A LEARNING ALGORITHM TO REPLAN AND ADAPT THE MISSION OF AN AUTONOMOUS UNDERWATER VEHICLE

Rodrigo Fonseca Carneiro

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Ramon Romankevicius Costa

Rio de Janeiro
Março de 2016

A LEARNING ALGORITHM TO REPLAN AND ADAPT THE MISSION OF AN AUTONOMOUS UNDERWATER VEHICLE

Rodrigo Fonseca Carneiro

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Ramon Romankevicius Costa, D.Sc.

Prof. Alessandro Jacoud Peixoto, D.Sc.

Prof. Antonio Candea Leite, D.Sc.

Prof. Tiago Roux de Oliveira, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2016

Carneiro, Rodrigo Fonseca

A learning algorithm to replan and adapt the mission of an autonomous underwater vehicle/Rodrigo Fonseca Carneiro. – Rio de Janeiro: UFRJ/COPPE, 2016.

XI, 50 p.: il.; 29, 7cm.

Orientador: Ramon Romankevicius Costa

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2016.

Bibliography: p. 48 – 50.

1. Underwater Autonomous Systems. 2. Artificial Intelligence. 3. Markov Decision Process. 4. Machine Learning. 5. Mission Planning. I. Costa, Ramon Romankevicius. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Ao meu filho Arthur com todo amor.

Agradecimentos

Primeiramente agradeço a minha família por todo apoio, amor e compreensão.

Primeiramente agradeço a minha família por todo apoio, amor e compreensão.

Um agradecimento especial a minha mãe Marly Saads por toda educação, dedicação e pelo fato de ter me ensinado a ser forte e lutar pelos meus objetivos.

Ao meu pai Lécio Carneiro, que mesmo longe oferece boas conversas e apoio para o meu desenvolvimento pessoal e profissional.

A minha esposa Monise Machado por todo carinho e compreensão em todos os momentos da minha vida e que nesse momento está trazendo para nossas vidas um grande tesouro, o Arthur.

Agradeço ao meu filhão Arthur que mesmo antes de nascer já me tornou um ser humano melhor e me motivou a sempre sonhar mais alto.

Aos meus avós por todo carinho depositado durante toda a minha jornada.

Gostaria de agradecer ao Professor, Orientador e amigo Ramon Romankevicius Costa por todo apoio, conversas, cobranças e acompanhamento deste aluno que não consegue externar suas ideias em uma folha de papel.

Aos professores Liu Hsu, Fernando Lizarralde, Eduardo Nunes e Alessandro Jacoud por seus ensinamentos e conversas técnicas que enriqueceram este trabalho.

Aos amigos que formei nesse período no laboratório, Marco Xaud, Alex Neves, Guilherme Carvalho, Renan Freitas, Henrique Duarte, Luciana Reys, Mariana Rufino, Alana Monteiro, Julia Campana, Gabriel Alcantara Eduardo Elael, Ighor Marcovistz, Bruno Campello e Fernando Coutinho que de certa forma sempre me incentivaram e tornaram a confecção deste trabalho mais agradável.

Aos amigos que formei na vida, principalmente nesse último ano, Camila, Leo, Leticia, Luciano, Elkya, Tha, Rafael, Bia, Daniel, Bebeto e Pablo, e claro aos amigos das antigas, Linconl, Daniel, Gustavo(s), Pedrão, Doug, Felipão e Ralph, que entre chopps, churrascos e conversas permitiram que eu tivesse momentos de diversão mesmo nas vezes em que levei o computador para desenvolver o trabalho durante nossos eventos. Obrigado por essa compreensão. Deu certo.

Agradeço ao povo brasileiro que indiretamente contribui para o estudo de qualidade que me foi oferecido e a COPPE pela oportunidade e confiança depositadas em mim.

Por último e não menos importante, agradeço a Deus pela vida e oportunidades que nunca deixaram aparecer. Espero estar retribuindo a altura, e deixando por aqui um mundo cada vez melhor.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

A LEARNING ALGORITHM TO REPLAN AND ADAPT THE MISSION OF AN AUTONOMOUS UNDERWATER VEHICLE

Rodrigo Fonseca Carneiro

Março/2016

Orientador: Ramon Romankevicius Costa

Programa: Engenharia Elétrica

Robótica autônoma é um dos principais desafios de pesquisa na área de robótica atualmente. Robôs submarinos são um caso especial por estarem imersos em um ambiente dinâmico com pouca comunicação com a superfície, sensores ruidosos e imprecisos e energia limitada.

Uma das principais tarefas para um robô submarino para ser autônomo é perceber o mundo, tomar decisões com base nessas percepções e navegar através do ambiente para cumprir sua missão.

Este trabalho irá introduzir os conceitos de percepção, tomada de decisão e Machine Learning a ser aplicado a um Veículo de Operação Remoto (ROV), a fim de transformá-lo em um Veículo de Operação Remoto Híbrido (H-ROV).

Esses conceitos serão aplicados em um replanejamento de missão onde o robô deve escolher a partir de um conjunto de diferentes ações aquelas que o farão atingir seus objetivo. Essas ações formarão um novo plano quando algo impede a execução do plano inicial. Para o replanejamento os conceitos de Reinforcement Learning serão aplicados, de modo a minimizar a energia necessária para a realização do novo plano de ações.

O algoritmo foi aplicado em um ambiente de simulação e mostrou-se eficaz para replanejamento e adaptação da missão.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

UM ALGORITMO DE APRENDIZADO PARA REPLANEJAMENTO E ADAPTAÇÃO PARA VEÍCULOS SUBMARINOS AUTONOMOS

Rodrigo Fonseca Carneiro

March/2016

Advisor: Ramon Romankevicius Costa

Department: Electrical Engineering

Autonomous robotics is one of the main challenges in present robotics research. Underwater robots are a special case as they are immersed in a dynamic and changing environment with poor communication with the surface, noisy and imprecise sensors and limited power.

One of the main tasks for an underwater robot to be autonomous is to perceive the world, to make decisions based on those perceptions and to navigate through the environment to accomplish its mission.

This work will introduce the concepts of perception, decision making and machine learning to be applied to a Remotely Operated Vehicle (ROV) in order to turn it into a Hybrid Remotely Operated Vehicle (H-ROV).

Those concepts will be applied on an autonomous mission replanning where the robot should choose from a set of different actions those which will make ir accomplishes its objectives. Those actions would make a new plan, when something prevents the execution of the initial plan. For replanning the concepts of Reinforcement Learning will be applied in order to minimize the energy in the formation of the new plan.

The algorithm was applied in a simulated environment and showed to be effective for mission replanning and adaptation.

Contents

List of Figures

Chapter 1

Introduction

Autonomous robots are one of the main challenges in present robotics research. Underwater robots are a special case as they are immersed in a dynamic and changing environment with poor communication with the surface, imprecise and noisy sensors and limited power.

One of the main tasks for an underwater robot to be autonomous is to perceive the world, to make decisions based on those perceptions and to navigate through the environment to accomplish its mission.

Autonomous decision making in robotics is the ultimate goal for autonomy.

1.1 Motivation

In 2004 Laboratório de Controle (LabCon) from Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia (COPPE) started the development of a Remotely Operated Vehicle (ROV) for hydroelectric intake tunnel inspection. This vehicle, Called LUMA (LUMA), was successfully tested in field and in 2007 it was modified for Antarctic Operations where it collaborated in three consecutive expeditions with the Census of Marine Life.

One of the objectives of using an underwater vehicle in Antarctic is to capture seafloor images and measure its geomorphology and this kind of mission is better accomplished by using an Autonomous Underwater Vehicle (AUV) instead of a ROV. Aligned with COPPE's idea about the strategic importance of the development of autonomous systems, Grupo de Sistemas de Controle em Automação e Robótica (GSCAR) started in 2010 a project funded by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) to turn the ROV LUMA in a Hybrid Remotely Operated Vehicle (H-ROV), which is a vehicle capable of working on a remote or autonomous mode.

The first effort to turn LUMA autonomous was developed by JUNIOR (2014). In this work it was developed a mission control system based on Petri Nets to inspect

underwater pipelines with the H-ROV LUMA. Petri Nets are very efficient to model *primitive actions* and to describe Discrete Event Systems (DES). PALOMERAS *et al.* (2012) combined together Petri Net Building Block (PNBB)s, that are small Petri Net (PN)s which properties are previously evaluated, to describe the behavior of the system and to guarantee that the mission plan would be followed.

The study on autonomous systems brought the necessity of the creation of Autonomous System Architecture and they are normally divided in three layers:

- Reactive Layer;
- Control and Execution Layer;
- Deliberative Layer.

JUNIOR (2014) work was focused on the Control and Execution Layer where the Petri Net are defined. It was adopted an *off-board* paradigm for the deliberative layer, this means that the mission must be pre-programmed for the vehicle to be able to accomplish it. So the mission success depends on the ability of the programmer to predict every fault that can happen and insert all the possibilities on the Deliberative's Layer Planner. This kind of approach works well on very structured environments.

To enhance the robots capabilities to work in unstructured environments it is needed to implement certain functionalities in the Deliberative Layer as:

- a good World Modeler;
- a good Planner;
- a plan adapter.

A good *World Modeler* will generate information about the environment and will update the World model based on the robots *perceptions* (better explained in Section 2.1) received from the Reactive Layer. It is very important to choose a good representation method as this model will be used during the whole mission and should be stored, accessed, and shared efficiently. MIGUELANEZ *et al.* (2011), for example, uses a Semantic Knowledge-based framework.

A good Planner will be able to get information from the World Modeler and from the Mission Definition given by the operator and transform it in a *Mission Plan* in the Execution Layer. In Component Oriented Layer-based Architecture for Autonomy (COLA2) architecture (PALOMERAS *et al.* (2012)), for example, the *Mission Plan* groups the PNBB to describe and execute the mission. This should be done by the operator in JUNIOR (2014) work.

The *Plan Adapter* is responsible to change the plan without the interference of the operator. It will use all the information given by a machine *perception* module and a *System Status* module. To develop this functionality it is necessary to introduce some intelligence on the robot so it can adapt and replace the primary plan. The decisions should be based on its Situation Awarness (SA_H) (Better explained in Chapter 2). This module is dependent on the robot perception of the world and its representation. PATRON (2010) used the SA_H to replan or repair an ongoing mission.SA_H is the capability of the vehicle to comprehend what is happening around it and within its components.

Another good example of Adaptive Mission Planning is the work of PAPADIMITRIOU e LANE (2014) where it was developed a Semantic Based Knowledge Representation and Adaptive Mission planning for Underwater Mine Countermeasures (MCM). In this work, they have used *ontological* representation of the world limited for the MCM mission capable of semantically express the knowledge of their components, their state and the state of the world in which they are deployed and to demonstrate how they use the semantic knowledge to plan the missions and adapt on the fly when the vehicle receives new information. They have executed the mission successfully in a simulation environment using a Planning Domain Definition Language (PDDL).

Another issue when talking about autonomous systems is how the robot can learn new things from the environment and update its *World Model*, optimizes its control algorithm or even its own model. One example is the work of EL-FAKDI (2010) where the robot learned how to track a pipeline by applying machine learning techniques.

One of the advantages of using machine learning algorithms, especially Reinforcement Learning (RL) in the robot control is that the robot or the environment model do not need to be known. Both can be acquired during the robot learning process and updated during its mission and every mission contributes for the learning process.

1.2 Objectives

The objective of this work is to modify LUMA's control architecture in order to turn it into and H-ROV.

This work proposes this new architecture based on the three-layered architectures found on literature.

It also implements a mission plan adaptation algorithm based on RL techniques.

1.3 Methodology

For the purpose of this work it will be considered that a Knowledge Representation framework is developed and the AUV always have a complete perception of the environment. This means that the robot always identifies targets, obstacles and its system status, more than that it always identifies if the primitive action chosen from JUNIOR (2014) was executed correctly.

The SA_H and the Semantic Knowledge-Based framework from MIGUELANEZ *et al.* (2011) will be introduced for better comprehension of a complete autonomous system and the parts that are desired for a system to become completely autonomous, but its implementation is left for a future work. The introduction of those frameworks will make the system capable of detecting the pipeline, inspect and adapt its mission plan in case something interfere with its initial plan (given by the operator).

The work from PATRON (2010) presents an effective adaptive mission planning where he uses high level primitive actions to accomplish its mission. The primitives presented in JUNIOR (2014) are much simpler than those found on PATRON (2010) work making the rearrangement of primitive actions easier as they can be considered policy actions instead of a policy plan. The extrapolation of the method can be considered easy if it is guaranteed the correct execution of the more complex primitive actions.

EL-FAKDI (2010) applied RL to learn how to track a pipeline, this knowledge can be used to implement in itself a pipeline tracking primitive creating an upgrade to the capabilities of the robot. Inspired by that, this work will apply RL methods so the robot can learn how to track fixed and moving targets by changing the mission plan created with JUNIOR (2014) primitives.

The plan will consist of simple actions like, `move forward`, `move backward`, `turn`, `move down` and `move up` that are easily mapped in the primitives proposed by JUNIOR, making this approach a simple way to introduce learning and replanning algorithms to an AUV in its initial development phase like the H-ROV LUMA.

1.4 Summary and Outlook

Chapter 2 will present a brief evolution on the autonomous systems control architecture, shows the present configuration of LUMA robot and proposes a new architecture that would include the results of this work and would make it possible for the robot to achieve some degree of autonomy.

Chapter 3 will introduce the concepts of machine learning, the algorithm used in this work and a brief description of the its implementation.

Chapter 4 will discuss the evolution on mission planning, make the proper definitions of mission and presents the missions LUMA should be able to accomplish.

Chapter 5 will present the simulation environment and show and discuss the simulation results.

Chapter 6 will conclude the work and suggests future works related to Autonomous Systems and with immediate use in LUMA.

Chapter 2

Control Architectures in Autonomous Systems

According to TURNER (2005) the primary goal in a Control Architecture is flexibility and adaptation as the autonomous robots are immersed in a dynamic and changing world.

The first suggested architecture was developed in 1960 and is called Classical Deliberative Architecture(FIKES e NILSSON (1971), NILSSON (1984), ALBUS (1991), LAIRD e ROSENBLOOM (1990)) and is represented by Figure 2.1. The main characteristic of this architecture is a top-down philosophy. In this architecture the sensors are read and flowed into a world model which is used by the Planner to execute the tasks to accomplish the mission without directly using the sensors data, this brings difficulties when environmental changes occur. This kind of approach proved to be very inefficient in non structured and highly unpredictable environments.

Figure 2.1: Classical Deliberative Architecture

To solve this problem (BROOKS (1986)) developed a behaviour-based control architecture which is built from layers of interacting finite-state machines (FSMs) - also called behaviours - that are executed in parallel bringing a fast and reactive characteristic. This makes the robot very efficient in respect to environmental changes, but it needs a very reliable arbitration mechanism as it needs to choose efficiently between a set of behaviours to be executed making long range missions very difficult to undertake.

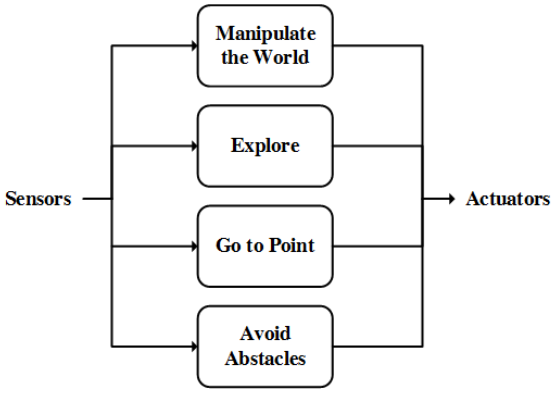

Figure 2.2: behaviour-based Architecture

A new architecture was then created joining together the benefits of both. Called Hybrid Architecture or Layered Architecture it combines planning capabilities of Classical Deliberative and reactivity of behaviour-Based architecture (FIRBY (1989) ARKIN e BALCH (1997) LYONS (1992) GAT (1991)). This approach is structured in three layers: Reactive layer; Control execution layer and Deliberative layer or Mission Layer. In this approach sensors and actuators are connected to the Reactive Layer which also receives information from control execution layer allowing fast response to environmental changes while executing the mission instructions generated by deliberative layer.

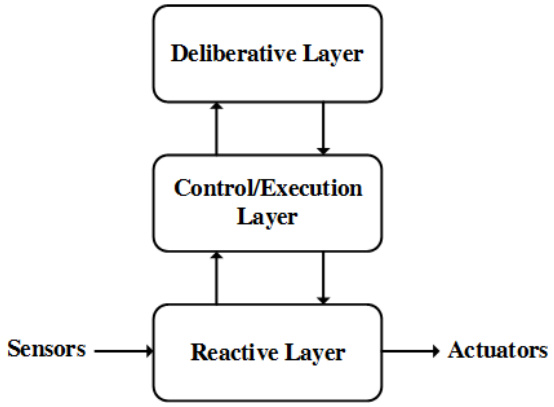

Figure 2.3: Hybrid Architecture

Deliberative/Mission Layer Transforms the mission into a set of tasks which defines a plan and determines the long-range tasks of the robot based on high-level goals.

Execution Layer Interacts between the upper and lower layers, supervises the accomplishment of the tasks, acts as an interface between the numerical reactive layer and the symbolical deliberative layer interpreting high-level plan primitives activation, monitors the primitives being executed and handles the events that these primitives may generate.

Reactive Layer takes care of the real-time issues related to the interactions with the environment. It is where the necessary sensors and/or actuators are directly connected and the data is collected to be passed to other layers.

One good example of Hybrid Architecture is the COLA2 presented by PALOMERAS *et al.* (2012) which diagram can be viewed in figure 2.4.

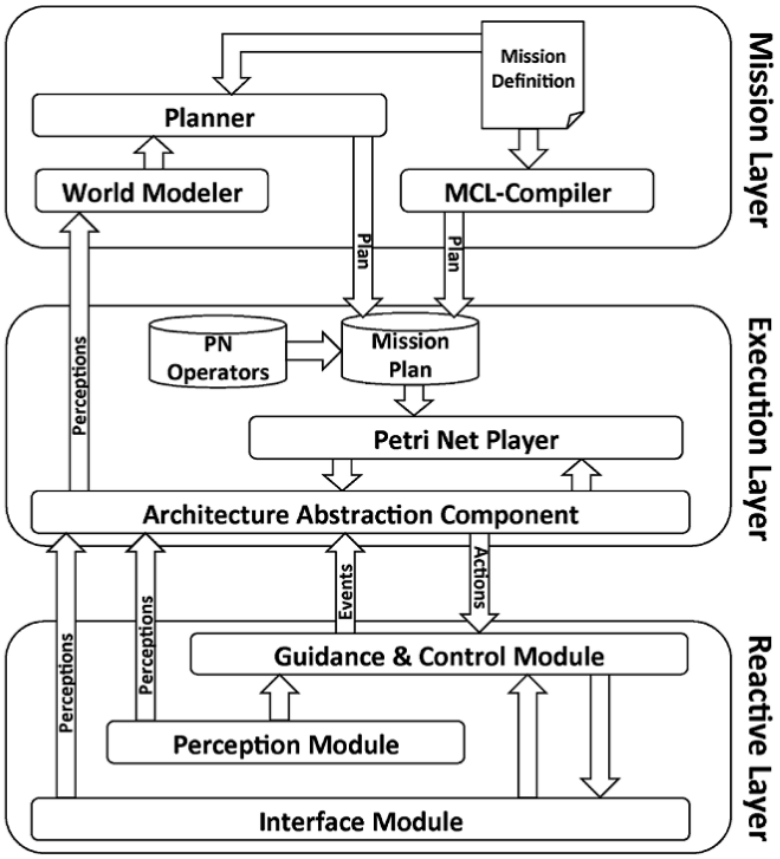

Figure 2.4: COLA2 Architecture[1]

Recent control architectures aims in a methodology to adapt to new incoming events that may occur during a planned mission (KORTENKAMP e SIMMONS (2008)). In 2012 LANE *et al.* (2012) started a project called PANDORA (LANE *et al.* (2012)) where they introduced the concept of persistent autonomy which means that the robots should be autonomous for a long time.

They identified three essentials areas to provide the foundations for Persistent Autonomy:

- *Describing the World*
- *Directing and Adapting Intentions*
- *Acting Roboustly*

[1]Extracted from PALOMERAS *et al.* (2012)

PANDORA architecture is divided in four levels: Execution, Operational, Tactical and Strategic and they are connected as shown in figure 2.5.

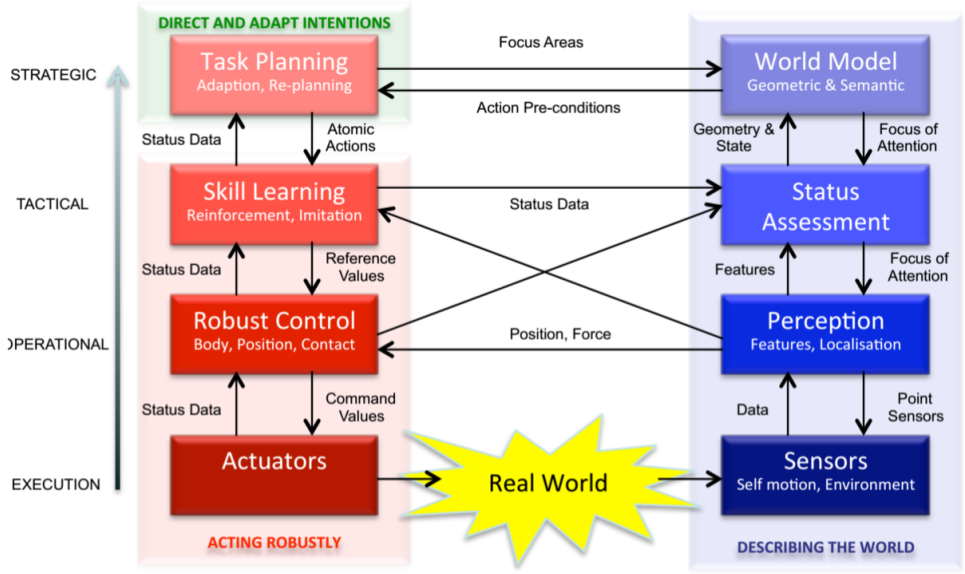

Figure 2.5: Pandora Architecture

In the Operational Level, the *Perception Module* process sensor data, removes noise, extract and track features and localize using SLAM. This module outputs are used on *Robust Control Module* that updates the controller reference

In Tactical Level, *Status Assessment Module* uses status information from around the robot in combination with expectations of planned actions, world model and observed features to determine whether actions were executed satisfactorily or have failed.

In Strategic level, sensor features and state information are matched with geometric data about the environment to update the world model. Planning uses both semantic and geometric information as pre-conditions on possible actions or action sequences that can be executed.

In figures 2.5 and 2.4 one can notice the Perception module. This module is the key for autonomous systems, note that its information is transmitted for the whole system. The more accurately information this module can generate the better for the system. This is used for localization and mapping, detecting targets or obstacles, executing an auto diagnostics to check the system status, calculate capabilities among others.

2.1 Perception

From the dictionary *Perception* is the ability to see, hear, or become aware of something through the senses. It is a way of regarding, understanding, or interpret-

ing something.

Perception covers all the way from detecting a certain configuration of the world around the observer to his way of thinking about it.

There are two contexts affecting the process of perceiving, the first covers the environment in which the perceived object rests within the observers world and the second concerns the way in which the world is understood inside the observer.

The object in the environment is perceived by the observer system because a certain interaction takes place between them. The most common interaction is called *stimulation* and it happens when the objects emits the signal, like the reflection of the incident light or a smell release, for example.

Perceiving an object depends on *environmental correlation* and *cognitive equivalence*. This relation is determined by the capacities, purposes and state of the observer system (PANIAGUA (2007)).

Studies and theories of perception are concentrated on cognitive equivalence and they are mainly divided in the following areas:

- Exploration and categorization of forms of cognitive equivalence in real systems;
- Studying the biological parts involved and their electrical, electro-chemical behaviour and the way they correlate with its cognitive equivalent;
- Investigating ways of designing cognitive equivalence relations for artificial systems and
- Extracting general principles and laws from the relation of cognitive equivalence with goal-oriented behaviour, survivability, efficiency, resources, etc.

Perceiving the world means that the robot has to get sensor data and make a **Cognitive Equivalence** from the collected data and the knowledge base available for him. For that reason, it is needed a representation of that knowledge in a language that a computer can understand. To share common understanding of the structure of information among people or software agents **Ontologies** are widely used (PAPADIMITRIOU e LANE (2014) and MIGUELANEZ *et al.* (2011)) and for that Stanford developed the Protege framework that is a free software with a Graphical User Interface (GUI) to define ontologies. Only with a good perception of the environment the autonomous decisions will be possible.

For the purpose of this work, it will be considered that the robot perceives perfectly the environment and its perception generates a trustful world model and an exactly localization, even though the model that will be presented will consider that a partial observability is possible, so the same model can be used in future works, thus the Perception Module is not implemented.

2.2 Situation Awareness and Decision Making

According to the U.S. Cost Guard SA_H is the ability to identify, process, and comprehend the critical elements of information about what is happening to the team with regards to the mission. More simply, its knowing what is going on around you. SA_H breaks down into three separate levels: perception of the environment, comprehension of the situation and projection of the future status (MIGUELANEZ *et al.* (2011)).

Improving an agent SA_H means improving the information in its database and

John Richard Boyd, a United States Air Force fighter pilot and Pentagon consultant of the late 20th century, described the OODA loop (Observe-Orient-Decide-Act) as the main command and control loop of a decision making process which is represented in figure 2.6 (BOYD (1992)).

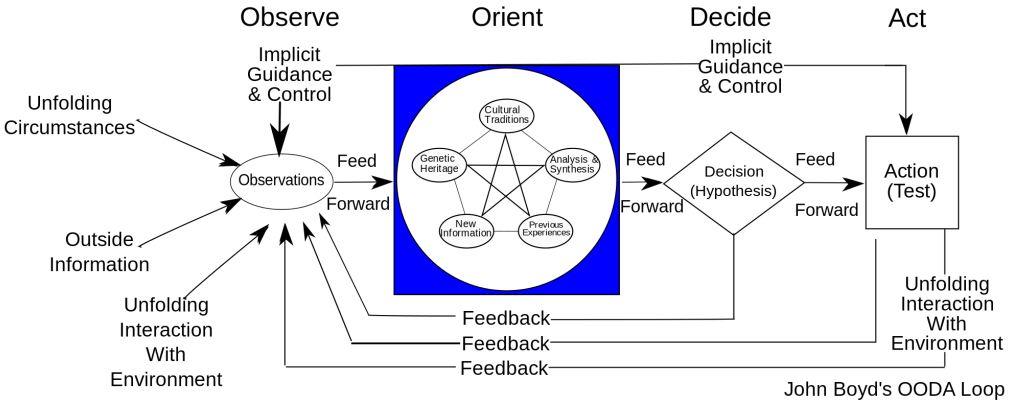

Figure 2.6: OODA Loop

According to ENDSLEY *et al.* (2003) and MIGUELANEZ *et al.* (2011) **Observation** is acquired through the available information and it can be said that this is the perception level of SA_H. **Orientation** is provided based on previous acquired knowledge and cultural factors of the decision maker and determines also which information should be observed and how it should be used. Decision stage represents the SA_H levels of comprehension and projection and determines the appropriate courses of action. Acting is made by the field units, in the robots case by the actuators and sensors, this stage not only actuate but it feed more information on the changing situation into the observation cycle.

Improving a robot SA_H means improving its capability of sensing the environment and improves its database and data access to correctly correlate the sensed data into useful information and improve its correlation speed. This is a research topic of MIGUELANEZ *et al.* (2011) and is a key business for robots to become persistently autonomous.

The SA_H improvement is achieved by constant updates to the database, the correct access and use of this database and the correct categorization of the new information received and perceived by the robot. One useful strategy for that is the use of knowledge frameworks and the use of Artificial Intelligence (AI) to collect and categorize the data received during the robot operation in a way it contributes to the enrichment of the knowledge (MIGUELANEZ *et al.* (2011)).

For the aim of this work SA_H level will not be considered as the knowledge frameworks are not implemented and tests with different levels of SA_H are not possible but it will be discussed where the SA_H will be included in the architecture.

2.3 H-ROV LUMA

The H-ROV LUMA (figure 2.7) is being upgraded in the last years so it could act autonomously, for that, new sensors are being integrated and some older equipments are receiving upgrades in order to improve its localization and SA_H. Also the video system gained a dedicated processor so it offers computer power to use video processing algorithms and the navigation system gained a dedicated processor so it is possible to implement more complex control, navigation and decision making algorithms.

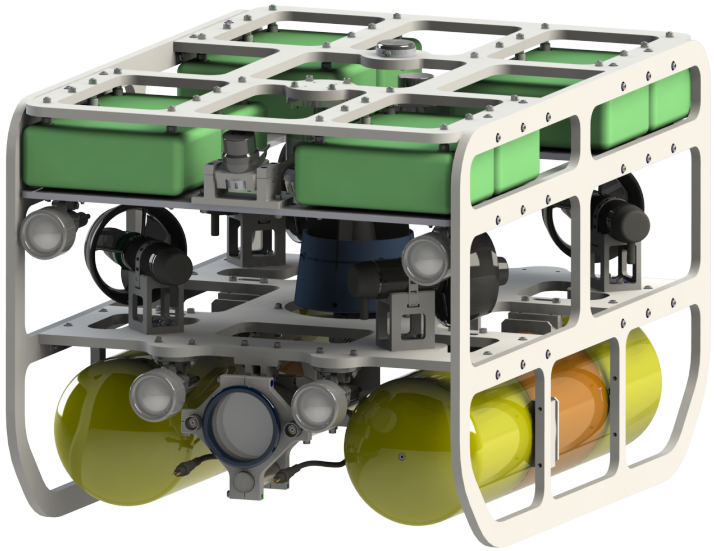

Figure 2.7: H-ROV LUMA

Nowadays LUMA counts with the following configuration:

- 5 1000 W electrical thruster
- 1 Pan&Tilt color camera

- 1 High Definition (HD) color camera with photo and zooming capabilities
- 1 Low light black&white camera
- 2 Standard definition (SD) color cameras
- 4 LED Lamps (Light Emitting Diode)
- 2 Laser pointers for size reference
- 1 Hydroacoustic modem
- 1 Pressure sensor
- 1 Altimeter
- 1 Inertial Measurement Unity (IMU)
- 1 Imaging sonar
- 2 PC-104 1.8 *Ghz* Dual core Atom processor
- 1500 m fiber optics communication link
- Expansion capabilities for more sensors or actuators

Moreover LUMA is now programmed using the Robot Operating System (ROS) framework, where all the control and observation algorithms are implemented in a way that new functionalities/capabilities would be easily integrated in the future. A good example would be the the work of DE OLIVEIRA LIMA (2015), where it was implemented a localization technique based on low cost sensors.

2.3.1 LUMA framework

LUMA software was developed using ROS framework. The ROS is a set of software libraries and tools that simplifies the task of creating complex and robust robot behaviour across a wide variety of robotic platforms (`ros.org` in 4th of March 2016).

The most important concept of this framework is that the robot software is divided into small units called **Nodes**, so instead of creating one chunk of code the developer creates a series of independent modules. These **Nodes** are processes that can communicate with each other even on different machines. This communication is transparent for the developer as the framework resolves that automatically.

ROS was built with code reutilization in mind. The codes are defined inside packages that are published in `ros.org` so everyone can use it. This functionality

makes ROS a powerful tool for development as there are bunches of packages available for lots of equipments, like sonars, motor drivers, depth sensors, among others. By that way you only need to download the package you need from the community and integrate it with the the robot's software.

Logging, parameters server for Node configuration, standard messages (data structure)) for devices or control and Unified Robot Description Format (URDF) are very useful features for software development and robot operation. Nodes communicate to other nodes by the standard messages so it is not needed to know what the other node does, only what messages it needs to work. Basically two types of communication happens in ROS Client-Server and Publisher-Subscriber and there are two basic types of messages Topics or Services, everything represented by Figure 2.8

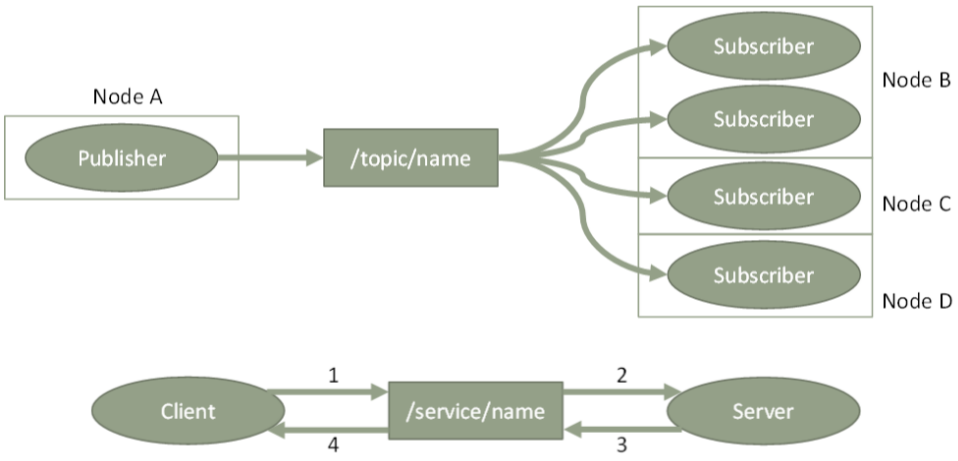

Figure 2.8: ROS Message Passing

LabCon/GSCAR software team designed the Robot GUI a GUI to be used on all laboratories projects that would be useful to save man-hour in software development in future projects. This software is being tested in two robots at that moment: LUMA and DORIS (DORIS).

The Robot GUI is a ROS Node and its windows behaviour is similar to ROS GUI. It has three main features:

- Component;
- Tool;
- Robot Package.

The Component has the same purpose of a Node, it is created inside the ROS GUI and can exchange ROS messages with topics or services. It is represented by one or more classes in C++ language. While a Tool is a Graphical windows in the

Robot GUI, the Tool is the way users can interact with the robots and it is also represented by one or more classes in C++. Figure 2.9 shows how these features are connected in the Robot GUI.

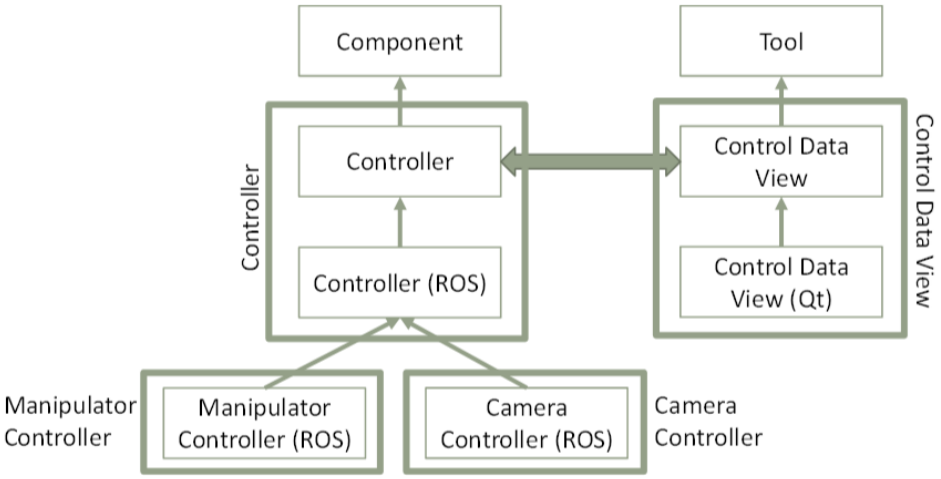

Figure 2.9: Component and Tools connections in Robot GUI

A Robot Package is a library containing Tools and Components for a certain Robot and can depend on other Robot Packages, also Components and Tools can be derived from Components and Tools from another ROS Package which allows Tools to be connected to Components defined on other Robot Packages.

Robot GUI can load Robot Packages and after loading, the User can create as many Tools as he desires and can customize the windows with them dynamically. the process of loading Robot Packages is what allows the same GUI to be used in different robots with the same Tools.

An Instance of a Robot Package has components. The components to be created are described in a Extensible Markup Language (XML) Components and tools to be created are described in XML files and when they are created Robot GUI searches for all suitable connections. Figure 2.10 shows some of the components on LUMA Package and how they are connected and figure 2.11 shows the connections between some of its components and tools.

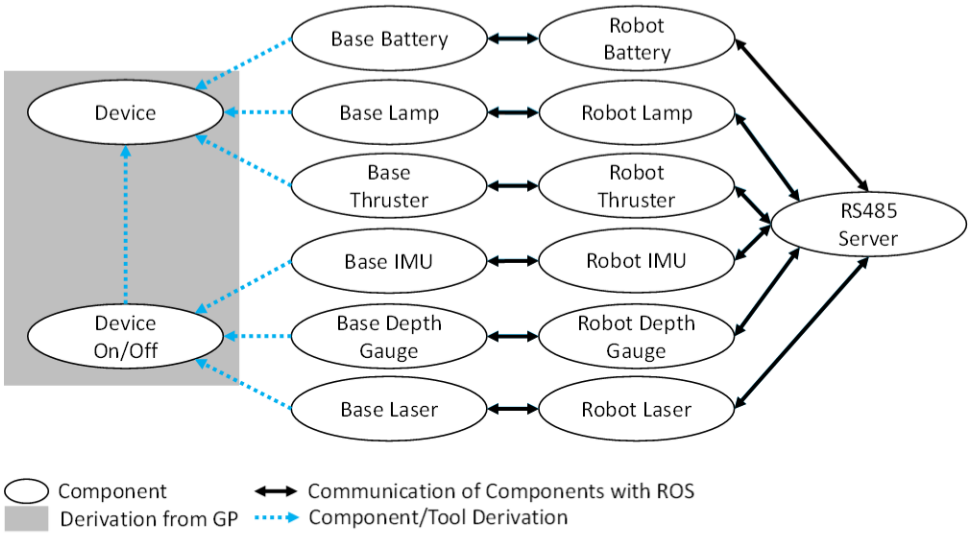

Figure 2.10: LUMA Package

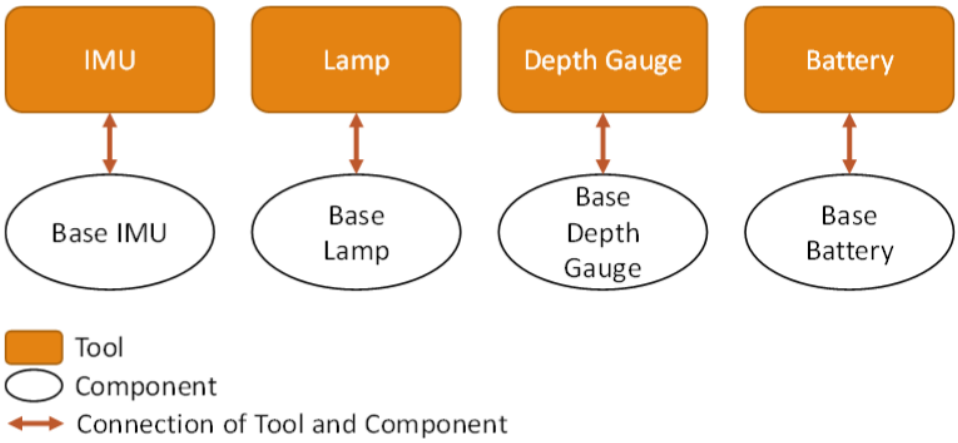

Figure 2.11: Component and Tools connections in LUMA Package

This software architecture allows the system to be modular, so any system upgrade is easily done by creating a new component and each module of LUMA control architecture should be developed as one or more components in the Robot GUI. The learning algorithms will also be implemented as a Component connected to the Robot GUI. Other components will be necessary to implement the **World Model**, the **Planner** and the **Knowledge Framework**.

A good ROS characteristic is that some simulation environments integrates with the framework making it natural to move from the simulation environment to the real world. This characteristic will prove to be very efficient when talking about machine learning as the agent can learn from a simulated environment before acting in the real world. Although it is not the same thing because of the uncertainties found on the real environment, letting the robot acting in the simulated environment for its first steps is a good policy.

Tecgraf and GSCAR developed a simulation environment for the offshore market where robot models are operated by its own software integrated through the ROS framework. One example can be found in SANTOS *et al.* (2013) where MOTOMAN, an anthropomorphic robot with two arms from YASKAWA, is controlled in a virtual offshore facility performing some tasks like valve control.

Using ROS it is possible to integrate MATLAB, SIMU-EP that already has a virtual underwater environment and the LUMA Robot GUI so that it is possible to check LUMA's control algorithms performance before deploying the robot in the real world. The same can happen with the learning algorithms, as it will be explained in chapter 3 it is needed some trial and error essays if we want LUMA to act autonomously and so, having a virtual environment for that more than accelerating the learning process, it makes it safer for the equipment and through the ROS integration the knowledge acquired in the virtual world can be applied in the real world.

2.3.2 LUMA control architecture

Figure 2.12 shows the present LUMA control architecture. The **Mission Plan** is defined *offline* by the system operator using the primitives available in LUMA's database. This plan is then transmitted to the **Mission Player** which will start to execute the tasks when a *start mission* signal is generated by the operator. From this point on, LUMA operation turns to autonomous mode and the **Mission Player** constantly sends reference signals to the **Controller Module** which will execute the low level controllers, using information received from **Signal Interface** in order to control Depth, Yaw and $X - Y$ plane position. Signals generate by the **Controller Module** will be sent to the **Signal Interface** that is also responsible to generate the correct signals to drive the actuators.

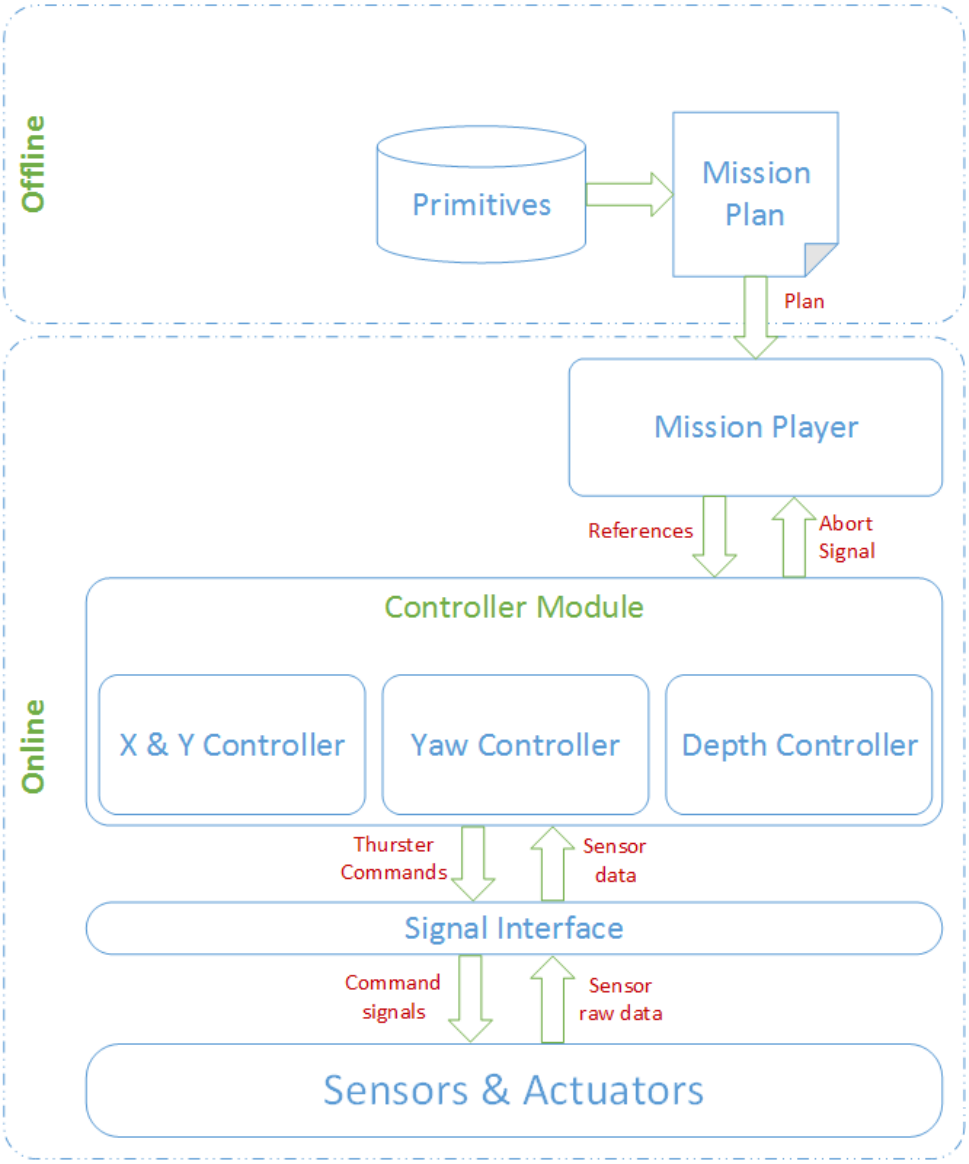

Figure 2.12: LUMA control architecture

This is the first autonomous control architecture developed to LUMA and although it is not implemented it has been successfully simulated in JUNIOR (2014). This work was developed before the changes on LUMA framework and no worries were considered by that time on the lack of compatibility of nowadays Petri Net players and the ROS-framework. Section 2.3.3 will show a solution for this problem.

2.3.3 New LUMA Architecture

With the development of this work, several changes should take place in LUMA's architecture so it would be possible to include the new modules, the changes can be seen in Figure 2.13.

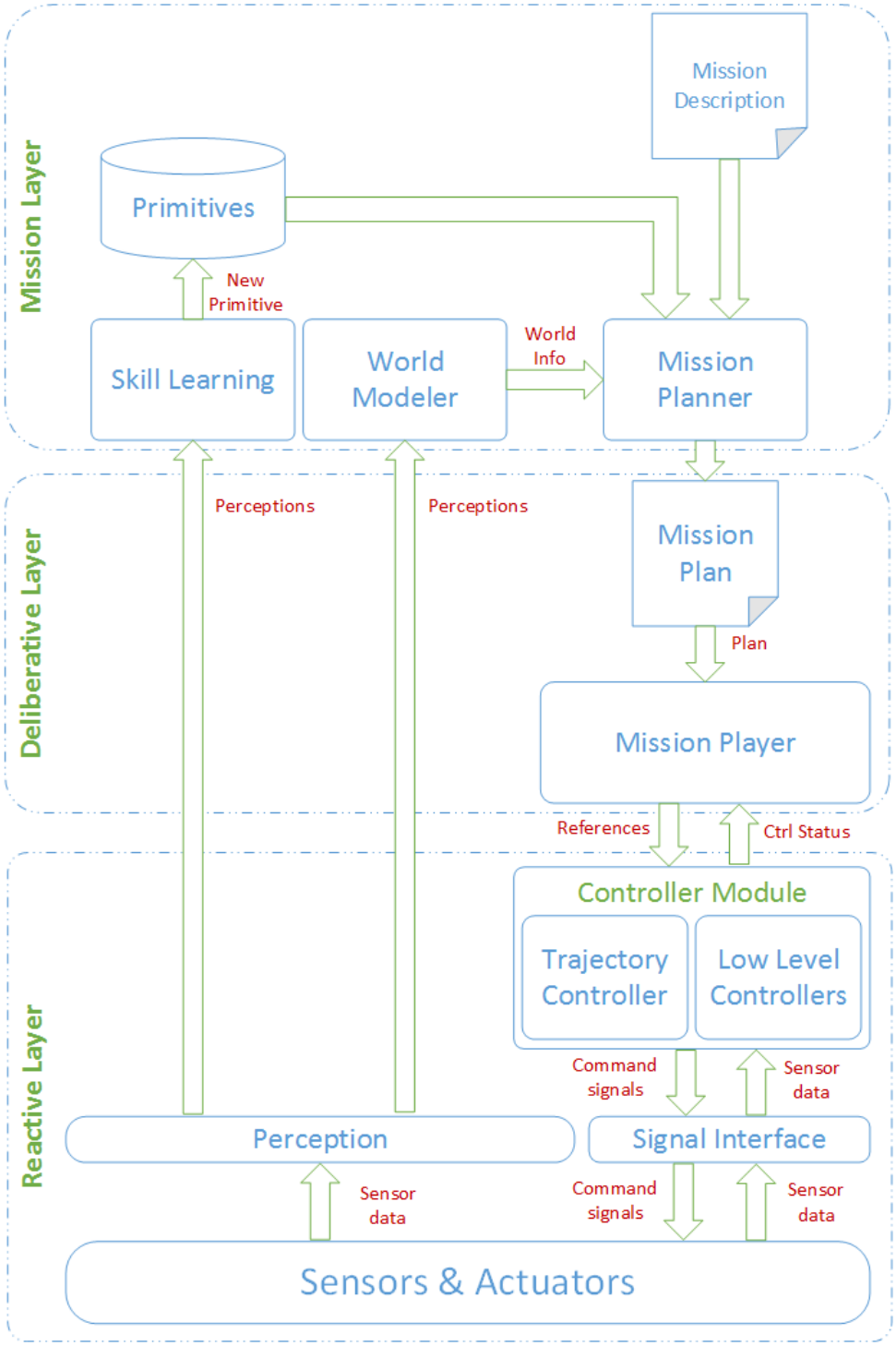

Figure 2.13: New LUMA control architecture

The first change should be the exchange of the PN developed by JUNIOR (2014), considering that there is not a stable connection between PN players and the ROS frameworks, with the state machines programmed in the Phyton Library SMACH proposed in DE FREITAS (2016) which is perfectly compatible with ROS. Now the **mission plan** is composed by a set of state machines which will be executed by the **Mission Player**.

A **Perception** module has been added so all sensor data, system status and controller status information are collected by it turning the system able to get a precise perception of the environment and generates the vehicle SA_H. These perceptions are transmitted back to the **Controller Module** in order to improve the control algorithms, and also to the **Skill Learning** module and **World Modeler** so the world model would be updated and new skill could be learned in each mission.

The **Skill Learning** module is responsible for the creation of new system **primitives** which would turn the robot capable of more complex missions. The **primitives** and the **Mission Description** would be used together by the **Mission Planner** to generate the **Mission Plan** based on the *world info* given by the **World Modeler**.

Besides the inclusion of new equipments, the **Controller Module**, **Signal Interface** and **Sensors & Actuators** remain the same, except that now more complex algorithms could be executed by the **Controller Module**.

Chapter 3

Machine Learning

Machine learning is present almost everywhere, like helping Netflix to suggest movies for us to watch or amazon to suggest new books or products or in smartphones in the technologies like google now or SIRI. Besides that it is known that European soccer teams invests thousands of dollars in machine learning to process all the kinds of data acquired during a game to improve the team's strategy. More than that there is a push to use machine learning to help in the cure of cancer and AIDS and possibly solve every problem humanity has. (DOMINGOS (2015))

Knowledge used to come from three sources: Evolution, Experience and Culture. Evolution is the knowledge encoded on your DNA, Experience is the knowledge in the neurons and Culture is the knowledge acquired by studying. According to DOMINGOS (2015), Professor of Computer Science and Engineering at the University of Washington, a forth source appeared recently: the computers. Yann LeCun, Director of AI Research once said "Most of the knowledge in the world in the future is going to be extracted by machines and will reside in machines".

The question here is how computer discovers new knowledge? For that there are mainly 5 ways to do that and each one of them is connected to school thoughts on Machine Learning. The 5 most expressive schools are summarized on table 3.1.

School	Origins	Algorithm
Symbolists	Logic, Philosophy	Inverse Deduction
Connectionists	Neuroscience	Backpropagation
Evolutionaries	Evolutionary Biology	Genetic programming
Baysians	Statistics	Probabilistic Inference
Analogizers	Psycology	Kernel Machines

Table 3.1: Schools of Machine Learning

Symbolists believe that the the discovery of new knowledge is to fill in the gaps of the knowledge one already have. This process is normally done by inverse Induction.

Connectionists believes that the process of learning should be like in human beings. The algorithm should be programmed to act like the brain works. This school is the responsible for the Neural Networks also know nowadays as Deep Learning. It was built a simple mathematical model of single neuron containing all that is needed to provide learning capabilities and they put lots of them together building a network. The neuron model is a sum of weighted inputs with an activation function such that if the sum is higher than a threshold its output is activated (See figure 3.1).

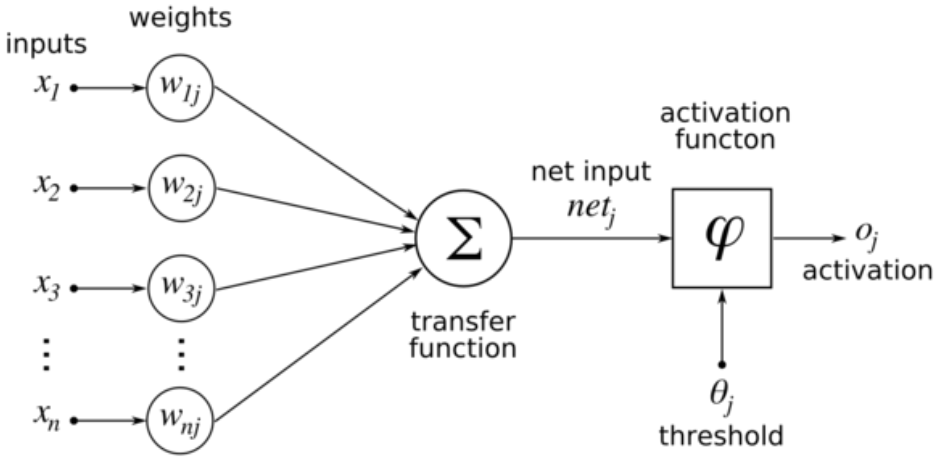

Figure 3.1: Neuron Model[1]

The challenge now is to make this network behaves like a reduced capability brain. This is done by training the network that is adjusting the weights and thresholds to make the neurons activate accordingly. The most common process of doing this is called back propagation, where you put an input on the network and check if the output is the expected one and if it is not all the neurons of the previous layer are checked and has its thresholds and weighted inputs adjusted so they shows the expected output. This is done until the first layer is reached and adjusted.

This technology is vastly used in almost everything nowadays like in predicting the stock market, video recognition, simultaneous translation, among others.

Evolutionaries believes that the knowledge comes from the evolution, in how the brain is built, so they believe that we should study how evolution works in order to learn really powerful things. The first work on that area started by the middle of the 50th century and started with Genetic Algorithm (GA). By the 80th century it was developed more powerful version called genetic programming.

In GA a population where each individual is represented by a genome interacts with the environment to perform a given task or tasks. The ones that achieve better results performing the given tasks are gains a bigger fitness value and for

[1]Extracted from `https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Print_Version`

that have a higher chance to be the parents of the next generation of individuals. GA operators are applied in two individuals generating a new one that has part of the genome of each one of the two individuals involved (father and mother), after that a random mutation can be applied on the result representing what actually happens in evolution. This process will generate the new population. (Figure 3.2)

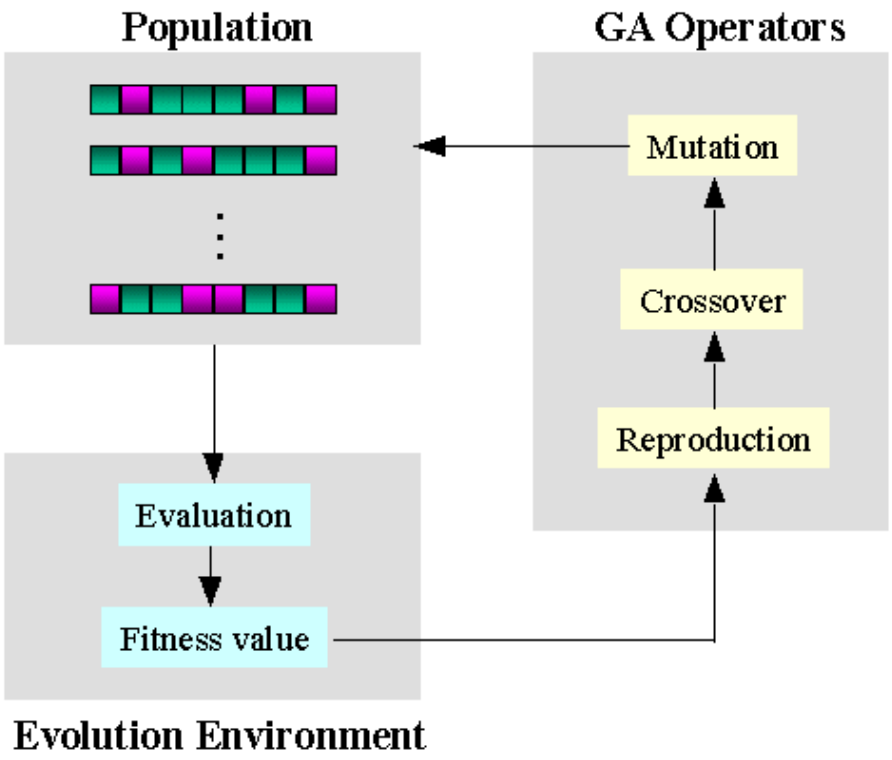

Figure 3.2: Genetic Algorithm Fluxogram Representation[2]

Hod Lipson from Creative Machine Lab in Columbia University is working in applying this algorithms to evolve real robots where they start as a pile of parts and as they interact with the environment they learn how they can improve themselves and generates a model to program a 3D printer to print their next generation.

Baysians believes that everything you learn is uncertain, so they compute the probabilities of the hypothesis and updates the probabilities as new data is given. Bayesian reasoning provides a formal and consistent way to reasoning in the presence of uncertainty; probability theory is an embodiment of common sense reasoning.Their learning algorithm is based on the Bayes Theorem (equation 3.1) which describes the probability of an event based on conditions that might be related to that event.

[2]Extracted from "An Educational Genetic Algorithms Learning Tool", Ying-Hong Liao, and Chuen-Tsai Sun, Member, IEEE - Weblink: `http://www.ewh.ieee.org/soc/es/May2001/14/Begin.htm`

$$P(H|e) = \frac{P(e|H)P(H)}{P(e)} \tag{3.1}$$

where:

$P(H)$ is the probability of H to happens

$P(e)$ is the probability of e to happens

$P(H|e)$ is the probability of observing event H given that e is true

$P(e|H)$ is the probability of observing event e given that H is true

From the point of view of machine learning, $P(H)$ is the **Prior** probability of each hypothesis H considered before any data or evidence e is observed. Then, as the data is observed, the probability of the hypothesis goes up if it is consistent with the data and goes down if it is not. The **likelihood** function $P(e|H)$ measures this consistence between the hypothesis and the data, this function measures how probable the data is observed if the hypothesis is true. The **marginal** function $P(e)$ measures how probable is the new evidence under all hypotheses and it is defined as $P(e) = \sum P(e|H_i)P(H_i)$. The function $P(H|e)$, called the **posterior** measures the probability of the hypothesis given the observed data and it shows how much you believe in the hypothesis after you see the evidence.

Some self driving car and SPAM filters are examples where Baysian learning is used.

Analogizers says that everything we learn is reasoning by analogies, that is making connections between the new situations and the situations that we are familiar with. This is widely used in recommendations systems like Netflix, Amazon or any other e-commerce site.

All the algorithms presented can be divided in three parts:

- Representation;
- Evaluation;
- Optimization.

Verifying that, DOMINGOS (2015) is working to unify them all in a universal learner capable of deriving all knowledgepast, present and futurefrom data. His work extracts the better characteristics of each school and apply all together in a new Algorithm that is being called the Master Algorithm.

The way he is doing that is through the application of the best algorithm that better suites in each of the parts. **Representation** contains the information on

how the *learner* represents what it is learning. In this part Domingos combined the logic characteristics of the symbolists, that are variations of First-order logic and the probabilistic model of the baysians represented by graphycal models (Baysen networks, Markov models among others). Both representations are extremely general and by combining them it is possible to represent almost everything. This generates a Probabilistic Logic model and the most widely used representation for that is called Markov Logic Networks (MLN).

The second part **Evaluation** tells how good a candidate model is, for that *posterior probability* is used as evaluation function. The third part, **Optimization**, is the algorithm that searches for the highest-scoring model and returns it. Genetic search coupled with gradient descent were chosen for that task.

Independent of the school, there are mainly three forms of learning:

- Supervised Learning;
- Unsupervised Learning and
- RL.

Supervised Learning basically is a function approximation
$y = f(x)$
where given a bunch of pairs $< x, y >$ the goal is to find the function f that maps a new x to a proper y. This is called Classification. Supervised Learning is used in prediction of future cases, knowledge extraction and compression, where the function is simpler than the data it represents. A good example of Supervised Learning is Anti Spam where the computer has to classify an e-mail as it is or it is not a Spam and the user sometimes shows tha a wrong classification was done.

In Unsupervised Learning you only have bunches of xs and has to find the function f that gives a compact description of the set of xs given. This is called *clustering* and is used to identify in which cluster a new x belongs to. This is like "learning what normally happens" and it produces no output function like in supervised learning. A good example is Bioinformatics where the computer uses the input data to give the most possibly diagnoses.

In RL, given a stream of $< x, z >$s the goal is to learn some f to generate y in
$y = f(x)$.

RL is used on game plays where a sequence of moves is needed to win, on Robots in a maze where a sequence of actions is necessary to achieve the goal, in multiple agents systems, among others. In this examples, the computer has to learn a policy, that is a sequence of outputs to produce the desired behaviour.

The main difference between RL and supervised learning is that the agent must interact with the environment to actually learn something, so no input/output pairs $< x, y >$ are given, they are produced dynamically on this robot interaction.

This work will use the concepts of RL and for that the variables $< x, z, y >$ and function f and its uses will be explained further on.

3.1 Reinforcement Learning

Although RL is widely used in statistics, psychology, neuroscience and computer science, only on the last twenty years it attracted the interests in the machine learning and artificial intelligence communities. It works on ways of programming agents by reward and punishment without needing to specify how the *task* is to be achieved (KAELBLING *et al.* (1996)).

Figure 3.3 shows a RL model. Basically an agent, in this case a robot, is connected to the environment via its perceptions and actions.

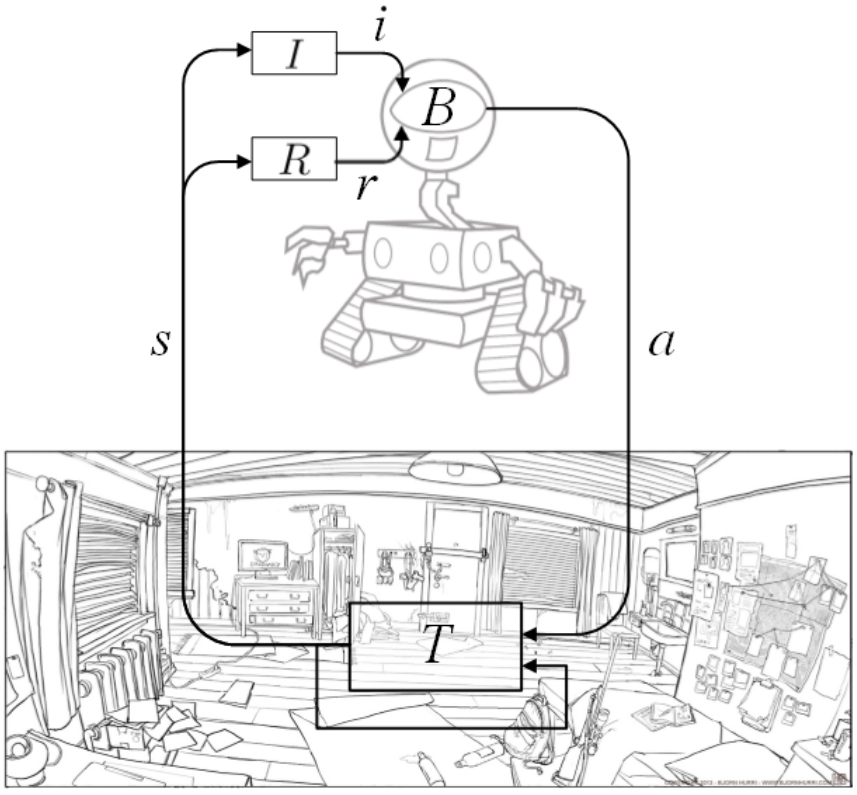

Figure 3.3: Reinforcement learning model[3]

On each step of interaction, the agent receives the input i that contains some indication of the current state s of the environment and based on that it chooses an action a as output. The action a changes the state of the environment and the

[3]Adapted from KAELBLING *et al.* (1996)

value of this transition is passed to the agent as a *reinforcement signal* r, also called reward. The agent's behaviour B should choose actions that tends to increase the sum of the r signals received. The problem on RL is that the agent only learn how to behaves in an environment overtime by systematic trial and error, guided by a wide variety of algorithms (KAELBLING *et al.* (1996)).

The RL model consists of

- a discrete set of environment states, $\mathcal{S}$;
- a discrete set of agent actions, $\mathcal{A}$;
- a set of scalar reinforcement signals, $\mathcal{R}$;
- an input function, I, wich determines how the agent perceives the environment state; and
- a function T that represents the model of the environment.

Although the environment is non-deterministic, which means that taking the same action in the same state on two different occasions may result in different state and reinforcement values, we are going to assume that the environment is deterministic, meaning that the *probabilities* of making state transitions or receiving reinforcement signals do not change over time. Even if this assumption is not very realistic, many RL algorithms showed to be effective in slowly-varying environments, unfortunately there are not many theoretical analysis for that.

Markov Decision Process (MDP) is widely used in many autonomous systems in different modules inside the architecture. PALOMERAS *et al.* (2012) and EL-FAKDI (2010) uses MDP in the reactive layer to learn and optmize the robot primitives. EL-FAKDI (2010), for example applyied Gradient-based RL techniques to learn a pipe tracking primitive. Once made, it can be easily used to learn other primitives.

URE *et al.* (2013) uses RL for solving the path planning problem on mobile robots acting in unknown dynamic environments.

PATRON (2010) on the other hand, uses MDP in the mission layer to implement an approach to mission planning. In this approach they map states to plan candidates which means that the selection of plans are done by calculating their estimated cost of execution and the reward obtained by reaching the new configuration of the mission environment.

3.1.1 Optimal Behavior

The objective of applying learning techniques on agents is for them to try to optimize their behavior. When talking about optimization it is need to specify what

should be optimized and the model of optimality that will be used. In the case of robots interacting with the environment it is needed to consider how it should take the future reward in account in the decisions it is taking now.

According to KAELBLING *et al.* (1996) there are various models of optimality but three of them are used in the majority of work in this area.

The easiest one to think about is the *finite-horizon* where the agent should optimize the expected reward for the next h steps:

$$E\Big(\sum_{t=0}^{h} r_t\Big) \tag{3.2}$$

Where r_t is the scalar reward received in step t. In this case it is not needed to worry what would happen after the last step. There are two ways to use this model. In the first, the agent with a non-stationary policy[4]would take the *h-step optimal action*, that is the action considered the best action given that there is h steps remaining, and then takes the *(h-1)-step optimal action* until the last *1-step optimal action*. In the second, the agent does what is called *receding-horizon control*, in which it always takes the *h-step optimal action* and the value of h limits how far the agent will take in account future steps. In this case after each step it includes in the reward expected value the action in step $h + 1$.

In the *infinite-horizon* model, the long-run reward is take into account but the more in the future the reward is, the more geometrically discounted it is:

$$E\Big(\sum_{t=0}^{\infty} \gamma^t r_t\Big) \tag{3.3}$$

where $\gamma \in [0, 1)$

The discounted value γ is necessary because the series may not sum up to $\pm\infty$ as it would make every policy optimal and no decision process would be taken, but it can be interpreted as an interest rate or the probability that the agent would be alive for one more step.

The third model is the *average-reward model* where the agent should optimize its long-rum average reward:

$$\lim_{h\to\infty} E\Big(\tfrac{1}{h}\sum_{t=0}^{h} r_t\Big) \tag{3.4}$$

This policy is also called as *gain optimal* policy, it is the limiting case of the infinite-horizon discounted model as the discounted factor approaches to 1 (BERTSEKAS (1995)). In this criterion it is not possible to distinguish between the policy that gains lots of rewards in the beginning and one that does not because the per-

[4]A policy that changes over time

formance received at the initial states are overshadowed by the long-run average performance.

When talking about choosing a policy, the model of optimality is decisive, depending on the model the choices in each state changes. For example, the *finite-horizon* is appropriate when the agents lifetime is known and an important fact about this model is that when its lifetime decreases it may change its policy.

For example, let's consider the grid world show in Figure 3.4. The final state represented by the green circle would give a total reward of $+10$, each other state except from the one marked by the red X would give a reward of -1 while the red X gives a reward of -7. The agent is represented by the the blue triangle.

Note that if the agents life is more than 6 steps, the optimal path would be the set $(LEFT, LEFT, UP, UP, RIGHT, RIGHT, RIGTH)$ resulting in a reward of $+4$, but if the agents life is fewer or equal to 6 steps the optimal policy would be the set $(UP, UP, RIGHT)$ resulting in a reward of $+2$. If the agent tries to follow the first policy with a limited number of steps left it would end in a state in between resulting in a reward of $-x$ where x is the agents life.

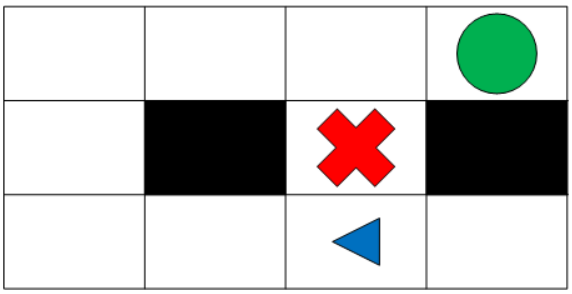

Figure 3.4: Grid World example showing the changes on reward function based on the agent's life

3.1.2 Exploration versus Exploitation

Exploration and Exploitation are what makes RL different from other kinds of learning. The agent should choose its action based on those concepts to try to reach the optimality. **Exploration** is a long-term process, with a risky and uncertain outcome while **Exploitation** consider the short-term process, where it tries to maximize the immediate benefit. For example, figure 3.5 shows a state machine where the first state has two possible transitions to be taken. If the agent chooses the $+5$ path it would receive $+5$ in the first action $+5$ in the second action and $+1$ for every action after the second one while if he chooses the $+1$ path it would receive $+1$ for its first action and then a $+10$ for every action taken after the 7th action. Note that if it is said that the agent will only live from 1 to 6 turns ($\in [0, 6]$) then the optimal path would be the $+5$ but if he would live more than 7 turns the

optimal path would be the +1. In the case of 7 turns both paths are equal, so the agent can choose freely.

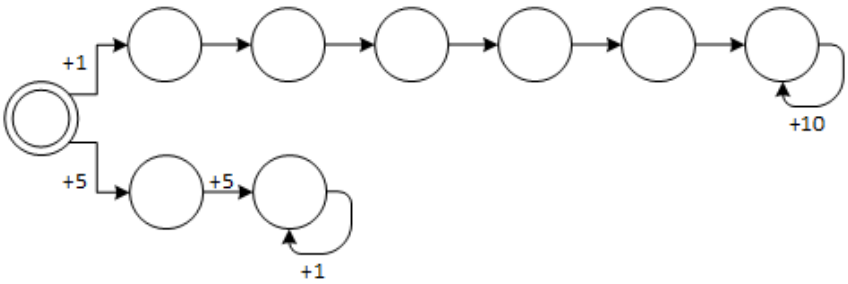

Figure 3.5: Exploration X Exploitation Example

It is easy to note that if the agent is an exploitation agent it would choose for the +5 path but if it is an exploration agent it would choose for the +1 path. This is easy when all the states and transitions are known, meaning that the model is completely defined, unfortunately in the case of autonomous agents this is not true, the agent should act and learn with the environment and in the environment so its first action would be chosen without knowing the true reward, what it normally computes is the expectation of the rewards and takes the actions based on that.

According to KAELBLING *et al.* (1996) there are fairly well-developed formal theory of explorations for very simple problems but those methods do not scale well to more complex problems like:

- Dynamic-Programming Approach
- Gittins Allocation Indices
- Learning Automata

These methods will not be discussed as their discussion are not part of the scope of this work but are mentioned here for a better overview of the RL problem.

3.1.3 Delayed Reward and the Markov Decision Processes

Although the methods present in section 3.1.2 were designed for the one state agent, they can be replicated for multiple state systems by applying them on each state individually. Most of these techniques converges to a regime where exploratory actions are rarely taken which is not a good strategy when talking about non-stationary environment. In a dynamic world, exploration must take place so the agent can notice its changes and learn with them.

In the general case, the agent's actions determine the immediate reward and the next (probable) state of the environment, remember that the real world is non

stationary and so the same action taken in the same state can lead to different states. The model of optimality described in section 3.1.1 will determine how the future would be taken into account.

In this case, the agent takes a long sequence of actions receiving insignificant reinforcement until it reaches a state with high reinforcement. During the process the agent must be able to learn which actions are desirable based on reward received arbitrarily in the future. This process is called **delayed reinforcement** or **delayed reward**.

The delayed reinforcement problems are well modeled as MDP that consists of:

- a set of states, $\mathcal{S}$
- a set of actions, $\mathcal{A}$
- a reward function: $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ that specifies expected instantaneous reward as function of current state s and action a. The notation is R(s,a).
- a state transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$, where a member of $\Pi(\mathcal{S})$ is a probability distribution over the set $\mathcal{S}$. The notation is $T(s, a, s')$ that is the probability of making a transition from state s to state s' using action a

The model is considered *Markov* if the state transitions are independent of any previous environment states or agent actions. That means the past does not matter for the future state transitions.

Finding a policy

The first step before letting an agent tries to learn how to behave in MDP environments is to explore techniques for determining the optimal police given the correct model as these techniques are the foundation for the learning algorithms.

The algorithms presented here are for the infinite-horizon discounted model, there are analogs algorithms for the other models of optimality but are not going to be discussed here as they are not going to be used in this work.

Let's start with the definition of the optimal *value* of state (equation 3.5) that is the expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy.

$$V^*(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \tag{3.5}$$

where π is the complete decision policy.

Doing some math and substitutions we find that

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s \in \mathcal{S}} T(s,a,s') V^*(s') \right), \forall s \in \mathcal{S} \tag{3.6}$$

Given equation 3.6 it is possible to specify the optimal policy as

$$\pi^*(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s \in \mathcal{S}} T(s,a,s') V^*(s') \right), \forall s \in \mathcal{S} \tag{3.7}$$

With the equations defined, one possible way to compute the optimal policy is to find the optimal value function. The simplest way is using an iterative algorithm called *value iteration* (presented below) that BELLMAN already proved its convergence to the correct V^* values.

Function *Value_Iteration* **is**
  initialize $V(s)$ arbitrarily
  while *policy not good enough* **do**
    forall $s \in \mathcal{S}$ **do**
      forall $a \in \mathcal{A}$ **do**
        $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V(s')$
      end
      $V(s) \leftarrow max_a Q(s,a)$
    end
  end
end

The biggest problem with iterative methods is choosing a stop criterion as a very restrict criterion would make the convergence very slow and a very loose criterion could make the algorithm to converge to a non optimal value. There are many works on literature that treat this problem like WILLIAMS e III (1993) or PUTERMAN (1994). In this work the stop criterion will be specified in each application example and explained as needed.

Updates like those used in this method is known as *full backups* since they use information from all possible successor states. The computational complexity of this algorithm, per iteration, is quadratic in the number of states and linear in the number of actions making it hard to compute in systems with a large variety of states.

Another algorithm needed for the understanding of the next steps is the *policy iteration* (presented below) that manipulates the policy directly instead of the

indirect value iteration method where only the value function is found.

Function *Policy_Iteration* **is**
  choose and arbitrary policy π_0
  while $\pi_t \neq \pi_{t+1}$ **do**
    $V_t(s) \leftarrow R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathbb{S}} T(s, \pi_t(s), s') V_t(s')$
    $\pi_{t+1} \leftarrow \arg\max_a \sum_{s' \in \mathbb{S}} T(s, \pi_t(s), s') V_t(s')$
  end
end

$V_t(s)$ is called a *Bellman Equation* and solving that problem means solving n equations in n unknowns.

3.1.4 Learning a policy

Section 3.1.3 cited methods for obtaining an optimal policy for an MDP assuming that $T(s, a, s')$ and $R(s, a)$ are known but what happens to the agent when it has to take an action without knowing future rewards and the probability function? In this case the only possible think to do is to interact with the environment and tries to compute an immediate reward.

According to KAELBLING *et al.* (1996) there are two methods that can be used:

- **Model-free:** Learn a controller without learning the model
- **Model-based:** Learn the model and use it to derive a controller

The simplest strategy would be to let the agent interact continuously with the environment with their chosen policy and checks if at the **end** its mission have been accomplished and give it the correct reward. The problem in this approach is that, sometimes it is difficult to determine what the **end** would be and if the agent does not perform correctly the task it would be impossible to know where the policy should be corrected and if the initial policy was the optimal one.

RL techniques suggests *temporal difference methods* that uses value iteration to adjust the estimated value of a state based on the immediate reward and the estimated value of the next state.

A well known algorithm is called *adaptive Heuristic Critic* that is composed by two blocks, the critic (AHC) and the RL, the connections of these two blocks are presented in figure 3.6.

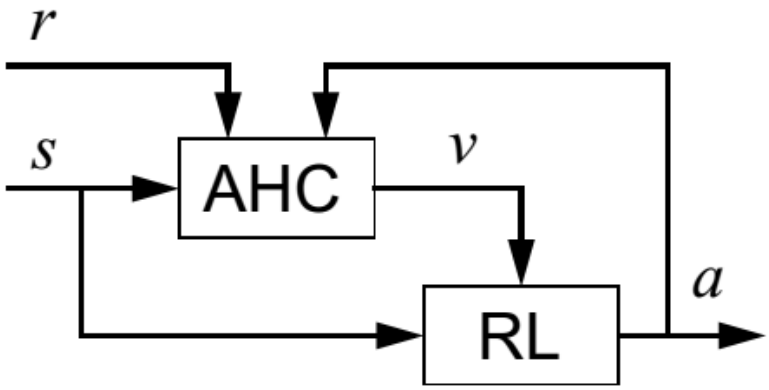

Figure 3.6: Adaptive heuristic critic algorithm[5]

In this algorithm the RL block will act in order to maximize the heuristic signal v instead of instantaneous reward r. The AHC, on the other hand, uses the reinforcement signal r to learn to map states to their expected discounted values given that the chosen policy is being executed in the RL component.

So the AHC learns the value function V_t for the policy π implemented by the RL, then, given this value function the RL component learns a new policy π' that maximizes it.

For the critic to learn the value of a policy Sutton's TD(0) algorithm is used (SUTTON (1988)) and its update rule is expressed by

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s)) \tag{3.8}$$

where α is a learning rate.

For the learning process the *experience tuple* $< s, a, r, s' >$ where s is the agent's state before the transition, a is its chosen action, r is the reward received and s' the resulting state. When a state s is visited its estimated value is updated to be closer to $r + \gamma V(s')$ where $V(s')$ is the estimated value of the actually occurring next state and $r + \gamma V(s')$is a sample of the value of $V(s)$. The main difference between this algorithm and the Value iteration is that the sample is collected from the environment instead of a known model.

There is a more general class of algorithm called TD(λ) with the same update rule but instead of updating only the visited state , it updates recent visited states. The more general one is TD(1) that updates every visited state. The larger the value of *lambda*, the more computationally expensive it is but it often converges faster.

[5]Extracted from KAELBLING *et al.* (1996)

Q-learning

WATKINS (1989) unified the two blocks of the *Adaptive heuristic algorithm* in the Q-learning algorithm that is typically easy to implement and because of that one of the most used in RL works found on literature.

Let's introduce a new function $Q^*(s,a)$ that is the expected discounted reinforcement of taking action a in state s by choosing the actions optimally. V^* is the value of s assuming that the best action is taken and so $V^*(s) = \max'_a Q^*(s',a')$. The recursively form of $Q^*(s,a)$ can be written as

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathbb{S}} T(s,a,s') \max_{a'} Q^*(s',a') \tag{3.9}$$

and the optimal policy π^* can be written as

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{3.10}$$

As the action is explicit in the Q function it becomes easy to estimate the Q-value online using the TD(0) method, but more than that it is easier to define the policy. The Q-learning update rule using the same $< s,a,r,s' >$ experience tuple is

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{3.11}$$

where α is the learning rate.

WATKINS (1989) proved that if each action is executes in each state an infinite number of times on an infinite run and α is decayed appropriately the Q values will converge to Q^*. Another advantage of Q-learning is that it is *exploration insensitive* meaning Q values will converge to the optimal values independent on how the agent behaves while the data is being collected as long as every $< s,a >$ pair is visited often.

This work will use the Q-learning algorithm as they are easy to implement and there are plenty of work on literature to support its implementation.

Chapter 4

Mission Planning

For the intent of this work a **mission plan** is a sequence of command primitives concatenated to accomplish a job given by a **Mission Description**.

In a ROV the mission is given to a **Human Operator** that has the role to elaborate a **Mission Plan** using human like primitives like `go forward`, `go to depth 50m`, `inspect ground` and `check equipment status`, on a mission planning phase. This Plan is pre-scripted based on what the operator thinks it is necessary to accomplish the mission. After that, the operator starts the mission by launching the ROV and manually controls it trying to follow the plan and in case something goes wrong or different from the plan, the operator has autonomy to take new decisions, modify the plan and take actions that would make it accomplish the mission. Figure 4.1 shows how the information cycles occur in a ROV architecture.

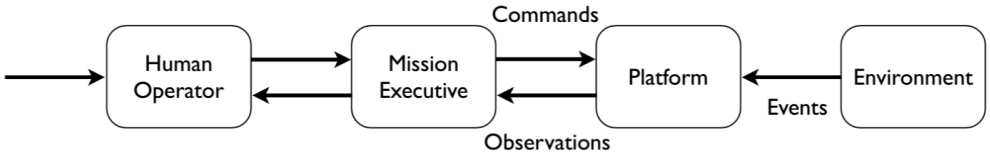

Figure 4.1: Information cycle in an ROV [1]

This kind of architecture uses the human knowledge and SA_H in order to determine the mission success.

When talking about going autonomous, a first step to be considered is to give some intelligence to the vehicle so the operator does not need to execute the whole mission. Figure 4.2 shows a representation on how information cycles occur on the most common architectures like those shown in FOSSEN (1994), RIDAO *et al.* (1999) and YUH (2000).

[1] Modified from PATRON (2010)

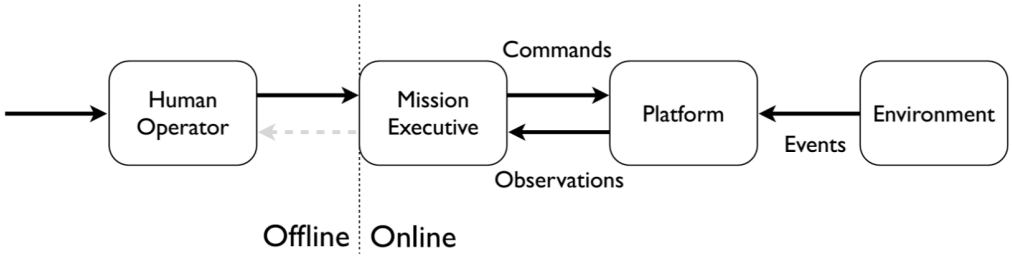

Figure 4.2: Information cycle in a vehicle with some autonomy[2]

In this kind of cycle, a **Human Operator** receives the mission description, process it in system primitives creating the mission plan and program it in the **Mission Executive**. **Mission Executive** sends *C*ommands and receives *O*bservations from the platform that are continuously receiving *E*vents from the environment. Different from the ROV loop (figure 4.1, in this loop the **Human Operator** may not receive information about the mission status letting the mission control to be executed by the vehicle without any surveillance. In this case it is important that the **Human Operator** computes a certain amount of pre-scripted situations that he believes could happen during the mission and let the vehicle to execute the steps of the mission plan converted in system primitives. The main problem on this kind of approach is that the lack of a decision module can cause a mission to abort unexpectedly.

More recently, PATRON (2010) proposed a modification in the cycle to include the decision making. This approach was motivated by the fact that two problems were affecting the the effectiveness of the decision loop. The first one is that *Orientation* and *Observations* should be linked together so the **new** *Observations* would be placed in context. The second one is that the decision and action should be iterating continuously. To solve this two additional components were added to the loop (see figure 4.3). In this configuration the **Human Operator** does not participate on the loop, the mission is passed directly to an offline **Mission Generator** that process it in system primitives creating the mission plan and program it in the **Mission Executive**. **Mission Executive** sends *C*ommands to the platform that are continuously receiving *E*vents from the environment and sending its *O*bservations to the **Status Monitor** that sends information to the **Mission Adapter**. In this case the **Mission Adapter** took the place of the **Human Operator** sending a new plan to the **Mission Executive** when needed. The **Mission Adapter** uses MDP to adapt the initial mission plan.

[2]Modified from PATRON (2010)

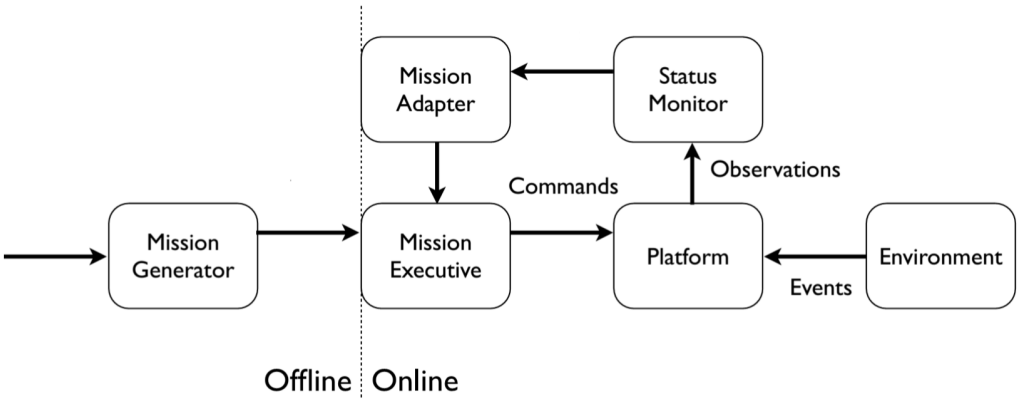

Figure 4.3: Information cycle in an AUV[3]

4.1 Adaptive Mission Planning

The purpose of this section is to show how the mission should be adapted during its execution.

Figure 4.4 shows the proposed mission execution flow in LUMA system. An **Initial Mission Plan** is passed to the system by the operator. An **Action Selector** gets the next primitive to be executed and make a capability check before its execution. This check is necessary to evaluate if the action can be executed. For example, suppose that the mission to be executed is a temperature profiling over a surface installed 300 m depth in the ocean. After reaching the surface, the primitive that appear in the plan is `check_temperature`, if the temperature sensor fails the check capabilities would show that the accomplishment of this primitive is impossible so the robot should try to replan the mission. As the robot does not have other means to check the temperature, like a thermal camera, the replaning is not possible and the `emergency_surface` primitive is executed. Note that if the thermal camera is an option, the robot would have to calculate another plan to reposition itself in a way that the use of the camera becomes possible.

[3]Modified from PATRON (2010)

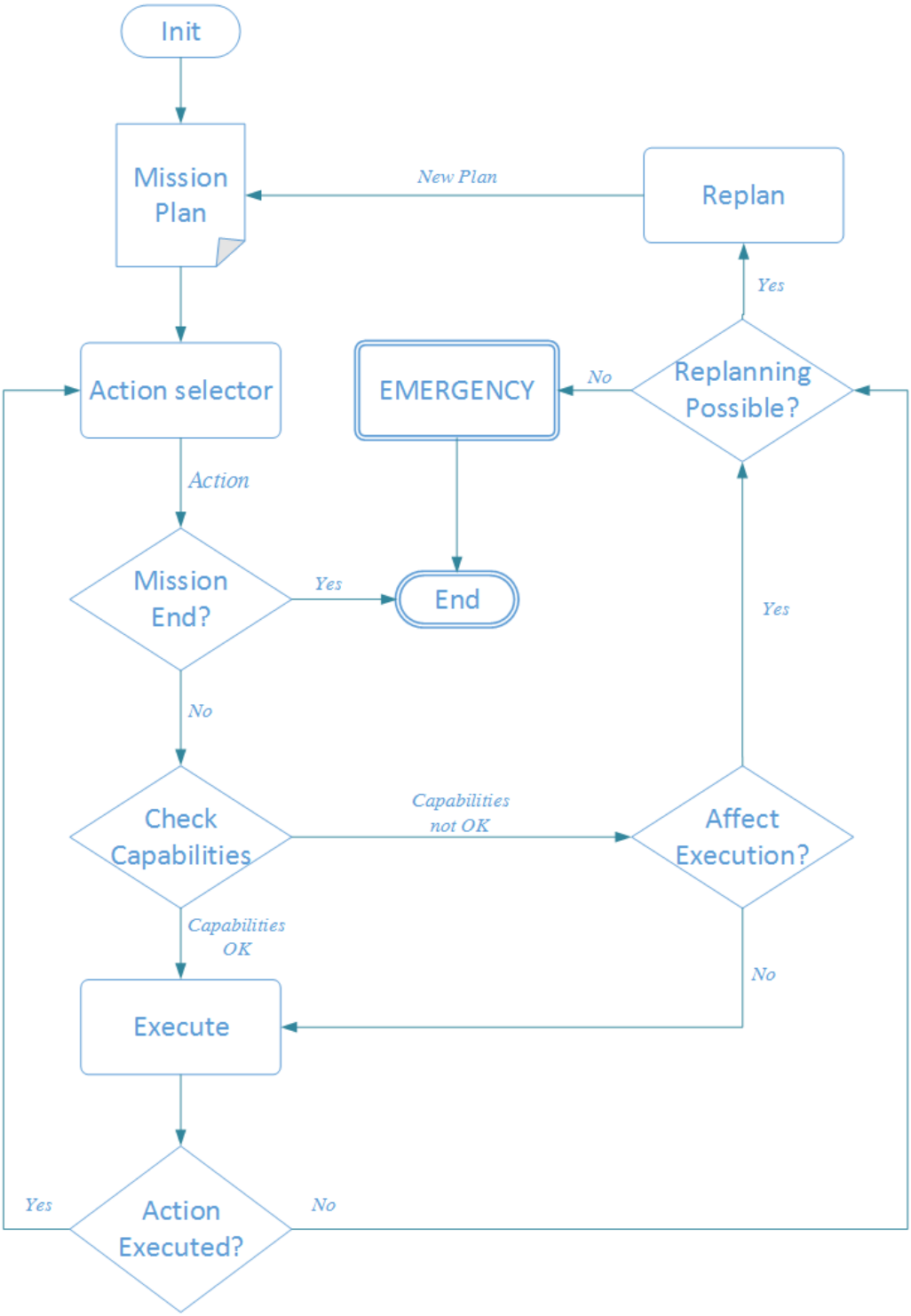

Figure 4.4: Execution Flowchart

The **Replan** block is responsible to make a new plan based on the system capabilities, the mission description and the world model present in LUMA architecture. The replanner uses MDPs in the World Model to check the optimal policy that would solve the mission. As the World Model may not be updated, modeled or perceived correctly by the vehicle the new mission plan can result in an impossible sequence of actions given that the robot can find itself in an environment completely

different from the one it is expecting. World Model Update is a big challenge and will not be treated in this work.

4.2 LUMA Mission Description

LUMA was firstly designed as an ROV for intake pipeline inspections, for that all equipments and mechanical characteristics were chosen to accomplish that mission. After the robot was fully functional and tested, it was adapted for Antarctic survey missions, where some of the mechanics were redesigned and the need of new equipment became imminent.

More than that, in recent years GSCAR was consulted for the possibility of using the ROV to register the whale migration in Abrolhos Archipelago in the southern coast of Bahia state in the northeast of Brazil.

The objective of this session is to describe how the Mission Control System (MCS) should act in order to execute the mission. In JUNIOR (2014) developed the first MCS for the HROV LUMA. The following primitives were developed:

- `heading` - Turns the vehicle along the z axis and maintains the direction during the movement;
- `depth` - Goes to the desired depth;
- `surface` - Goes to the surface;
- `emergencySurface` - Goes to the surface if something goes wrong that could put in danger the robot integrity or the mission;
- `goto` - Goes to the $< x, y >$ position;
- `initializeVehicle` - Initializes the vehicles equipments;
- `stopVehicle` - Stops the Data Acquisition and the Vehicle that could put in danger the robot integrity or the mission;
- `alarm` - Verify the status of every element on the vehicle and generates an alarm if any of them are wrong;

Using these primitives JUNIOR (2014) completed a pipeline tracking mission with and without maritime current in a simulated environment. All the mission have been pre-programmed and the actions in case something went wrong were defined before the robot started the mission which is and `emergencySurface` primitive.

For the purpose of this work 2 missions are going to be considered where the MDP will be applied:

- A pipeline tracking mission (fixed target)
- A whale tracking mission (moving target)

In the **pipeline tracking mission**, an initial plan will be passed to the robot, when a problem occur during the plan, the robot should calculate a new plan and try to execute it. A camera loss, a counter flow and an obstacle will be considered in the simulations.

In the whale tracking mission the robot should wait for a moving target to reach its field of vision and then approximate the target and stay inside a determined area without hitting the target. For this example, different perception levels will be simulated.

The two missions are going to be simulated and MDP will be applied during the mission so the vehicle should learn how to adapt in case anything unexpected happens.

Chapter 5

Simulation and Results

5.1 Simulation environment

The simulation is programmed using the Brown-UMBC Reinforcement Learning and Planning (BURLAP) java code library, this library was developed by the Computer Science department from Brown University in Rhode Island. It is ideal for the use and development of single or multi-agent planning and learning algorithms and domains to accompany them. At the core of the library is a rich state and domain representation framework based on the object-oriented (OO) MDP (DIUK *et al.* (2008)). The main advantages of using BURLAP is that the program created here can be used in the real robot without the need of adaptation and it has a ROS bridge so it can be connected with LUMA framework.

As the BURLAP does not have an intuitive way to show the results, the same code was developed in MATLAB so the results could be presented here.

5.2 The pipeline tracking problem

The first step for the simulation is defining the world model to be used:

- It is a $30x30x30$ grid world;
- All the primitives proposed by JUNIOR (2014) will be considered;
- The robot always receive a negative reward when it moves unless it is moving through its objective. The battery is discharging while the robot is ON
- When the robot reaches the limits of the environment, the action that would move it outside the environment would move it to the same state

Secondly a reward must be defined while the robot is tracking the pipeline. The proposed one is the following:

- When the robot reaches the pipeline in the entrance point it gets +10
- While tracking it receives +1
- If it drifts 1 state to both the left or right of the pipeline it receives +0.5
- If it drifts 2 state to both the left or right of the pipeline it receives +0.25
- If it drifts more than 2 state to both the left or right of the pipeline it is considered that the camera cannot capture the pipeline image so it gets −2

The first test is a simple follow the pipeline mission, where the operator entered the mission plan and nothing wrong happened. Figure 5.1 shows the state and action evolution for the mission.

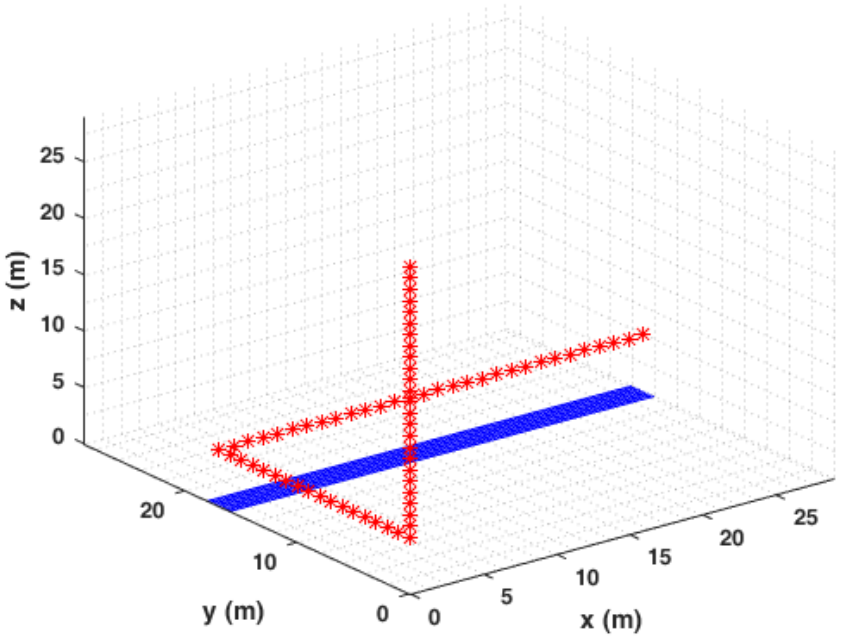

Figure 5.1: State Evolution for a mission without any occurrences. Red star represents the vehicle and Blue lines represent the pipeline.

The first experiment showed that the state evolution makes the mission to be accomplished properly and the offline method for planning is effective when nothing unexpected happens in the environment.

The same mission was executed without the replanner in an environment with a sea current and the mission could not be accomplished as showed in Figure 5.2.

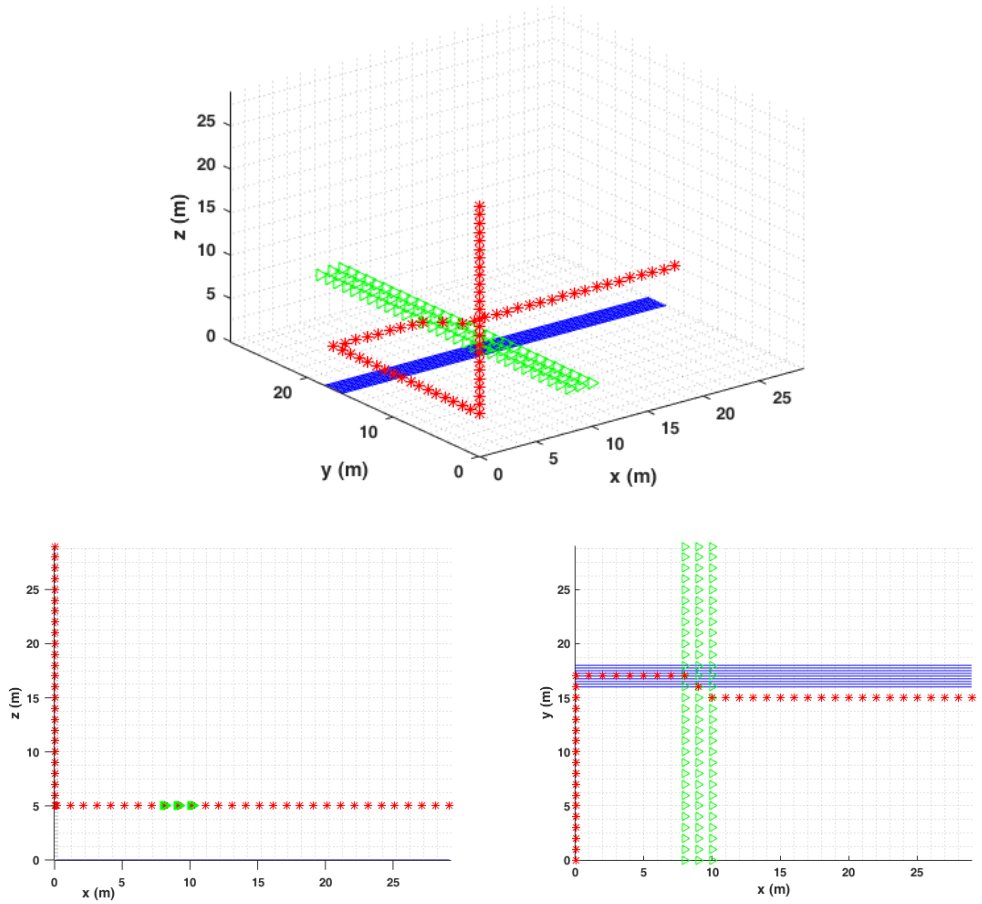

Figure 5.2: State evolution for a mission with a sea current without the replanner. Red star represents the vehicle, Blue lines represent the pipeline and Green triangles represents the sea current.

After turning on the replanner, the mission could be accomplished as the vehicle created a new plan that would take itself out of the sea current and returned to the pipeline through another path. Figure 5.3 show the state evolution for this experiment.

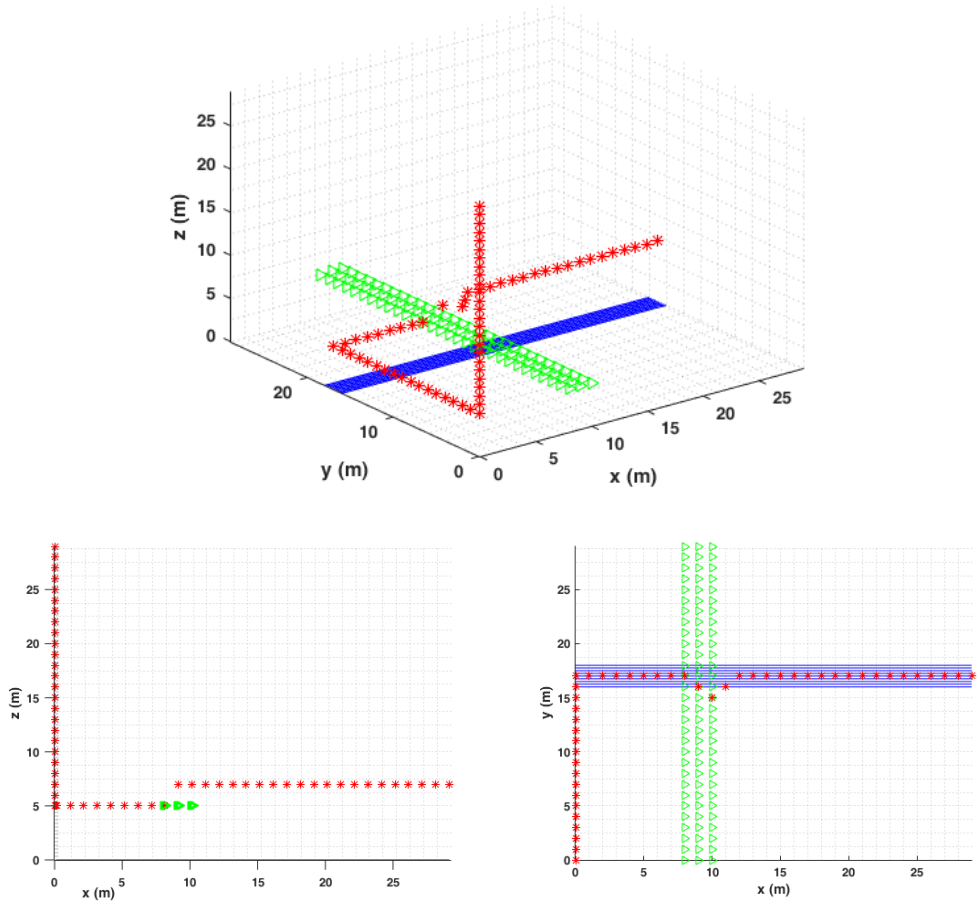

Figure 5.3: State evolution for a mission with a sea current with the replanner. Red star represents the vehicle, Blue lines represent the pipeline and Green triangles represents the sea current.

5.3 The whale tracking problem

The second mission suggested is the whale tracking mission. For this mission the vehicle should wait for the whale to be in its camera viewfinder and starts to follow the whale. Four experiments will be conducted where in each one the perception level of the whale will be modified. In the first one there is a 100 % chance that the whale is perceived by the robot and a 25 % decreased chance will occur in the following tests with the last one with a 25 % probability for the vehicle to perceive it. Only when the whale is perceived the robot starts to follow it. If during the mission the robot looses the capability to track it, it would return to the start point.

5.4 and 5.4 shows the state evolution for this problem.

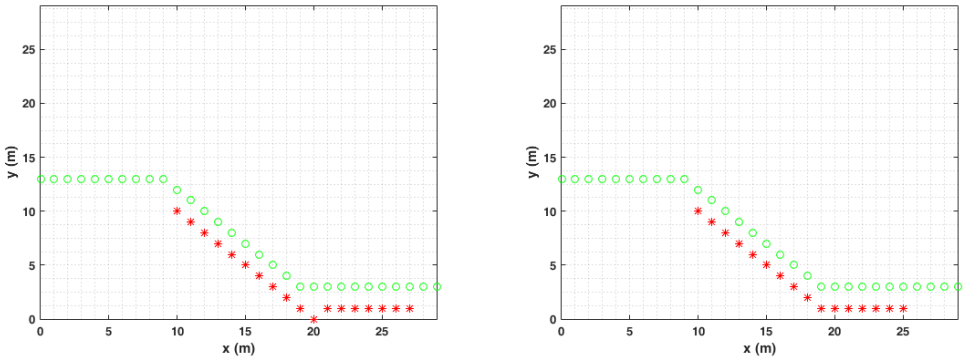

Figure 5.4: Whale tracking problem with perception level of 100%(Right) and 75%(Left). Red star represents the vehicle and green circles represent the whale

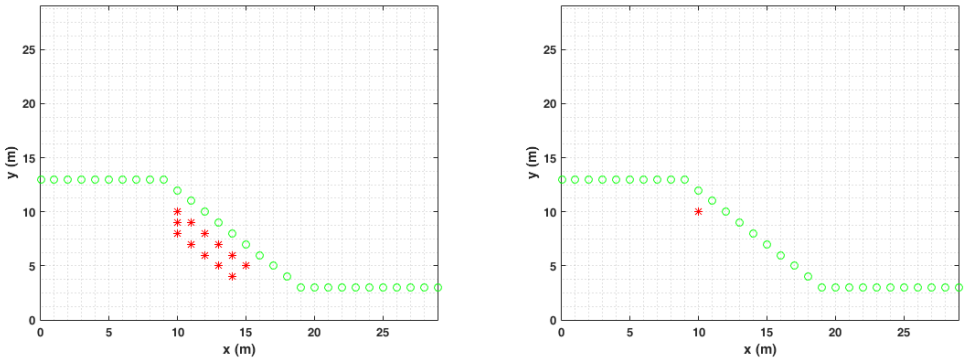

Figure 5.5: Whale tracking problem with perception level of 50%(Right) and 25%(Left). Red star represents the vehicle and green circles represent the whale

Note that with 25% perception level, the robot starts to track it but it looses the whale due to distorted perception, like in low visualization conditions and so it returns to the start point. With 50% perception level, the robot perceives the whale but only a few states after it passed by making the robot to follow it with a delay if compared with the 75% and 100%, wich gave the same behaviour.

5.4 Results Discussion

The results presented in this section showed that it is possible to implement a mission replanning for an autonomous vehicle. Although the underwater environment is much more aggressive and uncertain than the environment presented here, research in perception and robust control would make all the theory tested here to be perfectly applied in Underwater Autonomous System. The application of this theory in LUMA can present a high cost as the investment in new sensors and algorithms is required to improve the system perception and make it act more likely the behaviour presented in the simulated environment.

Chapter 6

Conclusion and Future Work

This work introduced some of the components needed to make a system autonomous. Although many other concepts are needed to make a system completely autonomous the concepts presented here showed to be an effective start on this area of research.

This chapter will review the contribution and the results achieved by this work.

6.1 Conclusion

As mentioned during the whole work, autonomous system are a wide area of research, there are lots of new concepts and new tools are been created to solve the various problems that appear every day.

Concepts like artificial intelligence, perception, situation awareness and others that tend to imitate human behavior are becoming common turning autonomous systems into more and more multidisciplinary area where studies on biology, psychology, neuroscience, statistics and computer science are changing the quality of the new control systems.

Studying human and animal behavior combined with the computer capabilities of manipulating lots amount of data is showing to be a key thing to solve almost every problem presented.

The new control architecture implementation was simulated and showed to be functional. Although not every module presented in the control architecture were implemented, the ones that were made quite a good job in giving some autonomy to the vehicle.

The assumptions on the environment, although very restrict, can be easily loosen with the implementation of the modules proposed for the architecture. Perception is a very important characteristic that has to be explored and upgraded in order to achieve higher levels of Situation Awareness and Decision Making.

RL applied to underwater robotics showed that it is possible to deal with unknown environments if the correct reward rules are given to the robot. MDP applied in Mission replanning showed to be very effective in solving the problem besides the high computational load.

The Q-learning algorithm used in this work, despite been very simple to implement needs lots of computational capabilities that can be very difficult to deal when implemented in real robots, other algorithms should be tested.

6.2 Future Work

As a second effort in turning LUMA autonomous, lots have been reached through this work and new challenges were presented opening new opportunities for research.

The implementation of the **Perception** module and ways to improve the **Situation Awareness** through the use of **Knowledge Frameworks** should be a priority in order to make the control system more reliable.

All the work presented here were simulated so its implementation on the robot should be done shortly. The development using BURLAP facilitates the implementation on the mission replanning block as it is ROS compatible. Besides that all the control system architecture should be executed.

An important decision should be taken between the use of the Petri Nets presented in JUNIOR (2014) and the SMACH system presented in DE FREITAS (2016). Tests should be done considering system stability and integration.

Other RL algorithms should be tested to minimize computational load so the mission replanning module does not represent risk to the mission control system making the computer almost all the time dedicated to the execution of the Q-Learning algorithm. Replanning should be a fast operation seen that the robot cannot stay still for a long time waiting for the calculations to be done. The overview on RL algorithms presented *dyna* as a good alternative but others should be considered.

Of course there are a lot of future works to be done considering that Autonomous Systems is a growing area on todays research and the implementation and tests of this new techniques in LUMA missions in Antarctic will be a considerably push on the studies on underwater autonomous systems on COPPE research and the first step for a journey that could include, for example, the World Climate Research Programme (WCRP) Polar Challenge opened since 2019 where Autonomous Underwater Vehicle have to complete a 2000 km continuous mission under the sea ice.

Bibliography

ALBUS, J., 1991, "Outline for a theory of intelligence", *IEEE Transactions on Systems, Man, and Cybernetics*, v. 21, pp. 473–509.

ARKIN, R. C., BALCH, T., 1997, "Aura: Principles and practice in review", .

BELLMAN, R., 1957, *Dynamic Programming*. Princeton University Press.

BERTSEKAS, D. P., 1995, "Dynamic Programming and Optimal Control", .

BOYD, J. R., 1992, *A discourse on winning and losing - organic design for command and control*. Relatório técnico.

BROOKS, R., 1986, "A robust layered control system for a mobile robot", *IEEE J. Robot. Autom.*, v. 2, pp. 14–23.

DE FREITAS, R. S., 2016, *Arquitetura Híbrida e Controle de Missão de Robôs Autônomos*. Tese de Mestrado, COPPE/UFRJ.

DE OLIVEIRA LIMA, V. F., 2015, *Localização Submarina Utilizando uma Única Referência Acústica via Filtro UKF*. Tese de Mestrado, COPPE.

DIUK, C., COHEN, A., LITTMAN, M. L., 2008, "An object-oriented representation for efficient reinforcement learning." In: *Proceedings of the 25th international conference on Machine learning*, pp. 240–270.

DOMINGOS, P., 2015, *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books.

EL-FAKDI, A., 2010, *Gradient-based Reinforcement Learning techniques for underwater robotics behavior learning*. Tese de Doutorado, University of Girona.

ENDSLEY, M. R., BOLTE, B., JONES, D. G., 2003, *Designing for Situation Awareness: An Approach to User-centred Design*. Taylor & Francis Group.

FIKES, R., NILSSON, N., 1971, "STRIPS: A new approach to the application of theorem proving to problem solving", *Artificial Intelligence*, v. 2, pp. 189–208.

FIRBY, R. J., 1989, *Adaptive execution in complex dynamic worlds*. Tese de Doutorado, Yale University.

FOSSEN, T. I., 1994, *Guidance and Control of Ocean Vehicles*. Wiley.

GAT, E., 1991, *Reliable goal-directed reactive control for real-world autonomous mobile robots*. Tese de Doutorado.

JUNIOR, L. M., 2014, *DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE DE MISSÃO ATRAVÉS DAS REDES DE PETRI*. Tese de Mestrado, October.

KAELBLING, L. P., LITTMAN, M. L., MOORE, A. W., 1996, "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence*.

KORTENKAMP, D., SIMMONS, R., 2008, "Robotic systems architectures and programming", .

LAIRD, J., ROSENBLOOM, P., 1990, "Integrating, execution, planning, andlearning in Soar for external environments", *8th Annu. Meeting Amer. Assoc. Artif. Intell.*, pp. 1022–1029.

LANE, D. M., MAURELLI, F., LARKWORTHY, T., etal., 2012, "PANDORA: Persistent Autonomy through Learning, Adaptation, Observation and Replanning", *Proceedings of IFAC NGCUV 2012 - Navigation, Guidance and Control of Underwater Vehicles*.

LYONS, D., 1992, "Planning for reactive robot behavior", .

MIGUELANEZ, E., PATRON, P., BROWN, K. E., etal., 2011, "Semantic Knowledge-Based Framework to Improve the Situation Awareness of Autonomous Robots", *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, v. 23, n. 5 (May), pp. 759–773.

NILSSON, N., 1984, "Shakey the robot", *SRI International*.

PALOMERAS, N., EL-FAKDI, A., CARRERAS, M., etal., 2012, "COLA2: A Control Architecture for AUVs", *IEEE JOURNAL OF OCEANIC ENGINEERING*, v. 37 (October), pp. 695–716.

PANIAGUA, I. L., 2007, *A Foundation for Perception in Autonomous Systems*. Tese de Doutorado, UNIVERSIDAD POLITÉCNICA DE MADRID.

PAPADIMITRIOU, G., LANE, D., 2014, "Semantic Based Knowledge Representation and Adaptive Mission Planning for MCM Missions Using AUVs scholars", .

PATRON, P., 2010, *Semantic-based adaptive mission planning for unmanned underwater vehicles*. Tese de Doutorado, Ocean Systems Laboratory School of Engineering and Physical Sciences Heriot-Watt University, April.

PUTERMAN, M. L., 1994, *Markov Decision Processes - Discrete Stochastic Dynamic Proggraming*. John Wiley & Sons, Inc.

RIDAO, P., BATLLE, J., AMAT, J., etal., 1999, "Recent trends in control architectures for autonomous underwater vehicles", *International Journal of Systems Science*, pp. 1033–1056. doi: 10.1080/002077299291895.

SANTOS, I. H. F., RIBEIRO, G. M., COUTINHO, F., etal., 2013, "A Robotics Framework for Planning the Offshore Robotizing Using Virtual Reality Techniques", *OTC Brasil*.

SUTTON, R. S., 1988, "Machine Learning". cap. Learning to predict by the method of temporal differences, pp. 9–44.

TURNER, R. M., 2005, "Intelligent mission planning and control of autonomous underwater vehicles", *Proc. Int. Conf. Autom. Planning Scheduling*.

URE, N. K., CHOWDHARY, G., HOW, J. P., etal., 2013, "Health Aware Planning under uncertainty for UAV missions with heterogeneous teams". In: *Control Conference (ECC), 2013 European*, pp. 3312–3319, July.

WATKINS, C., 1989, *Learning from delayed rewards*. Tese de Doutorado, Kings College.

WILLIAMS, R. J., III, L. C. B., 1993, *Tight performance bounds on greey policies based on inperfect value functions*. Relatório técnico, Northeastern University.

YUH, J., 2000, "Design and Control of Autonomous Underwater Robots: A Survey", *Journal of Autonomous Robots*, v. 8(1), pp. 7–24.