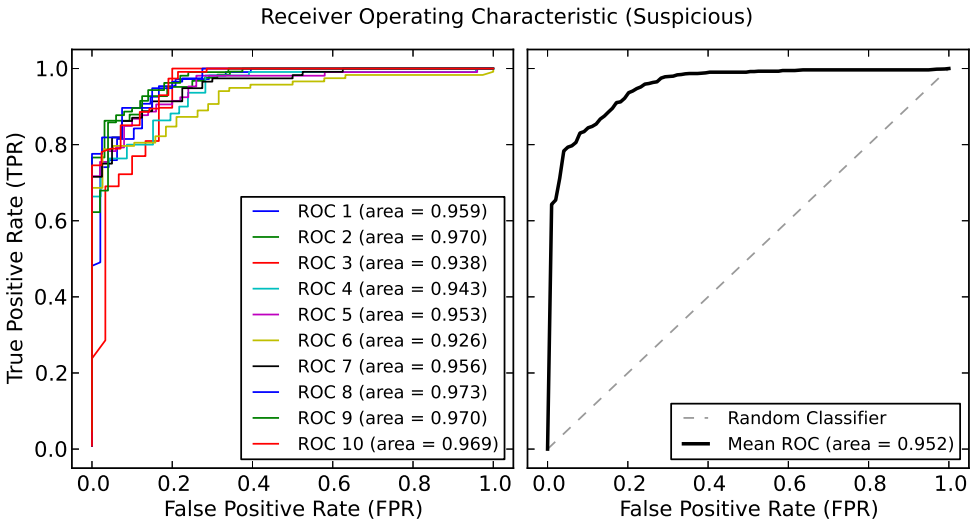


Figure 9.1. Performance of the MLN in classifying vessels as being suspicious or non suspicious. ROC curves and AUC for the 10-fold cross validation are illustrated in the left graph. Mean ROC curve and mean AUC are illustrated in the right graph.

value of 0.5 corresponds to the performance of a binary classifier that assigns classes at random (random classifier).

Regarding the performance of the MLN in correctly classifying (identifying) vessels that should or should not raise an alarm to a human operator shown in Figure 9.2, the AUC ranges between 0.761 and 0.854 among the ten different validation folds (left graph) with a mean of 0.816 (right graph). In addition, we can see that TPRs of one are achieved for

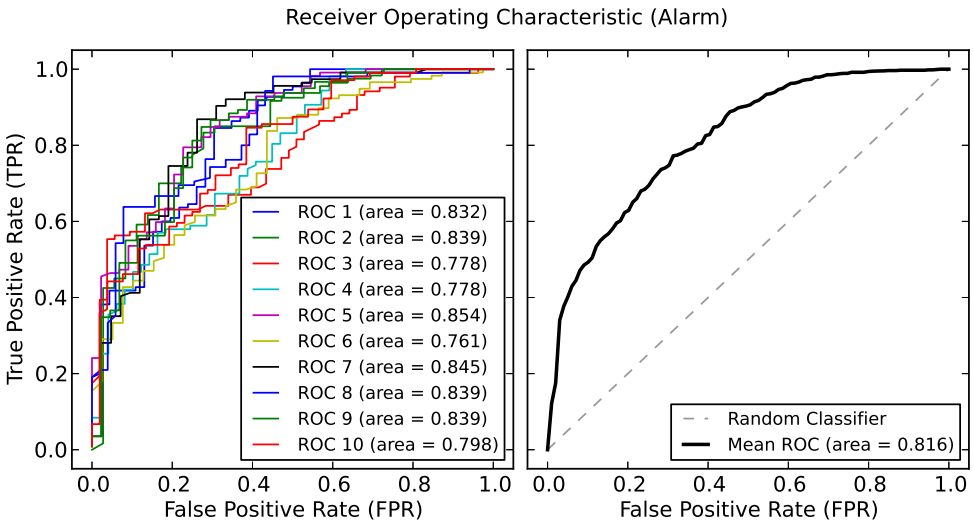


Figure 9.2. Performance of the MLN in identifying vessels that should raise or should not raise an alarm to a human operator. ROC curves and AUC for the 10-fold cross validation are illustrated in the left graph. Mean ROC curve and mean AUC are illustrated in the right graph.

higher values of FPRs when compared to the findings of Figure 9.1. Even though this performance is worse it is still very good and significantly better compared to a random classifier.

Finally, with regards to the performance of the MLN in correctly classifying (identify-

ing) pairs of vessels that are rendezvousing or not rendezvousing shown in Figure 9.3, the AUC ranges between 0.933 and 0.998 among the ten different validation folds (left graph) with a mean of 0.97 (right graph). The performance is similar to the case of identifying

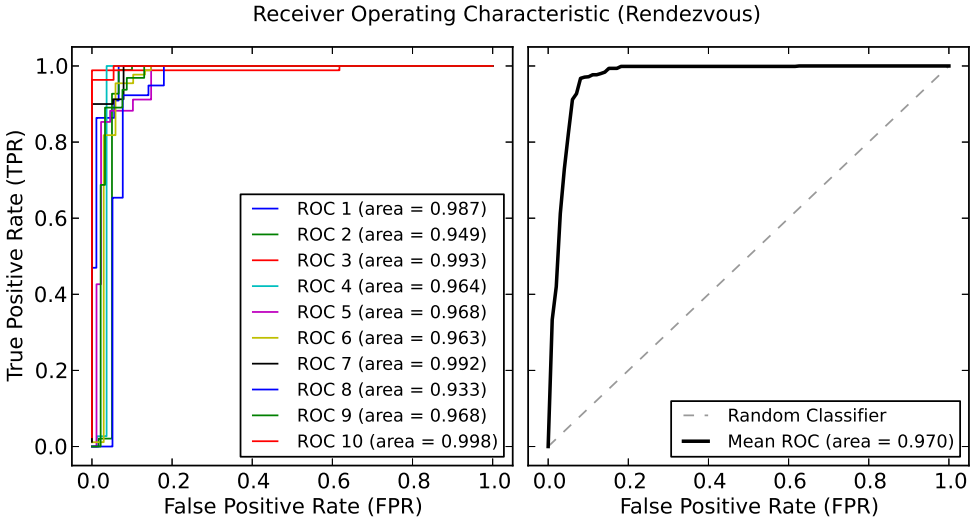


Figure 9.3. Performance of the MLN in classifying pairs of vessels rendezvousing or not rendezvousing. ROC curves and AUC for the 10-fold cross validation are illustrated in the left graph. Mean ROC curve and mean AUC are illustrated in the right graph.

suspicious vessels but slightly better and much better when compared to the case of identifying vessels that should raise an alarm to a human operator. In addition we can see that, in general, TPRs of one are achieved for lower values of FPRs compared to both previous cases.

Recall from Chapter 5, Section 5.2, that context plays a key role in interpreting a situation correctly. In order to investigate the impact of context, we used 10-fold cross validation but this time we omitted the predicates that represent context, i.e $IsIn(x,z)$ and $InsideCorridor(x)$, from the validation sets and performed the same queries as before. Training the MLN was performed in exactly the same way as before, that is, with the predicates that represent context present in the training data. However, the exclusion of the context predicates from the validation sets, forces the MLN to interpret the situations without contextual information. Figures 9.4, 9.5 and 9.6 illustrate the performance of the MLN in: i) classifying vessels as being suspicious or non suspicious, ii) identifying vessels that should or should not raise an alarm to a human operator and iii) classifying pairs of vessels rendezvousing or not rendezvousing, respectively, under context-free conditions.

As can be observed by looking at Figure 9.4 the AUC ranges between 0.782 and 0.899 (left graph) with an average of 0.834 (right graph). This constitutes a 0.118 drop on the mean AUC metric or in other words an 11.8% drop in the probability of the MLN ranking a randomly chosen suspicious vessel higher than a randomly chosen non suspicious vessel.

Regarding the performance of the network with respect to identifying vessels that should raise or should not raise an alarm to a human operator under context-free conditions (see Figure 9.5), the AUC ranges between 0.717 and 0.851 (left graph) with a mean of 0.789

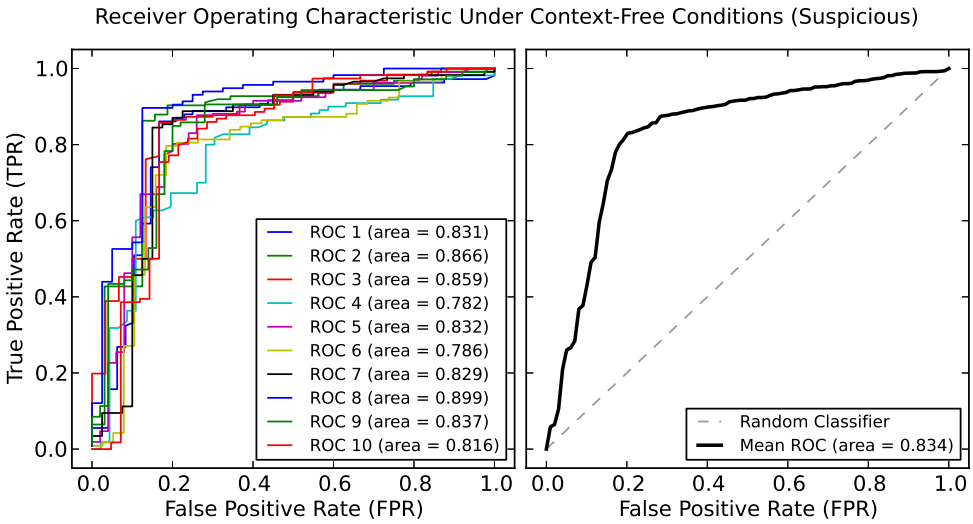


Figure 9.4. Performance of the MLN in classifying vessels as being suspicious or non suspicious, under context-free conditions. ROC curves and AUC for the 10-fold cross validation are illustrated in the left graph. Mean ROC curve and mean AUC are illustrated in the right graph.

(right graph). This represents a 2.7% drop in the probability of the MLN ranking a ran-

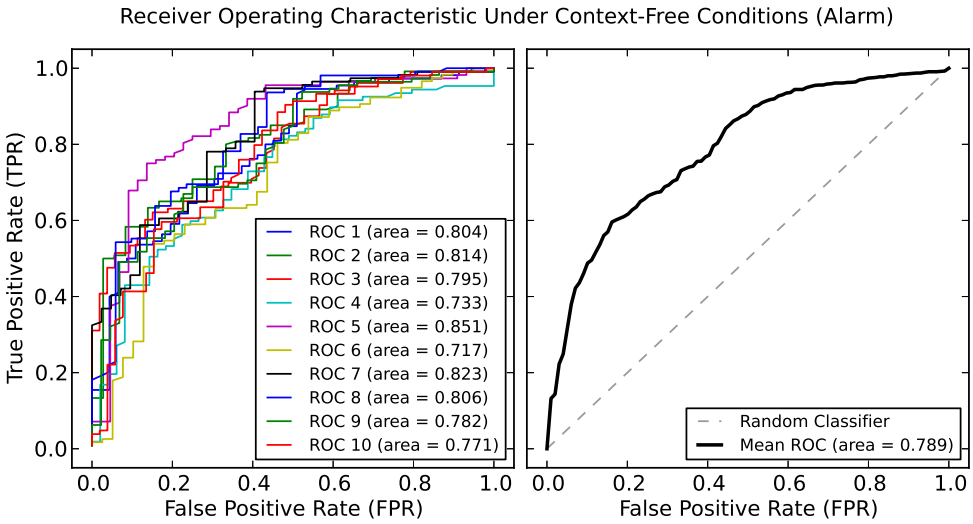


Figure 9.5. Performance of the MLN in identifying vessels that should raise or should not raise an alarm to a human operator, under context-free conditions. ROC curves and AUC for the 10-fold cross validation are illustrated in the left graph. Mean ROC curve and mean AUC are illustrated in the right graph.

domly chosen vessel that should raise an alarm higher than a randomly chosen vessel that should not do so. A question that naturally comes to mind is why this drop is lower than in the case of classifying suspicious vessels. This is explained by the fact that in the case of classifying a vessel as being suspicious or not, contextual information impacts indicator formulas much more than in the case of an alarm. More specifically, there is high uncertainty in the network arising from formulas #3-#6, #9, #10 (see Table 9.4) with only two formulas (#11, #12) to reduce uncertainty since human intelligence reports are not excluded from the validation sets. On the other hand, despite the fact that raising an alarm is affected

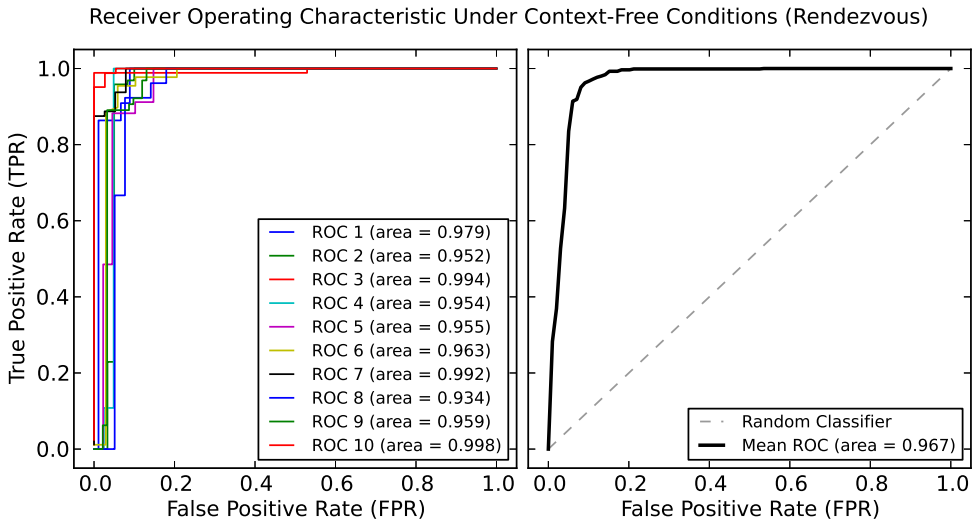


Figure 9.6. Performance of the MLN in classifying pairs of vessels rendezvousing or not rendezvousing, under context-free conditions. ROC curves and AUC for the 10-fold cross validation are illustrated in the left graph. Mean ROC curve and mean AUC are illustrated in the right graph.

by the outcome of the aforementioned classification, formulas #7, #8 reduce uncertainty since observations with respect to the AIS of vessels are present in the validation sets.

As far as the performance of the MLN with respect to classifying (identifying) pairs of vessels rendezvousing or not rendezvousing (under context-free conditions) is concerned, the AUC ranges between 0.934 and 0.998 with an average of 0.967 (see Figure 9.6). Compared to the performance of the network when contextual information was provided, there is a slight drop of 0.3%. This, only slight, decrease in performance is due to the fact that despite formula #16 being impacted by increased misclassifications of vessels not being suspicious, formulas #17, #18 help disambiguate the situation since the validation sets are not stripped of their $Stopped(x)$, $Stopped(y)$ and $Proximity(x,y)$ predicates.

Finally, we looked into the outcomes of probabilistic inference when querying for $P(Suspicious(V_i)|M_{L,C}))$, $P(Alarm(V_i)|M_{L,C})$, $P(Rendezvous(V_i,V_j)|M_{L,C}))$ and found that irrespective of whether a situation is assessed correctly or not, the lack of contextual information contributes towards incorrect assessments. Table 9.5 illustrates two exemplar evidence sets with evidence for two vessels: $V124$ and $V496$. The first set contains contextual information while the second is generated by stripping the contextual information from the first one. The ground truth for those evidence sets is of course the same. The outcomes of $P(Suspicious(V_i)|M_{L,C}))$ and $P(Alarm(V_i)|M_{L,C})$ for vessels $V124$ and $V496$, i.e. $V_i = \{V124, V496\}$, are illustrated in Table 9.6. As can be observed, when querying for $P(Suspicious(V124)|M_{L,C}))$ and $P(Alarm(V124)|M_{L,C}))$ in the presence of contextual information, the assessment of the situation is incorrect. However, this incorrect assessment is further enhanced by the lack of contextual information since there is a drop in the probabilities of the predicates that describe the situation correctly. On the other hand, when querying for $P(Suspicious(V496)|M_{L,C}))$ and $P(Alarm(V496)|M_{L,C}))$ in the presence and in the absence of contextual information, the assessment of the situation is correct. Nev-

Evidence Set	1	2
Ground Atoms	$Stopped(V496)$	$Stopped(V496)$
	$Stopped(V124)$	$Stopped(V124)$
	$IsIn(V496, OpenSea)$	—
	$IsIn(V124, IntWaters)$	—
	$!InsideCorridor(V496)$	—
	$Humint(V496, Smuggling)$	$Humint(V496, Smuggling)$
	$InsideCorridor(V124)$	—
	$Humint(V124, Clear)$	$Humint(V124, Clear)$
	$!AIS(V496)$	$!AIS(V496)$
	$AIS(V12496)$	$AIS(V124)$
	$Proximity(V496, V124)$	$Proximity(V496, V124)$
	$Proximity(V124, V496)$	$Proximity(V124, V496)$
Ground Truth	$Suspicious(V496)$	$Suspicious(V496)$
	$Suspicious(V124)$	$Suspicious(V124)$
	$Alarm(V496)$	$Alarm(V496)$
	$Alarm(V124)$	$Alarm(V124)$
	$Rendezvous(V496, V124)$	$Rendezvous(V496, V124)$
	$Rendezvous(V124, V496)$	$Rendezvous(V124, V496)$

Table 9.5. Two exemplar evidence sets with evidence for two vessels: $V124$ and $V496$. The first set contains contextual information illustrated in red while the second set is generated by stripping all contextual information from the first set. Also illustrated, are the ground truth situations that correspond to the evidence.

Query	**Probability (Context)**	**Probability (Context-Free)**	**Ground Truth**
$P(Suspicious(V124)\|M_{L,C}))$	0.385	0.128	$Suspicious(V124)$
$P(Suspicious(V496)\|M_{L,C}))$	0.995	0.792	$Suspicious(V496)$
$P(Alarm(V124)\|M_{L,C}))$	0.374	0.208	$Alarm(V124)$
$P(Alarm(V496)\|M_{L,C}))$	0.952	0.893	$Alarm(V496)$

Table 9.6. Outcomes of probabilistic inference when querying for whether vessels $V124$, $V496$ are suspicious and for whether an alarm should be raised for those vessels. That is, given the evidence found in Table 9.5.

ertheless, in the absence of contextual information there is higher uncertainty since the probability of the predicates that describe the situation correctly is reduced. This is also the case when querying for $P(Rendezvous(V124, V496)|M_{L,C}))$ as can be observed by looking at Table 9.7.

Query	**Probability (Context)**	**Probability (Context-Free)**	**Ground Truth**
$P(Rendezvous(124, 496)\|M_{L,C}))$	0.823	0.708	$Rendezvous(V124, V496)$

Table 9.7. Outcome of probabilistic inference when querying for the existence of a rendezvous incident between vessels $V124$, $V496$ given the evidence found in Table 9.5.

9.4 Summary and Conclusions

This chapter was concerned with training and evaluating Markov logic networks for maritime situation awareness. As such, this chapter makes the following contributions:

- The utilization of the maximum pseudo-likelihood weight learning method for training MLNs for maritime situation awareness based on data instead of hard-coding weights.
- The evaluation of the performance of an MLN trained with data both in the presence and absence of contextual information in identifying: i) suspicious vessels at sea, ii) vessels that should raise an alarm to a human operator and iii) vessels rendezvousing in order to get involved in illegal activities.

Let us now elaborate on the above list.

Regarding training, we presented two approaches for learning the weights of MLNs which are based on maximum likelihood learning and maximum pseudo-likelihood learning. As opposed to maximum likelihood learning, maximum pseudo-likelihood learning does not require inference at each step of the process which makes the process slow. At the same time it enables us avoid the problem of intractability. As such we utilised the maximum pseudo-likelihood approach for weight learning. The experimental results demonstrate that training Markov logic networks for maritime situation awareness is a very promising alternative to hard-coding the network weights. In our view, this is a very important feature because it constitutes a step towards minimizing the need for involving human experts for encoding prior knowledge into the system. As such, prior knowledge is built directly from the source, i.e. the data, and potentially contradicting suggestions from human experts need not be manually fused into the model. Instead, the process of training itself, deals with contradictions potentially present in the data, automatically. The experimental results also demonstrate the efficiency of Markov logic networks in fusing information from diverse sources and building maritime situation awareness correctly under uncertainty. Moreover, our findings suggest that contextual information (context) is an important factor towards correct interpretation of a situation. Overall, Markov logic networks are very powerful models which when used as the basis for maritime situation awareness systems can provide human operators decision support mechanisms of high value.

Part IV

Conclusions

Chapter 10

Conclusions

This thesis was concerned with autonomous underwater operations with AUVs and maritime situation awareness systems with a special interest in their utilization for enhancing maritime defence and security respectively. Regarding autonomous underwater operations, we presented a generic, easy to understand and extend, ontology-based framework complemented by mission planning and execution system. The framework can be deployed in AUVs aiming at increasing their persistence in maintaining autonomy in the presence of mission requirement changes and vehicle subsystem failures during such operations.

In the context of maritime defence, the framework was deployed in an AUV intended for MCM in simulation. In that respect, an initial set of experiments demonstrated the ability of the framework in enabling the AUV to plan and execute MCM missions successfully under three different high level mission priorities. That is, reacquiring and inspecting mines in: i) an energy-efficient manner, ii) an energy-efficient plus probability-efficient manner, iii) an energy-efficient plus entropy-efficient manner. This constitutes an advancement from the standard MCM missions where energy consumption/execution time are prioritized when trying to fulfil mission goals. Experimental results demonstrate that when considering entropy and probability in addition to energy the vehicle consumes more energy and requires more time to execute its mission. However, this is compensated in terms of uncertainty reduction in the first case and in terms of certainty exploitation in the latter. The power of the ontology-based knowledge representation and reasoning is demonstrated through the vehicle's persistence in maintaining autonomy in two cases.

First, in communicating high level mission priority changes to the vehicle during mission execution to which the vehicle responded successfully with adaptive planning and continuation of the mission given pending goals. As was mentioned earlier in this thesis, this is a very useful feature on the grounds that it facilitates the need for adaptation due to real time mission requirement changes. Most importantly, without the vehicle having to terminate its mission and be redeployed anew with the new requirements.

Second, in the presence of vehicle component faults (subsystem failures), where the vehicle demonstrated an adaptive behaviour in order to successfully recover from such faults and carry on with its mission. More specifically, the vehicle benefits from an ontological representation based on which the interdependencies amongst vehicle components, capa-

bilities and actions are captured and vehicle's components health status is modelled. This representation is complemented with semantically equivalent capabilities and (semantically equivalent) actions subject to the existence of redundant functional components for the same type of capability and the existence of semantically equivalent capabilities required for the same type of action respectively. Based on this knowledge and reasoning modules written in Prolog the vehicle is able to determine its internal state in the form capabilities and actions dynamically and at runtime and achieve recovery though adaptation when component faults affect mission execution. The adaptive behaviour in this context comes in two forms: adaptive planning and adaptive execution. The vehicle resorts to adaptive planning when the current plan under execution can no longer be executed due to faults in critical components that deprive the vehicle of all the necessary capabilities and actions to do so. In that respect, experimental results demonstrate the efficiency of the framework in enabling the vehicle to adapt to the new reality by devising new plans that allow it to continue with remaining mission goals. This is but a highly desirable feature on the grounds that the vehicle does not have to abort its mission altogether. Alternatively, the vehicle resorts to adaptive execution when there are redundancies for a faulty component whose usage can be substituted for the usage of a functional healthy one that enables the vehicle take advantage of the existence of the semantically equivalent capabilities and actions that it offers. Adaptive execution is always prioritized over adaptive planning not only as a means to save computational resources but also as a means to satisfy all existing mission goals. Ultimately, seeking and exploiting alternatives to satisfy mission goals is not only indicative of persistence but also of increased intelligence.

Moving on to maritime situation awareness in the context of maritime security, an overview of the field was presented, including a review of current approaches for building awareness through systems. Such systems aim to support human operators in their challenging work of dealing with high volumes of potentially contradicting information which should be fused efficiently to facilitate reliable decision making and action under potentially multiple interpretations of a situation. Focusing on maritime situation awareness built with Markov logic networks, we presented an approach based on maximum pseudo-likelihood for training them and evaluated their performance using 10-fold cross validation. Experimental results demonstrated that training Markov logic networks intended for maritime situation awareness is a very promising alternative to hard-coding network weights. As was explained earlier in this thesis, this is a very important feature on the grounds that it constitutes a step forward towards disengaging human experts from transferring their prior knowledge into the system. Instead, prior knowledge is built directly from data and any potential contradictions present in the data are addressed during the training process automatically. Moreover, performance evaluation results suggest that Markov logic networks constitute a mechanism which offers efficient fusion of information from various sources and achieve high success rates in identifying and disambiguating between abnormal and normal vessel activities under uncertainty. This is due to their underlying structure which unifies first-order logic which is ideal for representing the relations among entities

in a domain with Markov networks for uncertainty management. In addition, experimental findings suggest that context is an important factor, the lack of which affects the interpretation correctness of an unfolding situation. Altogether, utilizing Markov logic networks for building maritime situation awareness systems can provide human operators with high value decision support mechanisms.

10.1 Future Work Suggestions

The work presented in this thesis could be extended in several ways as well as in several areas. Currently, the persistently autonomous operations framework assumes the existence of a fault detection system that is able to detect faults appearing on vehicle components. Given this information an AUV can reason about its internal state in terms of available capabilities and actions. In the future it would be beneficial to develop or integrate an existing fault detection system so that the framework can offer a more complete solution in that respect, promoting wider adoption and operability of the framework. An additional future direction would be to address the issue of gradual degradation of the performance of some components. A representative example is vehicle thrusters which in some cases instead of becoming non-functional, they demonstrate a degradation of their performance by operating at reduced thrust capacities. Taking into consideration and reasoning about performance degradation however is not enough. It also requires controllers that are capable of adjusting the thrust of remainder thrusters so that they compensate for such degradation, enabling the vehicle to continue operating as appropriate even at reduced speeds.

Another interesting future direction to follow would be the extension of the framework to support cooperative multi-AUV operations. One way to achieve this would be the development and integration of a task allocation mechanism capable of utilizing the reasoning outcomes that assess each AUV's capabilities as well as topological information to assigning tasks dynamically. That is, in the event that a vehicle that had initially been assigned a task faces a problem with a critical component for this task, the allocation mechanism would assign the task to another vehicle subject to that vehicle having the capability to undertake it and execute it successfully. This would of course require adjustments to the framework based on which each vehicle would be able to share its knowledge about the world with the other vehicles so that the new vehicle undertaking the task is up to date.

Except for the suggested future work with respect to improving and extending the persistently autonomous operations framework, future work can be put into examining and evaluating structure learning for Markov logic networks in the context of providing maritime situation awareness for maritime security. This would be beneficial on the grounds that it will further disengage human experts from having to transfer their prior domain knowledge into the system since relations amongst entities would be learned directly from data. However, the efficiency of such an approach would have to be tested through comparative experiments.

Finally, future work should focus on deploying the persistently autonomous operations

framework on a real vehicle where it can be tested under real world conditions. Similarly, it would be beneficial to acquire real world data for training and evaluating Markov logic networks for maritime situation awareness.

Bibliography

[1] K. B. Laskey and P. C. Costa, "Multi-entity bayesian networks without multi-tears," 2006.

[2] M. Tenorth, *Knowledge Processing for Autonomous Robots*. PhD thesis, Technische Universität München, 2011.

[3] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.

[4] G. Papadimitriou, Z. Saigol, and D. M. Lane, "Enabling fault recovery and adaptation in mine-countermeasures missions using ontologies," in *OCEANS 2015-Genova*, pp. 1–7, IEEE, 2015.

[5] L. Snidaro, I. Visentini, and K. Bryan, "Fusing uncertain knowledge and evidence for maritime situational awareness via markov logic networks," *Information Fusion*, vol. 21, pp. 159–172, 2015.

[6] C. Office and G. B. Parliament, *Securing Britain in an age of uncertainty: the strategic defence and security review*, vol. 7948. The Stationery Office, 2010.

[7] M. Glandrup, "Improving situation awareness in the maritime domain," in *Situation Awareness with Systems of Systems*, pp. 21–38, Springer, 2013.

[8] J. Gómez-Romero, M. A. Serrano, J. García, J. M. Molina, and G. Rogova, "Context-based multi-level information fusion for harbor surveillance," *Information Fusion*, vol. 21, pp. 173–186, 2015.

[9] M. Nilsson, J. van Laere, T. Ziemke, and J. Edlund, "Extracting rules from expert operators to support situation awareness in maritime surveillance," in *Information Fusion, 2008 11th International Conference on*, pp. 1–8, IEEE, 2008.

[10] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, pp. 107–136, 2006.

[11] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[12] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[13] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM (JACM)*, vol. 7, no. 3, pp. 201–215, 1960.

[14] B. Selman, H. A. Kautz, and B. Cohen, "Noise strategies for improving local search," in *AAAI*, vol. 94, pp. 337–343, 1994.

[15] D. McAllester, B. Selman, and H. Kautz, "Evidence for invariants in local search," in *AAAI/IAAI*, pp. 321–326, 1997.

[16] A. J. Robinson and A. Voronkov, *Handbook of automated reasoning*, vol. 1. Elsevier, 2001.

[17] P. H. Salus, *Handbook of Programming Languages (HPL), Volume 4: Functional and Logic Programming Languages*. Sams, 1998.

[18] D. M. Gabbay and J. A. Robinson, *Handbook of Logic in Artificial Intelligence and Logic Programming: Volume 5: Logic Programming*. Clarendon Press, 1998.

[19] E. W. Elcock, "Absys: The first logic programming language—a retrospective and commentary," *J. Log. Program.*, vol. 9, pp. 1–17, July 1990.

[20] H. Gallaire, J. Minker, and J.-M. Nicolas, "Logic and databases: A deductive approach," *ACM Computing Surveys (CSUR)*, vol. 16, no. 2, pp. 153–185, 1984.

[21] S. S. Huang, T. J. Green, and B. T. Loo, "Datalog and emerging applications: An interactive tutorial," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, (New York, NY, USA), pp. 1213–1216, ACM, 2011.

[22] W. Chen, M. Kifer, and D. S. Warren, "Hilog: A foundation for higher-order logic programming," *The Journal of Logic Programming*, vol. 15, no. 3, pp. 187–230, 1993.

[23] R. A. Kowalski, "The early years of logic programming," *Communications of the ACM*, vol. 31, no. 1, pp. 38–43, 1988.

[24] A. Colmerauer and P. Roussel, "The birth of prolog," in *History of programming languages—II*, pp. 331–367, ACM, 1996.

[25] Z. Somogyi, F. J. Henderson, and T. C. Conway, "Mercury, an efficient purely declarative logic programming language," *Australian Computer Science Communications*, vol. 17, pp. 499–512, 1995.

[26] P. Norvig, "Techniques for automatic memoization with applications to context-free parsing," *Computational Linguistics*, vol. 17, no. 1, pp. 91–98, 1991.

[27] U. A. Acar, G. E. Blelloch, and R. Harper, *Selective memoization*, vol. 38. ACM, 2003.

[28] M. Johnson, "Memoization in constraint logic programming," *arXiv preprint cmp-lg/9404005*, 1994.

[29] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds., *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.

[30] M. Krötzsch, F. Simancik, and I. Horrocks, "A description logic primer," *arXiv preprint arXiv:1201.4089*, 2012.

[31] M. Schmidt-Schaubßand G. Smolka, "Attributive concept descriptions with complements," *Artif. Intell.*, vol. 48, pp. 1–26, Feb. 1991.

[32] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[33] J. D. Park, "Map complexity results and approximation methods," in *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pp. 388–396, Morgan Kaufmann Publishers Inc., 2002.

[34] J. Kwisthout, "Most probable explanations in bayesian networks: Complexity and tractability," *International Journal of Approximate Reasoning*, vol. 52, no. 9, pp. 1452–1469, 2011.

[35] H. Chan and A. Darwiche, "On the robustness of most probable explanations," *arXiv preprint arXiv:1206.6819*, 2012.

[36] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.

[37] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[38] D. Geiger, T. S. Verma, and J. Pearl, "d-separation: From theorems to algorithms," *arXiv preprint arXiv:1304.1505*, 2013.

[39] J. Kwisthout, H. L. Bodlaender, and L. C. van der Gaag, "The necessity of bounded treewidth for efficient inference in bayesian networks.," in *ECAI*, vol. 215, pp. 237–242, 2010.

[40] J. H. Kim and J. Pearl, "A computational model for causal and diagnostic reasoning in inference systems," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'83, (San Francisco, CA, USA), pp. 190–193, Morgan Kaufmann Publishers Inc., 1983.

[41] N. L. Zhang and D. Poole, "A simple approach to bayesian network computations," in *Proc. of the Tenth Canadian Conference on Artificial Intelligence*, 1994.

[42] S.-S. Leu and C.-M. Chang, "Bayesian-network-based safety risk assessment for steel construction projects," *Accident Analysis & Prevention*, vol. 54, pp. 122–133, 2013.

[43] N. Khakzad, F. Khan, and P. Amyotte, "Dynamic safety analysis of process systems by mapping bow-tie into bayesian network," *Process Safety and Environmental Protection*, vol. 91, no. 1, pp. 46–53, 2013.

[44] M. Hänninen, O. A. V. Banda, and P. Kujala, "Bayesian network model of maritime safety management," *Expert Systems with Applications*, vol. 41, no. 17, pp. 7837–7846, 2014.

[45] M. Velikova, J. T. van Scheltinga, P. J. Lucas, and M. Spaanderman, "Exploiting causal functional relationships in bayesian network modelling for personalised healthcare," *International Journal of Approximate Reasoning*, vol. 55, no. 1, pp. 59–73, 2014.

[46] F. L. Seixas, B. Zadrozny, J. Laks, A. Conci, and D. C. M. Saade, "A bayesian network decision model for supporting the diagnosis of dementia, alzheimer s disease and mild cognitive impairment," *Computers in biology and medicine*, vol. 51, pp. 140–158, 2014.

[47] S. Windecker, S. Stortecky, G. G. Stefanini, A. W. Rutjes, M. Di Nisio, M. G. Siletta, A. Maione, F. Alfonso, P. M. Clemmensen, J.-P. Collet, *et al.*, "Revascularisation versus medical treatment in patients with stable coronary artery disease: network meta-analysis," 2014.

[48] B. Cai, Y. Liu, Q. Fan, Y. Zhang, Z. Liu, S. Yu, and R. Ji, "Multi-source information fusion based fault diagnosis of ground-source heat pump using bayesian network," *Applied Energy*, vol. 114, pp. 1–9, 2014.

[49] Z. Yongli, H. Limin, and L. Jinling, "Bayesian networks-based approach for power systems fault diagnosis," *Power Delivery, IEEE Transactions on*, vol. 21, no. 2, pp. 634–639, 2006.

[50] C. Romessis and K. Mathioudakis, "Bayesian network approach for gas path fault diagnosis," *Journal of engineering for gas turbines and power*, vol. 128, no. 1, pp. 64–72, 2006.

[51] R. Cano, C. Sordo, and J. M. Gutiérrez, "Applications of bayesian networks in meteorology," in *Advances in Bayesian networks*, pp. 309–328, Springer, 2004.

[52] A. Ershadi, M. F. McCabe, J. P. Evans, G. Mariethoz, and D. Kavetski, "A bayesian analysis of sensible heat flux estimation: Quantifying uncertainty in meteorological forcing to improve model prediction," *Water Resources Research*, vol. 49, no. 5, pp. 2343–2358, 2013.

[53] T. Boneh, G. T. Weymouth, P. Newham, R. Potts, J. Bally, A. E. Nicholson, and K. B. Korb, "Fog forecasting for melbourne airport using a bayesian decision network," *Weather and Forecasting*, vol. 30, no. 5, pp. 1218–1233, 2015.

[54] C. Bielza and P. Larrañaga, "Bayesian networks in neuroscience: a survey," *Frontiers in computational neuroscience*, vol. 8, p. 131, 2014.

[55] H.-S. Park and S.-B. Cho, "A modular design of bayesian networks using expert knowledge: Context-aware home service robot," *Expert Systems with Applications*, vol. 39, no. 3, pp. 2629–2642, 2012.

[56] D. Kortenkamp and T. Weymouth, "Topological mapping for mobile robots using a combination of sonar and vision sensing," in *AAAI*, vol. 94, pp. 979–984, 1994.

[57] A. Joshi, T. C. Henderson, and W. Wang, "Robot cognition using bayesian symmetry networks.," in *ICAART (1)*, pp. 696–702, 2014.

[58] R. Kindermann, J. L. Snell, *et al.*, *Markov random fields and their applications*, vol. 1. American Mathematical Society Providence, RI, 1980.

[59] D. Kahle, T. Savitsky, S. Schnelle, and V. Cevher, "Junction tree algorithm," *STAT*, vol. 631, 2008.

[60] B. J. Frey and D. J. MacKay, "A revolution: Belief propagation in graphs with cycles," *Advances in neural information processing systems*, pp. 479–485, 1998.

[61] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 498–519, 2001.

[62] Y. Weiss, "Correctness of local probability propagation in graphical models with loops," *Neural computation*, vol. 12, no. 1, pp. 1–41, 2000.

[63] J. Besag, "On the statistical analysis of dirty pictures," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 259–302, 1986.

[64] P. Brémaud, *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, vol. 31. Springer Science & Business Media, 2013.

[65] P. Kohli and P. H. Torr, "Dynamic graph cuts for efficient inference in markov random fields," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 12, pp. 2079–2088, 2007.

[66] Y. Boykov and G. Funka-Lea, "Graph cuts and efficient nd image segmentation," *International journal of computer vision*, vol. 70, no. 2, pp. 109–131, 2006.

[67] D. Singaraju, L. Grady, and R. Vidal, "P-brush: Continuous valued mrfs with normed pairwise distributions for image segmentation," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 1303–1310, IEEE, 2009.

[68] A. Shekhovtsov, I. Kovtun, and V. Hlaváč, "Efficient mrf deformation model for non-rigid image matching," *Computer Vision and Image Understanding*, vol. 112, no. 1, pp. 91–99, 2008.

[69] B. Glocker, N. Komodakis, G. Tziritas, N. Navab, and N. Paragios, "Dense image registration through mrfs and efficient linear programming," *Medical image analysis*, vol. 12, no. 6, pp. 731–741, 2008.

[70] C. Liu, J. Yuen, and A. Torralba, "Sift flow: Dense correspondence across scenes and its applications," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 5, pp. 978–994, 2011.

[71] T. Goldstein, X. Bresson, and S. Osher, "Global minimization of markov random fields with applications to optical flow," *Inverse Problems and Imaging*, vol. 6, no. 4, pp. 623–644, 2012.

[72] R. Zhang, C. A. Bouman, J.-B. Thibault, and K. D. Sauer, "Gaussian mixture markov random field for image denoising and reconstruction," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pp. 1089–1092, IEEE, 2013.

[73] K. Lekadir, M. Lange, V. A. Zimmer, C. Hoogendoorn, and A. F. Frangi, "Statistically-driven 3d fiber reconstruction and denoising from multi-slice cardiac dti using a markov random field model," *Medical image analysis*, vol. 27, pp. 105–116, 2016.

[74] J. Diebel and S. Thrun, "An application of markov random fields to range sensing," in *NIPS*, vol. 5, pp. 291–298, 2005.

[75] D. Metzler and W. B. Croft, "A markov random field model for term dependencies," in *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, (New York, NY, USA), pp. 472–479, ACM, 2005.

[76] M. Lease, "An improved markov random field model for supporting verbose queries," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 476–483, ACM, 2009.

[77] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, (New York, NY, USA), pp. 57–66, ACM, 2001.

[78] E. Zheleva, L. Getoor, and S. Sarawagi, "Higher-order graphical models for classification in social and affiliation networks," in *NIPS Workshop on Networks Across Disciplines: Theory and Applications*, vol. 2, Citeseer, 2010.

[79] H. Li, A. Mukherjee, B. Liu, R. Kornfield, and S. Emery, "Detecting campaign promoters on twitter using markov random fields," in *Data Mining (ICDM), 2014 IEEE International Conference on*, pp. 290–299, IEEE, 2014.

[80] L. Getoor, *Introduction to statistical relational learning*. MIT press, 2007.

[81] D. Jain, S. Waldherr, and M. Beetz, "Bayesian logic networks," *IAS Group, Fakultät für Informatik, Technische Universität München, Tech. Rep*, 2009.

[82] D. Fierens, H. Blockeel, J. Ramon, and M. Bruynooghe, "Logical bayesian networks," in *Third workshop on multi-relational data mining*, pp. 19–30, Citeseer, 2004.

[83] D. Fierens, H. Blockeel, M. Bruynooghe, and J. Ramon, "Logical bayesian networks and their relation to other probabilistic logical models," in *Inductive Logic Programming*, pp. 121–135, Springer, 2005.

[84] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer, "Learning probabilistic relational models," in *Relational data mining*, pp. 307–335, Springer, 2001.

[85] T. Sommestad, M. Ekstedt, and P. Johnson, "A probabilistic relational model for security risk analysis," *Computers & security*, vol. 29, no. 6, pp. 659–679, 2010.

[86] K. B. Laskey, "Mebn: A language for first-order bayesian knowledge bases," *Artificial intelligence*, vol. 172, no. 2, pp. 140–178, 2008.

[87] C. Y. Park, K. B. Laskey, P. Costa, and S. Matsumoto, "Multi-entity bayesian networks learning in predictive situation awareness," tech. rep., DTIC Document, 2013.

[88] B. Taskar, P. Abbeel, M.-F. Wong, and D. Koller, "6 relational markov networks," *STATISTICAL RELATIONAL LEARNING*, p. 175, 2007.

[89] A. Jaimovich, O. Meshi, and N. Friedman, "Template based inference in symmetric relational markov random fields," *arXiv preprint arXiv:1206.5276*, 2012.

[90] D. Nyga, F. Balint-Benczedi, and M. Beetz, "Pr2 looking at thingsensemble learning for unstructured information processing with markov logic networks," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 3916–3923, IEEE, 2014.

[91] S. Sarkhel, D. Venugopal, P. Singla, and V. Gogate, "Lifted map inference for markov logic networks.," in *AISTATS*, pp. 859–867, 2014.

[92] S. Sarkhel, D. Venugopal, T. A. Pham, P. Singla, and V. Gogate, "Scalable training of markov logic networks using approximate counting," 2016.

[93] D. Lowd and P. Domingos, "Efficient weight learning for markov logic networks," in *Knowledge discovery in databases: PKDD 2007*, pp. 200–211, Springer, 2007.

[94] S. M. Mahoney and K. B. Laskey, "Constructing situation specific belief networks," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 370–378, Morgan Kaufmann Publishers Inc., 1998.

[95] R. Mateescu and R. Dechter, "Mixed deterministic and probabilistic networks," *Annals of mathematics and artificial intelligence*, vol. 54, no. 1-3, pp. 3–51, 2008.

[96] V. Gogate and R. Dechter, "Samplesearch: A scheme that searches for consistent samples," in *International Conference on Artificial Intelligence and Statistics*, pp. 147–154, 2007.

[97] M. Genesereth and N. Nilsson, *Logical foundations of artificial intelligence*. Morgan Kaufmann, 1987.

[98] P. Domingos and D. Lowd, "Markov logic: An interface layer for artificial intelligence," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 3, no. 1, pp. 1–155, 2009.

[99] H. Poon and P. Domingos, "Sound and efficient inference with probabilistic and deterministic dependencies," in *AAAI*, vol. 6, pp. 458–463, 2006.

[100] W. Wei, J. Erenrich, and B. Selman, "Towards efficient sampling: Exploiting random walk strategies," in *AAAI*, vol. 4, pp. 670–676, 2004.

[101] D. Jain, P. Maier, and G. Wylezich, "Markov logic as a modelling language for weighted constraint satisfaction problems," *Constraint Modelling and Reformulation (ModRef09)*, p. 60, 2009.

[102] H. Kautz, B. Selman, and Y. Jiang, "A general stochastic approach to solving problems with hard and soft constraints," *The Satisfiability Problem: Theory and Applications*, vol. 17, pp. 573–586, 1997.

[103] L. Mihalkova and M. Richardson, "Speeding up inference in statistical relational learning by clustering similar query literals.," in *ILP*, pp. 110–122, Springer, 2009.

[104] J. W. Shavlik and S. Natarajan, "Speeding up inference in markov logic networks by preprocessing to reduce the size of the resulting grounded network.," in *IJCAI*, vol. 9, pp. 1951–1956, 2009.

[105] F. Niu, C. Ré, A. Doan, and J. Shavlik, "Tuffy: Scaling up statistical inference in markov logic networks using an rdbms," *Proceedings of the VLDB Endowment*, vol. 4, no. 6, pp. 373–384, 2011.

[106] K. Beedkar, L. Del Corro, and R. Gemulla, "Fully parallel inference in markov logic networks.," in *BTW*, pp. 205–224, Citeseer, 2013.

[107] L. A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[108] R. Fagin, J. Y. Halpern, Y. Moses, and M. Vardi, *Reasoning about knowledge*. MIT press, 2004.

[109] N. Guarino, *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1st ed., 1998.

[110] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *Int. J. Hum.-Comput. Stud.*, vol. 43, pp. 907–928, Dec. 1995.

[111] W3C, "Resource description framework (rdf) model and syntax specification." http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, Accessed on 25/4/2016.

[112] W3C, "Rdf vocabulary description language 1.0: Rdf schema." http://www.w3.org/TR/rdf-schema/, Accessed on 25/4/2016.

[113] I. Horrocks, P. F. Patel-Schneider, and F. V. Harmelen, "From shiq and rdf to owl: The making of a web ontology language," *Journal of Web Semantics*, vol. 1, p. 2003, 2003.

[114] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, "Ontology language semantics and abstract syntax, w3c recommendation," 2004.

[115] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, "Owl 2: The next step for owl," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 4, pp. 309 – 322, 2008. ¡ce:title¿Semantic Web Challenge 2006/2007¡/ce:title¿.

[116] W3C, "OWL 2 web ontology language profiles – computational properties." `http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/#Computational_Properties`, 2012. Accessed on 23/08/2016.

[117] A. Pease, "Standard upper ontology knowledge interchange format." Unpublished language manual. Available at http://ontolog.cim3.net/file/resource/reference/SIGMA-kee/suo-kif.pdf, February 2004.

[118] I. Niles and A. Pease, "Towards a standard upper ontology," in *Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, (New York, NY, USA), pp. 2–9, ACM, 2001.

[119] S. E. Fahlman, "Marker-passing inference in the scone knowledge-base system," in *Proceedings of the First international conference on Knowledge Science, Engineering and Management*, KSEM'06, (Berlin, Heidelberg), pp. 114–126, Springer-Verlag, 2006.

[120] C. Matuszek, J. Cabral, M. Witbrock, and J. Deoliveira, "An introduction to the syntax and content of cyc," in *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pp. 44–49, 2006.

[121] I. 13250-2:2006, "Information technology – topic maps – part 2: Data model," tech. rep., International Organization for Standardization, Geneva, Switzerland, 2006.

[122] A. Marchetti, F. Ronzano, M. Tesconi, and S. Minutoli, "Formalizing knowledge by ontologies: Owl and kif," tech. rep., Istituto di Informatica e Telematica CNR, 2008.

[123] M. Kifer, "Rules and ontologies in f-logic," in *Reasoning Web* (N. Eisinger and J. Mauszyski, eds.), vol. 3564 of *Lecture Notes in Computer Science*, pp. 22–34, Springer Berlin Heidelberg, 2005.

[124] C. J. Matheus, M. M. Kokar, and K. Baclawski, "A core ontology for situation awareness," in *Proceedings of the Sixth International Conference of Information Fusion, 2003*, vol. 1, pp. 545–552, 2003.

[125] M. Tenorth and M. Beetz, "Knowrob – knowledge processing for autonomous personal robots," in *IROS'09 Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, pp. 4261–4266.

[126] S. Lemaignan, R. Ros, L. Mösenlechner, R. Alami, and M. Beetz, "ORO, a knowledge management platform for cognitive architectures in robotics," in *International Conference on Intelligent RObots and Systems*, 2010.

[127] G. Lortal, S. Dhouib, and S. Grard, "Integrating ontological domain knowledge into a robotic dsl," in *Models in Software Engineering* (J. Dingel and A. Solberg, eds.), vol. 6627 of *Lecture Notes in Computer Science*, pp. 401–414, Springer Berlin / Heidelberg, 2011.

[128] E. Miguelanez, P. Patron, K. E. Brown, Y. R. Petillot, and D. M. Lane, "Semantic knowledge-based framework to improve the situation awareness of autonomous underwater vehicles," *IEEE Trans. on Knowl. and Data Eng.*, vol. 23, pp. 759–773, May 2011.

[129] D. B. Lenat, "Cyc: A large-scale investment in knowledge infrastructure," *Commun. ACM*, vol. 38, pp. 33–38, Nov 1995.

[130] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, "SWI-Prolog," *Theory and Practice of Logic Programming*, vol. 12, no. 1-2, pp. 67–96, 2012.

[131] V. Vassiliadis, *A Web Ontology Language - OWL Library for [SWI] Prolog*. Release Notes.

[132] V. Vassiliadis, J. Wielemaker, and C. Mungall, "Processing owl2 ontologies using thea: An application of logic programming," in *PROCEEDINGS OF THE 5TH INTERNATIONAL WORKSHOP ON OWL: EXPERIENCES AND DIRECTIONS*, 2009.

[133] J. Wielemaker, G. Schreiber, and B. Wielinga, "Prolog-based infrastructure for RDF: Scalability and performance," in *Semantic Web - ISWC 2003* (Fensel, D and Sycara, K and Mylopoulos, J, ed.), vol. 2870 of *Lecture Notes In Computer Science*, pp. 644–658, 2003. 2nd International Semantic Web Conference, Sanibel, Florida, Oct 20-23, 2003.

[134] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.

[135] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[136] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[137] F. Giunchiglia and P. Traverso, "Planning as model checking," in *European Conference on Planning*, pp. 1–20, Springer, 1999.

[138] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "Weak, strong, and strong cyclic planning via symbolic model checking," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 35–84, 2003.

[139] M. Pistore, R. Bettin, and P. Traverso, "Symbolic techniques for planning with extended goals in non-deterministic domains," in *Sixth European Conference on Planning*, 2014.

[140] 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.

[141] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl-the planning domain definition language," 1998.

[142] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains.," *J. Artif. Intell. Res.(JAIR)*, vol. 20, pp. 61–124, 2003.

[143] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, "Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners," *Artificial Intelligence*, vol. 173, no. 5-6, pp. 619–668, 2009.

[144] J. Benton, A. J. Coles, and A. I. Coles, "Temporal planning with preferences and time-dependent continuous costs.," in *Proceedings of the Twenty Second International Conference on Automated Planning and Scheduling (ICAPS-12)*, June 2012.

[145] Z. A. Saigol, *Automated planning for hydrothermal vent prospecting using AUVs*. PhD thesis, University of Birmingham, 2011.

[146] M. Fox, D. Long, and D. Magazzeni, "Plan-based policy-learning for autonomous feature tracking.," in *ICAPS*, 2012.

[147] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen, "A deliberative architecture for auv control," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 1049–1054, IEEE, 2008.

[148] T. Bedrax-Weiss, C. McGann, A. Bachmann, W. Edington, and M. Iatauro, "Europa-2: User and contributor guide. nasa amesresearch center," tech. rep., Technical report, Moffett Field, CA, 2005.

[149] M. P. Brito, R. Lewis, N. Bose, P. Alexander, G. Griffiths, and J. Ferguson, "The role of adaptive mission planning and control in persistent autonomous underwater vehicles presence," in *Autonomous Underwater Vehicles (AUV), 2012 IEEE/OES*, pp. 1–9, IEEE, 2012.

[150] C. McGann, F. Py, K. Rajan, J. P. Ryan, and R. Henthorn, "Adaptive control for autonomous underwater vehicles.," in *AAAI*, pp. 1319–1324, 2008.

[151] D. M. Lane, F. Maurelli, T. Larkworthy, D. Caldwell, J. Salvi, M. Fox, and K. Kyriakopoulos, "Pandora: Persistent autonomy through learning, adaptation, observation and re-planning," *IFAC Proceedings Volumes*, vol. 45, no. 5, pp. 367–372, 2012.

[152] M. Cashmore, M. Fox, T. Larkworthy, D. Long, and D. Magazzeni, "Auv mission control via temporal planning," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 6535–6541, IEEE, 2014.

[153] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, "Rosplan: Planning in the robot operating system.," in *ICAPS*, pp. 333–341, 2015.

[154] O. Sidek and S. Quadri, "A review of data fusion models and systems," *International Journal of Image and Data Fusion*, vol. 3, no. 1, pp. 3–21, 2012.

[155] J. R. Boyd, "The essence of winning and losing," *Unpublished lecture notes*, 1996.

[156] E. Shahbazian, D. E. Blodgett, and P. Labbé, "The extended ooda model for data fusion systems," in *Intl Conf. on Info. Fusion-Fusion01*, 2001.

[157] A. N. Steinberg, C. L. Bowman, and F. E. White, "Revisions to the jdl data fusion model," in *AeroSense'99*, pp. 430–441, International Society for Optics and Photonics, 1999.

[158] A. N. Steinberg and C. L. Bowman, "Rethinking the jdl data fusion levels," *NSSDF JHAPL*, vol. 38, p. 39, 2004.

[159] E. Turban, J. Aronson, and T.-P. Liang, *Decision Support Systems and Intelligent Systems 7 Edition*. Pearson Prentice Hall, 2005.

[160] J. Roy, "Rule-based expert system for maritime anomaly detection," in *SPIE Defense, Security, and Sensing*, pp. 76662N–76662N, International Society for Optics and Photonics, 2010.

[161] S. Mascaro, A. E. Nicholso, and K. B. Korb, "Anomaly detection in vessel tracks using bayesian networks," *International Journal of Approximate Reasoning*, vol. 55, no. 1, pp. 84–98, 2014.

[162] R. O. Lane, D. A. Nevell, S. D. Hayward, and T. W. Beaney, "Maritime anomaly detection and threat assessment," in *Information Fusion (FUSION), 2010 13th Conference on*, pp. 1–8, IEEE, 2010.

[163] K. Kowalska and L. Peel, "Maritime anomaly detection using gaussian process active learning," in *Information Fusion (FUSION), 2012 15th International Conference on*, pp. 1164–1171, IEEE, 2012.

[164] A. Vandecasteele and A. Napoli, "Spatial ontologies for detecting abnormal maritime behaviour," in *2012 Oceans-Yeosu*, pp. 1–7, IEEE, 2012.

[165] J. Roy and M. Davenport, "Exploitation of maritime domain ontologies for anomaly detection and threat analysis," in *2010 International WaterSide Security Conference*, pp. 1–8, IEEE, 2010.

[166] F. T. Cetin, B. Yilmaz, Y. Kabak, J.-H. Lee, C. Erbas, E. Akagunduz, and S.-J. Lee, "Increasing maritime situational awareness with interoperating distributed information sources," tech. rep., DTIC Document, 2013.

[167] G. L. Rogova, P. D. Scott, C. Lollett, and R. Mudiyanur, "Reasoning about situations in the early post-disaster response environment," in *2006 9th International Conference on Information Fusion*, pp. 1–8, IEEE, 2006.

[168] P. Smets and R. Kennes, "The transferable belief model," *Artificial intelligence*, vol. 66, no. 2, pp. 191–234, 1994.

[169] G. Pilato, A. Augello, M. Missikoff, and F. Taglino, "Integration of ontologies and bayesian networks for maritime situation awareness," in *Semantic Computing (ICSC), 2012 IEEE Sixth International Conference on*, pp. 170–177, IEEE, 2012.

[170] R. N. Carvalho, R. Haberlin, P. C. G. Costa, K. B. Laskey, and K.-C. Chang, "Modeling a probabilistic ontology for maritime domain awareness," in *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on*, pp. 1–8, IEEE, 2011.

[171] R. Carvalho, K. Laskey, L. Santos, M. Ladeira, P. Costa, and S. Matsumoto, *UnBBayes: modeling uncertainty for plausible reasoning in the semantic web*. Citeseer, 2010.

[172] P. C. G. Da Costa, K. B. Laskey, and K. J. Laskey, "Pr-owl: A bayesian ontology language for the semantic web," in *Uncertainty Reasoning for the Semantic Web I*, pp. 88–107, Springer, 2008.

[173] K. B. Laskey, R. Haberlin, R. N. Carvalho, and P. C. G. da Costa, "Pr-owl 2 case study: A maritime domain probabilistic ontology.," in *STIDS*, pp. 76–83, 2011.

[174] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[175] ROS.org, "Ros introduction." http://wiki.ros.org/ROS/Introduction, Accessed on 15/01/2017.

[176] M. Cashmore, M. Fox, T. Larkworthy, D. Long, and D. Magazzeni, "Planning inspection tasks for auvs," in *2013 OCEANS-San Diego*, pp. 1–8, IEEE, 2013.

[177] L. Kunze, T. Roehm, and M. Beetz, "Towards semantic robot description languages," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5589–5595, IEEE, 2011.

[178] G. Papadimitriou and D. Lane, "Semantic Based Knowledge Representation and Adaptive Mission Planning for MCM Missions using AUVs," in *Proceedings of the OCEANS'14 MTS/IEEE Conference*, (Taipei), April 2014.

[179] G. Papadimitriou and D. Lane, "Semantic based knowledge representation and adaptive mission planning for mcm missions using auvs," in *OCEANS 2014-TAIPEI*, pp. 1–8, IEEE, 2014.

[180] S. Sariel, T. Balch, and J. R. Stack, "Distributed multi-auv coordination in naval mine countermeasure missions," Tech. Rep. GIT-GVU-06-04, Georgia Institute of Technology, GVU, 2006. http://www2.itu.edu.tr/šariel/publications.php.

[181] U. Navy, "The navy unmanned undersea vehicle (uuv) master plan," *US Navy, November*, vol. 9, 2004.

[182] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton university press, 2011.

[183] M. Prats, J. Pérez, J. J. Fernández, and P. J. Sanz, "An open source tool for simulation and supervision of underwater intervention missions," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2577–2582, IEEE, 2012.

[184] R. Wang and X. Qian, *OpenSceneGraph 3 Cookbook*. Packt Publishing Ltd, 2012.

[185] IRSLab, "Uwsim features." `http://www.irs.uji.es/uwsim/uwsim-features`, Accessed on 31/12/2016.

[186] E. Coumans, "Bullet physics engine." http://bulletphysics.org/wordpress, Accessed on 31/12/2016.

[187] R. Baxter, J. Cartwright, J. Clay, O. Clert, B. Davis, J. Lopez, F. Maurelli, Y. Petillot, P. Patrón, and N. Valeyrie, "Nessie v autonomous underwater vehicle," Accessed on 30/12/2016.

[188] "Open Source Computer Vision Library (OpenCV)." `http://opencv.org/`, Accessed on 1/1/2017.

[189] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[190] P. Singla and P. Domingos, "Discriminative training of markov logic networks," in *AAAI*, vol. 5, pp. 868–873, 2005.

[191] T. N. Huynh and R. J. Mooney, "Discriminative structure and parameter learning for markov logic networks," in *Proceedings of the 25th international conference on Machine learning*, pp. 416–423, ACM, 2008.

[192] T. N. Huynh and R. J. Mooney, "Max-margin weight learning for markov logic networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 564–579, Springer, 2009.

[193] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[194] J. Snyman, *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, vol. 97. Springer Science & Business Media, 2005.

[195] J. Besag, "Statistical analysis of non-lattice data," *The statistician*, pp. 179–195, 1975.

[196] "Alchemy - Open Source AI." `https://alchemy.cs.washington.edu`, Accessed on 3/1/2017.